2010

A Structured ASIC Approach to a Radiation Hardened by Design Digital Single Sideband Modulator for Digital Radio Frequency Memories

Thomas B. Pemberton
Wright State University

A STRUCTURED ASIC APPROACH TO A RADIATION HARDENED BY DESIGN DIGITAL SINGLE SIDEBAND MODULATOR FOR DIGITAL RADIO FREQUENCY MEMORIES

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Engineering

By

Thomas B. Pemberton
B.S.C.E, Wright State University, 2008

2010
Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

June 11, 2010

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Thomas B. Pemberton ENTITLED A Structured ASIC Approach to a Radiation Hardened by Design Digital Single Sideband Modulator for Digital Radio Frequency Memories BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Engineering.

John M. Emmert, Ph.D.
Thesis Director

Kefu Xue, Ph.D.
Department Chair

Committee on Final Examination

John M. Emmert, Ph.D.

Ray E. Siferd, Ph.D.

Saiyu Ren, Ph.D.

Stephen L. Hary, Ph.D.

John A. Bantle, Ph.D.
Vice President for Research and Graduate Studies
and Interim Dean of Graduate Studies

Abstract

Pemberton, Thomas B. M.S. Egr., Department of Electrical Engineering, Wright State University, 2010. A Structured ASIC Approach to a Radiation Hardened by Design Digital Single Sideband Modulator for Digital Radio Frequency Memories.

Digital Radio Frequency Memories (DRFM) are widely used as modules in digital signal processing. These modules can provide several forms of signal manipulation and storage capabilities. With single event effects caused by environmental radiation the need for a radiation hardened DRFM is increased. Typical radiation hardening involves the use of specialized foundries utilizing proprietary CMOS libraries that are expensive to build or adding lead packages around a chip that is expensive and add weight to the chip. An alternative radiation hardening technique is to utilize a radiation hardened by design library. This library includes digital gates that have been hardened by the use of guard rings, reverse body bias or other methods. With the use of the hardened library, commercial synthesis tools can create a structural Verilog output from the behavioral VHDL design. The radiation hardened by design circuit will be larger than a non-hardened design, but can be fabricated using standard foundries.

This research also takes advantage of current advancements of commercially available software and designs that have led to a structured ASIC approach for fabricating a design. This structured ASIC approach fabricates a design in two stages. The first stage is the transistor and bottom metal layers with the second stage being the top metal layers. Silicon wafers can be fabricated in bulk using the first stage of uncommitted logic with separate top metal layer masks applied to commit the logic to a specific design. A radiation hardened by design standard cell library was used to create the Structured ASIC standard cells and will allow production of radiation hardened circuits with a short design time.

For this research, a generic frequency shifting DSSM is proposed that targets a radiation hardened by design Structured ASIC to deliver performance in processing as well as radiation hardening at both the transistor level and gate level. This research produces a parameterizable DSSM VHDL design that can be easily modified to produce a DSSM with various signal processing and storage capabilities with minimal modifications. The designed DSSM was tested on an FPGA board for prototyping, but was ultimately targeted for the radiation hardened by design structured ASIC. The design created through this research was compared to a non-hardened DSSM using a similar CMOS process for area, power, speed and Spur Free Dynamic Range.

Table of Contents

Page

Page

List of Figures

List of Tables

List of Abbreviations

Abbreviation		Page
DDS	Direct Digital Synthesizer	10
IP	Intellectual Property	11
ROM	Read-Only Memory	11
RAM	Random Access Memory	11
IIR	Infinite Impulse Response	12
FIR	Finite Impulse Response	13
CMOS	Complimentary Metal Oxide Semiconductor	15
Rad	Radiation Absorbed Dose	15
SI	International System of Units	15
TID	Total Ionizing Dose	16
SEE	Single-Event Effects	16
SEU	Single-Event Upset	16
SET	Single-Event Transient	16
SEL	Single-Event Latchup	16
MOSFET	Metal Oxide Semiconductor Field Effect Transistor	16
MBU	Multiple-Bit Upset	16
SEFI	Single-Event Functional Interrupt	16
BJT	Bi-Polar Junction Transistor	17
RBB	Reverse Body Bias	19
SOI	Silicon-On-Insulator	19
SRAM	Static Random Access Memory	19
TSMC	Taiwan Semiconductor Manufacturing Company	19
ECC	Error Checking and Correction	21
CRC	Cyclic Redundancy Check	22
DFF	D-Type Flip Flop	24
GaAs	Gallium Arsenide	25
MMIC	Monolithic Microwave IC	25
HDL	Hardware Description Language	30

Abbreviation		Page
WTM	Wallace-Tree Multiplier	33
CSA	Carry Save Adder	33
CLA	Carry Look-Ahead Adder	33
RCA	Ripple Carry Adder	34
PCI	Peripheral Component Interconnect	39
STA	Static Timing Analysis	43
GDSII	Graphic Data System II	43
FFT	Fast Fourier Transform	47

Acknowledgements

I would first like to thank God for giving me all of the abilities that I possess. Next I would like to thank my parents for always encouraging me and giving me all the support that I could ever ask for. A big thanks to my wonderful wife for putting up with the long hours of work on this thesis and all of the continued support that she gives. I would like to give thanks to my thesis committee: Dr. Marty Emmert, Dr. Ray Siferd, Dr. Saiyu Ren and Dr. Steve Hary. Also, a thanks to the New Electronic Warfare Specialists Through Advanced Research by Students (NEWSTARS) program for sponsoring my research.

Thomas B. Pemberton

A Structured ASIC Approach to a Radiation Hardened by Design Digital Single Sideband Modulator for Digital Radio Frequency Memories

I. Introduction

In the aerospace industry a Digital Radio Frequency Memory (DRFM) can serve many purposes in an Electronic Warfare (EW) environment. A DRFM can manipulate the Electromagnetic (EM) spectrum by modifying the frequency of a captured signal. This signal can also be captured, stored and then re-transmitted with or without any modifications and used as a part of an Electronic Countermeasure (ECM) system.

1.1 Motivation

In the aerospace field, a DRFM can be used for various purposes to modify an incoming Radio Frequency (RF) signal. Since the DRFM will be operating in the EW environment, the DRFM will be exposed to different amounts and types of radiation effects. In the aerospace industry, the added cost of radiation hardening an Integrated Circuit (IC) can be substantial. Few foundries exist that can create a hardened by process IC which not only drives up cost but also increases the design cycle due to long fabrication times. Another approach to hardening an IC is a by design approach that takes advantage of several techniques to mitigate radiation effects in an IC. This radiation hardening by design approach can potentially save money versus the by process approach, but for fabrication on a Application Specific Integrated Circuit (ASIC) the cost can be substantial if the number of devices is low. To minimize the cost of a hardened by design ASIC, a similarly designed IC can be created on a Structured ASIC (S-ASIC). Implementing a design on a structured ASIC allows for quick design cycles and fabrication times with a much lower cost due to the shared cost of the fabrication. Adding in a radiation hardened by design standard cell library to

the S-ASIC further advances the technology to provide a low cost radiation hardened design that can be quickly fabricated. The S-ASIC provides a lower cost alternative than a normal ASIC and with the radiation hardening by design library used for the digital cells used in the design, provides a low cost radiation hardened IC.

1.2 Research Goal

The goal of this research is to design, implement and test a radiation hardened by design Digital Single Sideband Modulator (DSSM). This design will feature parameterizable bit widths for the design as well as a parameterizable filter length to exploit different spectral purity versus clock frequency trade offs. The design will be designed using Very High Speed Integrated Circuit Hardware Description Language (VHDL) and will be targeted to an S-ASIC as well as prototyped on a Field Programmable Gate Array (FPGA). The prototyping on the FPGA will allow for further analysis of the overall DRFM design before the addition of the radiation hardened library.

1.3 Research Approach

This research will focus on the DSSM for a DRFM as well as radiation effect mitigation. The DSSM in this research will assume a suitable radiation hardened Analog to Digital Converter (ADC) and Digital to Analog Converter (DAC) are available for use. This research will focus on utilizing a radiation hardened by design digital standard cell library to synthesis a parameterizable DSSM for fabrication. This DSSM design will be primarily focused on the S-ASIC, but will be prototyped on a FPGA.

1.3.1 Testing and Analysis. The DSSM designed through this research will be tested for Spur Free Dynamic Range (SFDR) as well as power consumption, maximum clock frequency and area. The design will be simulated as well as prototyped on an FPGA to study the effects of different hardware scenarios such as the input and output bit widths as well as the filter lengths and shifting frequency resolution. The testing on the FPGA was performed on a Virtex FPGA demo board which included

on board ADC and DAC modules. The DSSM simulations includes floating-point as well as fixed-point simulations to provide functionality testing as well as behavioral design, post-synthesis and post-Place and Route (PAR) simulations achieved using a Cadence RTL Compiler, ViAsic ViaPath PAR tool and the Cadence SimVision simulation suite. Also included is a comparison of the DSSM designed through this research to a DSSM created using a similar, but non-hardened, library for any improvement or declination in area, speed, power and dynamic range.

1.4 Document Organization

This thesis document is organized into five chapters. Chapter 1 provides a brief introduction to the thesis, the motivation behind the research and the scope of the research. Chapter 2 gives a detailed background on the theory and aspects of the research that may not be readily known by the reader. Chapter 3 provides the methodology taken during this research. This chapter provides a detailed description from the mathematical properties used in the design to the hardware design itself. Chapter 4 provides the results gathered through each of the simulation and testing phases. Finally, Chapter 5 discusses the final conclusions drawn from this research as well as future research projects that can expand upon this research.

II. Background and Theory

A DRFM may contain different modules that can modify an incoming digitally sampled signal using various methods. One method creates a frequency shift using a DSSM that emulates a Doppler shift in either the positive or negative direction from the center frequency. This manipulation may either be produced off-line or in real-time. If the DRFM is to operate in real-time then the datapath needs to either be pipelined or partitioned and the various filters split to provide high throughput. If the incoming signal is to be processed off-line then the requirements for the DRFM are not as stringent and many more features can be added. The key aspects of the DRFM as it applies to EW and for this research is the bandwidth of the DRFM, storage space, frequency modulation and targeted platform. Each of these key aspects will be addressed in this chapter.

2.1 Aspects of a Digital Radio Frequency Memory

2.1.1 Electronic Warfare. When a DRFM is used in an EW application several parameters need to be taken into account, one of which is the intended operation of the DRFM. The DRFM needs to have the supporting structures around it in the form of an ADC and DAC, as well as a storage medium. Each of these structures can either be incorporated into a single chip, surface mounted onto a common Printed Circuit Board (PCB) or connectorized and placed on multiple PCBs. The type of storage medium is not important to the operation of the DRFM as long as the data can be stored and retrieved within the desired time requirements. The structure of the ADC and DAC is also not critical to the DRFM as long as the sampling rates are sufficiently high to ensure no loss of data and that the bit widths of each are sufficient.

A possible high-level view of a DRFM for the EW environment is shown in Figure 2.1. This figure shows the optional Bandpass Filter (BPF), which can be used to select different bands of interest in the RF spectrum. Also included is the Low-pass Filter (LPF) used to increase the single to noise ratio.

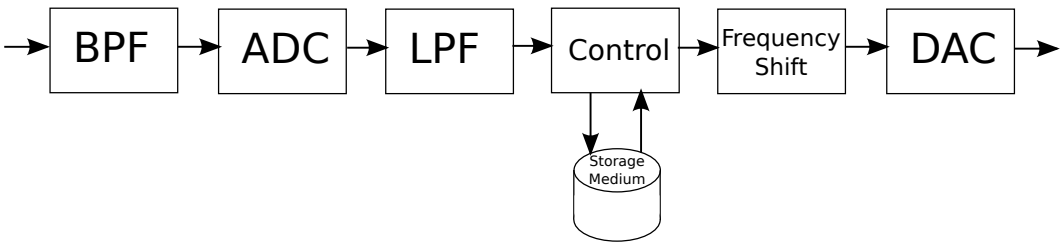


Figure 2.1: High-Level Overview of DRFM

2.1.2 Importance of Bandwidth in a DRFM. The instantaneous RF spectrum bandwidth of the DRFM is extremely important. According to Nyquists' sampling theorem [26] the clock frequency of the ADC that will be sampling the incoming RF signal must meet Equation 2.1 when the incoming RF signal has a bandwidth of β. Through the use of this principal, different Nyquist bands can be created based on the sampling frequency of the ADC, f_s, as shown in Figure 2.2.

$$f_S = 2 \times \beta \tag{2.1}$$

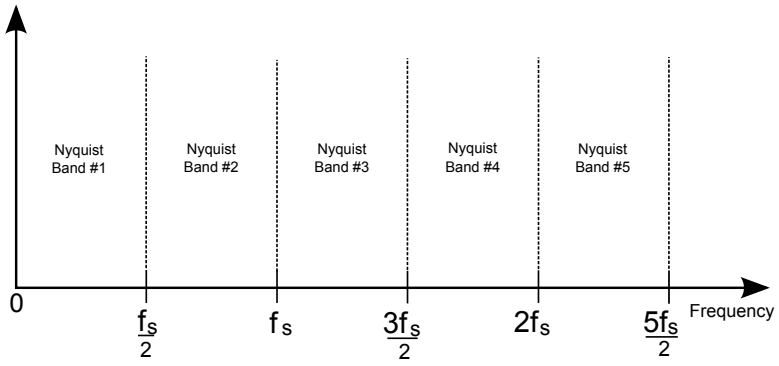


Figure 2.2: Nyquist Frequency Bands

Using the equation for the sampling rate of an incoming RF signal, the ADC sampling rate must be twice that of the bandwidth of interest. Also, the bandwidth of interest plays a crucial role in the amount of storage needed when capturing the incoming RF signal. For instance, if the incoming RF signal has a bandwidth of interest of β and is sampled at a rate of f_S and the ADC has 2^N quantization levels,

the amount of bit storage needed to store one second of data is given in Equation 2.2, but can be generalized for any length of time ΔT in seconds as shown in Equation 2.3. Since storage space is not infinitely large, decisions must be made to ensure that there is ample storage capacity for all signals of interest.

$$\text{Storage Space} = f_S \times N \tag{2.2}$$

$$\text{Storage Space} = f_S \times N \times \Delta T \tag{2.3}$$

2.2 Modulators

The key focus is the frequency shifting of a received signal. Several modulators exist, each with their own benefits and drawbacks. Two of the most common analog modulators are the Double Sideband Modulator (DSM) and the Single Sideband Modulator (SSM). Both modulation techniques uses the trigonometric identity as shown in Equation 2.4 for multiplying two sinusoidal signals.

$$cos(u) * cos(v) = \frac{1}{2}[cos(u - v) + cos(u + v)] \tag{2.4}$$

If you assume a modulation signal $m(t)$, with modulating frequency of f_m, equal to

$$m(t) = cos(2\pi f_m t) \tag{2.5}$$

and the signal to be modulated is $c(t)$, with a center frequency of f_c, equal to

$$c(t) = cos(2\pi f_c t) \tag{2.6}$$

then the DSM signal is equal to,

$$m(t) * c(t) = \frac{1}{2}[cos(2\pi(f_m - f_c)t) + cos(2\pi f_m + f_c)t)] \tag{2.7}$$

SSM is different from that of DSM, in which it requires a separate signal that is 90° out of phase relative to the incoming signal. The original signal and the phase shifted signal are referred to as the I and Q signals. If the I signal has a center frequency of f_c, it would be defined as,

$$I(t) = cos(2\pi f_c t) \tag{2.8}$$

then the Q signal is defined as,

$$Q(t) = cos(2\pi f_c t + \phi) \tag{2.9}$$

with ϕ equal to 90°, the Q signal can be simplified as,

$$Q(t) = sin(2\pi f_c t) \tag{2.10}$$

The original signal can be constructed as $I + jQ$ with j equal to $\sqrt{-1}$. Now assuming there are two modulating signals, both with a modulating frequency of f_m, then with $m_1(t)$ defined as,

$$m_1(t) = cos(2\pi f_m t) \tag{2.11}$$

and $m_2(t)$ defined as,

$$m_2(t) = sin(2\pi f_m t) \tag{2.12}$$

the final SSM signal is defined as,

$$I(t) * m_1(t) + Q(t) * m_2(t) \tag{2.13}$$

which can be expanded into,

$$cos(2\pi f_c t) * cos(2\pi f_m t) + sin(2\pi f_c t) * sin(2\pi f_m t) \tag{2.14}$$

This can be further expanded by using the trigonometric identity shown in Equation 2.4,

$$\frac{1}{2}[cos(2\pi(f_c - f_m)t) + cos(2\pi(f_c + f_m)t)] + \frac{1}{2}[sin(2\pi(f_c - f_m)t) - sin(2\pi(f_c + f_m)t)] \tag{2.15}$$

While the equations are mathematically correct, it may not be clear what the output signal appears to be. Assume the input RF spectrum is captured as pictured in Figure 2.3, then the DSM representation of the shifted signal would be pictured as shown in Figure 2.4 while the SSM representation is shown in Figure 2.5. As can be seen from the figures, the SSM is a more efficient representation of the frequency shifted signal [13, 27]. Since SSM is a more efficient modulation technique, further description on the SSM structure needs to be looked at.

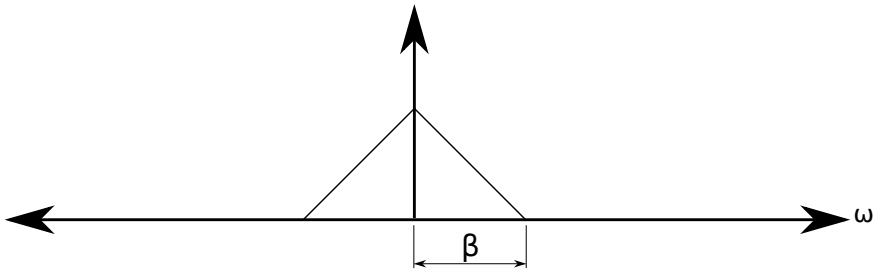


Figure 2.3: Baseband RF Signal

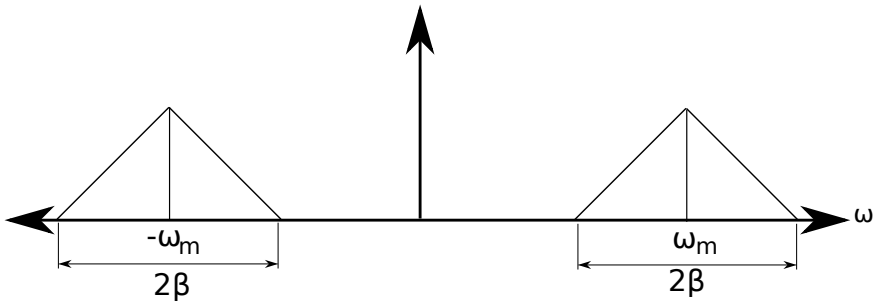


Figure 2.4: RF Signal Modulated using DSM

2.2.1 Analog Single Sideband Modulator. An analog SSM involves a few key components, namely an analog mixer and an analog adder. These two modules will perform the steps needed to create the single sideband functionality. While the analog

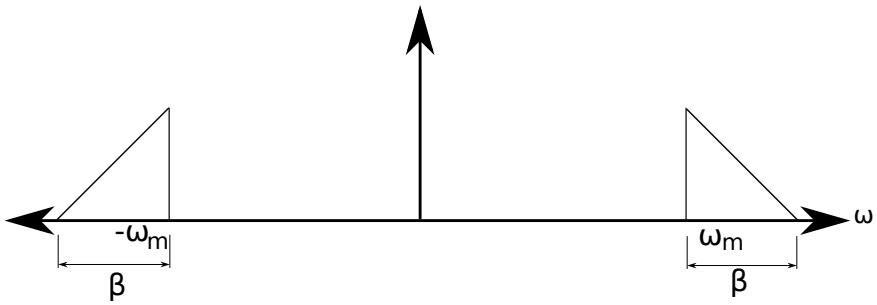


Figure 2.5: RF Signal Modulated using SSM

SSM seems simple, several considerations need to be taken into account. First, the $m_1(t)$ and $m_2(t)$ signals from Equations 2.11 and 2.12 will need to be created. If the desired operating environment requires an IC then the *sin* and *cos* analog waveforms need to be created on chip with some form of oscillator. Since the frequency f_m for both $m_1(t)$ and $m_2(t)$ needs to be adjustable, so does the oscillator. With this condition the oscillator is then converted into a Voltage Controlled Oscillator (VCO) that can be added to the IC. The basic idea behind a VCO is to allow the output frequency (f_m in this case) to be linearly adjustable by an input biasing voltage. A VCO could be created in many different forms, but the concept remains the same.

Along with the VCO consideration there must also be a consideration on the power levels of each of the components and the Direct Current (DC) operating voltages for each of the transistors in the circuit. All of the DC operating voltages along with the layout of the circuit must be created manually which takes a considerable amount of time. A block diagram outlining the flow of the analog signal through an analog SSM is shown in Figure 2.6. Since the DRFM is strictly a digital IC, the analog SSM is not feasible, and a digital approach must be utilized.

2.2.2 Digital Single Sideband Modulator. The Digital Single Sideband Modulator (DSSM) [9] uses digital versions of the same key components that the analog SSM requires (a multiplier and an adder), but they will be digital devices. The multiplier is a digital M-bit multiplier and the adder is a digital N-bit adder, where M and N may or may not be related. The main difference between the analog SSM and

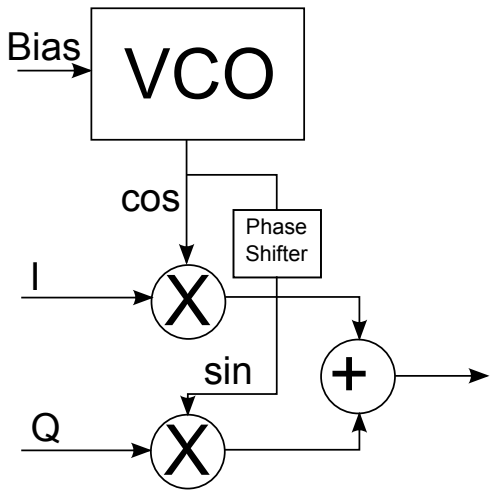


Figure 2.6: Analog SSM Structure

the DSSM is that the analog SSM allows for a doubling of the bandwidth of interest, while the DSSM allows for a slower processing frequency. An important design criteria is the *sin* and *cos* modulating signals $m_1(t)$ and $m_2(t)$. One method for creating these signals would be to create them off chip from an analog signal generator and use an N-bit ADC to digitize the data for use in the digital multipliers. This method is not practical as it needs a separate ADC for the *sin* and *cos* signals and the analog signal generator may not be on the IC as would be the optimum approach. Another approach would be to use a digitizing signal generator and have N Input/Output (IO) pins connected to the IC. This approach, too, is flawed by the use of an external signal generator and also flawed by the use of IO pins for the modulating signals. IO pins on an IC are limited by its size and therefore the IC may not have sufficient available IO pins to allow multiple input signals with a large number of bits. A novel approach is to a Direct Digital Synthesizer (DDS) to generate the digital modulating signals.

2.2.2.1 DDS. There are a few different methods for creating a DDS. The first incorporates a compression algorithm that produces a full wave output signal based on the signals slope [10]. While this method will be capable of producing a signal with the desired frequency with great precision, it is not usually capable of fitting in the amount of area allowed on the IC and therefore another approach must

be taken. The simplest approach to create a DDS uses a quarter-wave approximation technique [33] where one-quarter of a full period is loaded into memory elements and the desired output signal frequency is obtained by the repeated addressing of the memory elements. Figure 2.7 shows the symmetry that is exploited in this DDS method. If the memory elements are read in sequential order to produce region 1, then reading the elements in reverse order will produce region 2. Regions 3 and 4 are produced by inverting the results gathered from regions 1 and 2.

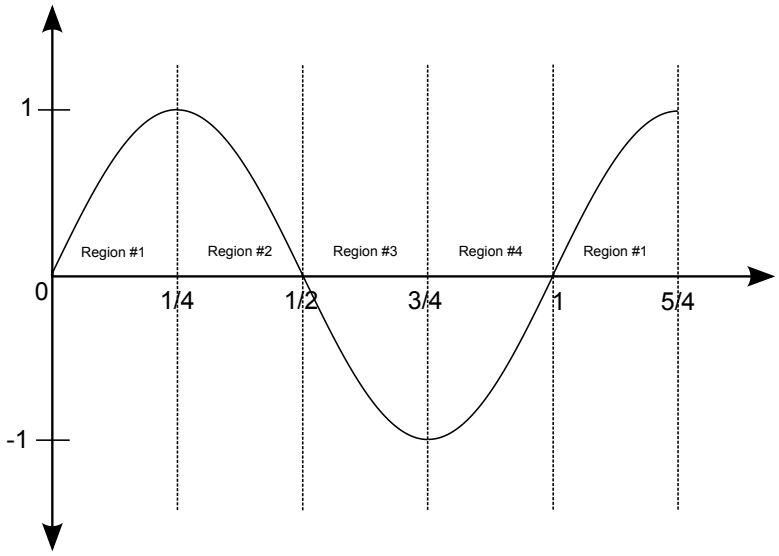


Figure 2.7: Symmetry Exploitation of Quarter-Wave Technique

Many methods exist for storing the sampled signal in memory elements. If the architecture being targeted has Intellectual Property (IP) Read-Only Memory (ROM) or Random Access Memory (RAM), then those can be used to store the sampled signal. If no IP exists for the targeted architecture, then a gate implementation of the memory elements can be constructed. The memory element type used is not so crucial, with the key to producing the desired output signal with the desired modulation frequency being the addressing of the memory elements and the correction circuitry on the output after the memory elements have been accessed. The addressing of the memory elements depends on a few key attributes, one of which is the operating frequency of the memory elements while the others are the desired output frequency

and the addressability of the memory elements. If we assume that the operating frequency of the memory elements is f_o, the desired output frequency is f_m and the memory includes N addressable words, we can define an equation that will produce the incremental factor for the desired output frequency. The k incremental value is constrained by,

$$k \in \mathbb{Z}^+ \tag{2.16}$$

The incremental value k is further constrained by the Nyquist sampling theorem such that there must be at least two samples taken from each quarter-wave and thus has the constraint of

$$k < \lfloor \frac{N}{2} \rfloor \tag{2.17}$$

With these constraints, the frequency resolution of the quarter-wave DDS is defined as a function of the operating frequency f_o and the addressability of the memory elements N. The frequency resolution f_r is defined as,

$$f_r = \frac{f_o}{N} \tag{2.18}$$

With the incremental factor and the frequency resolution of the DDS defined, the output frequency f_m of the DDS is therefore defined as,

$$f_m = k * f_r \tag{2.19}$$

2.3 Hilbert Transform Theory

The Hilbert Transform [15] is one of the basic building blocks that can be used in the DSSM to do the frequency shifting as shown previously in Figure 2.1. The Hilbert Transform creates a $+90^\circ$ phase shifted signal relative to the input signal. In theory, the Hilbert Transform is defined as an Infinite Impulse Response (IIR) Filter.

The Analog IIR Hilbert filter coefficients [8] are defined as,

$$h(x) = \begin{cases} 0 & : |x| < 1 \\ \frac{1}{\pi x} & : otherwise \end{cases}, x \in \mathbb{R} \tag{2.20}$$

The IIR Hilbert filter has a frequency response that is given by,

$$H(f) = \begin{cases} j & : f < 0 \\ 0 & : f = 0 \\ -j & : f > 0 \end{cases} \tag{2.21}$$

With the filter coefficients defined, Figure 2.8 shows what the analytical Hilbert filter coefficients looks like when plotted.

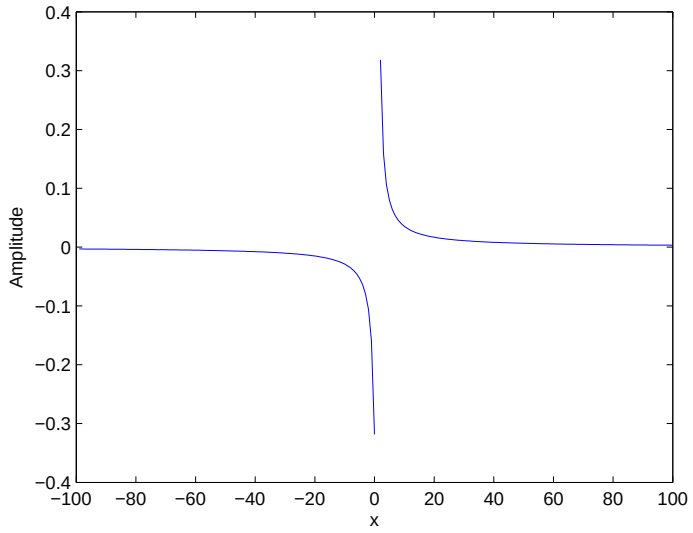


Figure 2.8: Analytical Hilbert Coefficients ($|x| \leq 100$)

Since the IIR filter does not have a closed form output function based solely on the inputs, it must be modified into a Finite Impulse Response (FIR) digital filter of order N.

2.3.1 Digital Hilbert Filter. The key to the digital Hilbert filter is creating a sample-based filter that approximates the analytical Hilbert filter [14]. The

coefficients for the digital Hilbert filter with N+1 filter taps is defined by,

$$h(n) = \begin{cases} 0 & : odd\ n \quad , n \in \{0, 1, 2, ..., N\} \\ \frac{1}{\pi(n-M)} & : otherwise \quad , M = \frac{N-1}{2} \end{cases} \tag{2.22}$$

To satisfy the symmetry of the IIR Hilbert filter, the digital Hilbert filter has the following constraints on the parameter N,

$$\begin{aligned} \tfrac{N-1}{2} &\in \mathbb{Z} \\ \tfrac{N-1}{4} &\in \mathbb{Z} \end{aligned} \tag{2.23}$$

The filter coefficients defined above represent a sampled IIR Hilbert filter with a defined number of samples. The resulting filter coefficients plot resembles Figure 2.9 with 33 filter taps which satisfies both constraints given in Equation 2.23.

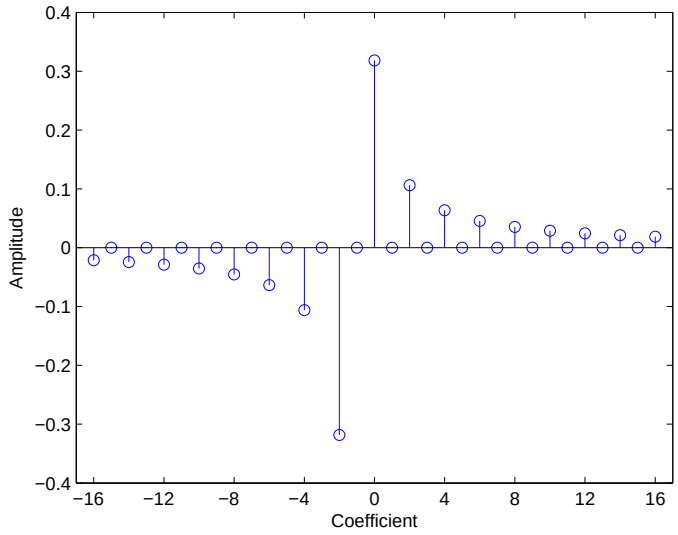
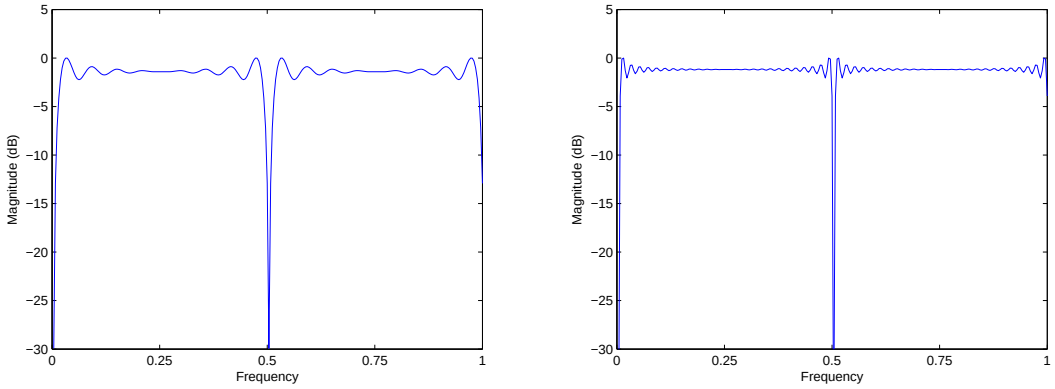


Figure 2.9: Digital Hilbert Filter with 33 Taps

The two key elements of the digital Hilbert filter is the length of the filter and the bit width of the filter coefficients. While the coefficient bit width will help with precision of the sampled analytical Hilbert filter, the overall Hilbert filter length determines what the frequency response of the filter resembles. To show an example,

Figure 2.10(a) shows the frequency response of a 33 tap digital Hilbert filter while Figure 2.10(b) shows the frequency response for a 101 tap digital Hilbert filter. It can easily be seen that with a longer filter, the passband ripple is reduced and has sharper rolloffs at the edge of the passband. While a longer filter will lead to a better frequency response, it will also lead to larger resource usage and a slower clock frequency.



(a) 33 Tap Filter (b) 101 Tap Filter

Figure 2.10: Digital Hilbert Filter Frequency Response Using 256-pt FFT (Normalized X-Axis to f_s)

2.4 Radiation Hardening

Digital Complimentary Metal Oxide Semiconductor (CMOS) devices are susceptible to radiation by various sources, whether anticipated or not. The process to create a digital device that can withstand a certain amount of radiation before corrupt data decreases the performance of the device is known as radiation hardening. Each radiation hardened device will have a defined hardness level at which the device can tolerate a certain amount of radiation without allowing corrupt data to propagate through the device. The standard unit of a device's radiation hardness is given in terms of the Radiation Absorbed Dose (Rad), with 1 Rad equal to the dose of energy absorbed by 1 gram of material. Also defined is the International System of Units (SI) unit of gray (Gy) ($1Gy = 100\ Rad = \frac{1J}{1kg}$). Many radiation hardened digital devices are measured in terms of KRads (1×10^3 Rads) and MRads (1×10^6 Rads). The

two key radiation effects are known as Total Ionizing Dose (TID) and Single-Event Effects (SEE), with SEE including multiple categories of effects. The three main types of SEE are the Single-Event Upset (SEU), Single-Event Transient (SET) and Single-Event Latchup (SEL). Each of these different types will be discussed further.

2.4.1 Total Ionizing Dose. The TID radiation effect occurs whenever the dose of radiation felt by the digital circuit becomes high enough that it begins to modify the operation of a Metal Oxide Semiconductor Field Effect Transistor (MOSFET) within a digital cell. The effect felt on the MOSFET is a shift in the threshold voltage needed for the transistor to be turned on [2, 5, 21]. This shift in the threshold voltage is caused by particles of radiation energy becoming trapped in the silicon dioxide gate insulator. The shift in threshold voltage can cause lower slew rates as well as modify the transition time from high to low voltage (τ_{HL}) or low to high voltage (τ_{LH}). Both of these effects are not harmful to the digital device but can cause inaccurate outputs.

2.4.2 Single-Event Upset. The SEU type of SEE radiation effect occurs in memory devices whenever a single ion interacts with the CMOS transistors in the device and causes a stored binary bit to flip. The SEU is often called a *soft* error in memory devices due to its harmless interaction with the digital devices. The SEU is caused by an ion striking a portion of a CMOS transistor and adding electrons and/or holes into the substrate under a transistor [17]. If this increase in electrons or holes is opposite from the current operation of the transistor, the output of the transistor will be flipped. A pictorial slice of a CMOS transistor receiving a ion strike is shown in Figure 2.11.

Due to the layout of most memory devices being a grid, the chances of an ion strike affecting adjacent transistors is highly likely. Whenever multiple bits flip due to SEU it is called a Multiple-Bit Upset (MBU). Also, if the affected CMOS transistor is a part of a memory device that is a sub component of a finite state machine, the error is called a Single-Event Functional Interrupt (SEFI).

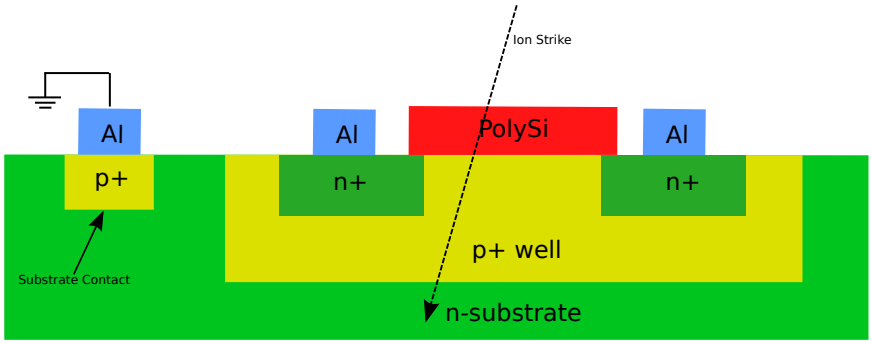


Figure 2.11: Corruption of a N-MOS Transistor by SEU

2.4.3 Single-Event Transient. A SET occurs when a charge collected from ionization discharges in the path of the normal circuit output [32]. This spurious signal is indistinguishable from a normal output and is much harder to detect. The SET radiation effect also depends on the technology being targeted [3]. As the technology size reduces, so does the supply voltage. The effects of SET has been shown to depend on the supply voltage with a much higher transient pulse width in lower voltage technologies.

2.4.4 Single-Event Latchup. The most harmful of the SEE radiation effects is the SEL type effect. This effect can cause a circuit to behave incorrectly until the device is power cycled or could create a high enough current that the device is shorted and will not function properly again. The SEL type effect is caused by the parasitic NPN and PNP Bi-Polar Junction Transistors (BJT) inherent in the CMOS process design [29]. Figure 2.12 shows the corresponding NPN and PNP BJT transistors created in a simple digital CMOS inverter circuit with the PNP and NPN parasitic transistors shown as the dashed rectangles. Figure 2.13 shows the corresponding circuit of the BJT parasitic transistors in the inverter circuit. Depending on which region of the CMOS device is hit by the radiation strike, one or both of the parasitic transistors can be turned on, it can either cause an erroneous output or create a high current scenario that can destroy the device. The erroneous error would be in effect

until the device is power cycled and is usually referred to as a *hard* error if the affected transistor is part of a memory device.

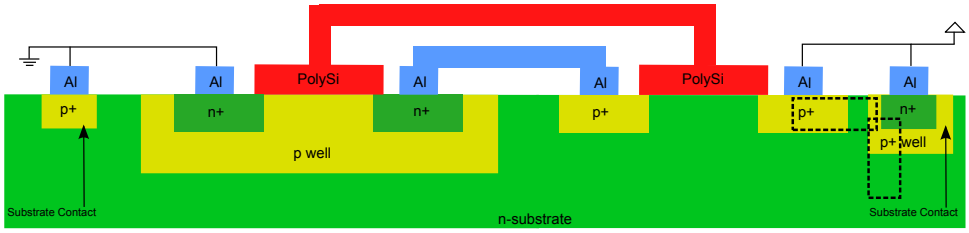


Figure 2.12: Parasitic BJTs in MOSFET

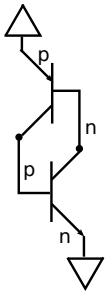


Figure 2.13: Parasitic Connections in CMOS Transistor

2.4.5 Increasing Radiation Hardness in Digital Circuits. Now that the causes of radiation effects on a digital circuit are known, the methods to counteract these effects must be looked at. Several methods also exist to restrict the number of radiation hits to the IC as well as methods to correct for any sustained radiation hits. The first set of methods target the digital CMOS standard cells. Within this set of methods are two main categories to decrease the number of radiation hits at the gate level and comes from the two ways to harden a device. These categories are the by process and by design. The by process method seeks to increase the radiation hardness through specialized CMOS processes while the by design method increases the radiation hardness by strengthening each cell in the CMOS process library. The hardened by design category also adds redundancy at the gate level into the most susceptible portion of the IC or any other strategic placement where erroneous data

cannot be allowed to propagate forward through the circuit. Both by design methods for radiation hardness will be looked at further.

2.4.5.1 Transistor Level Radiation Hardening. At the heart of hardened by design is structurally hardened memory devices and voting circuits. A structurally hardened memory device may employ multiple hardening techniques such as Reverse Body Bias (RBB) of the N-MOS transistors [23, 24] as well as guard ring material around each set of P- and N-MOS transistors [4]. Among the other techniques is to utilize a Silicon-On-Insulator (SOI) process or a BJT process. It has been shown that the SOI process increases the radiation hardness of an IC due to the n- and p-wells being completely separated [7]. It has also been shown that BJTs have a much higher tolerance to radiation effects [6]. While both the SOI and BJT processes have merit, this research targets the standard CMOS process, thus only the RBB and guard ring techniques will be considered. Both of these techniques create memory cells, namely Static RAM (SRAM) that will withstand an SEU.

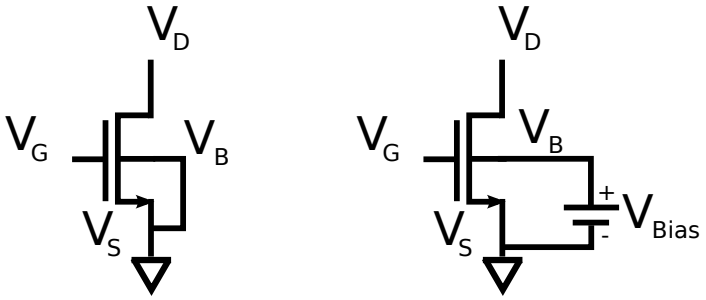
For the RBB technique, the principle idea is to increase the transistor threshold voltage such that it can withstand a certain amount of TID radiation before the radiation causes a threshold voltage shift [18]. The driving principle for this shift in transistor threshold is due to the equation for the transistor threshold voltage given in Equation 2.24. The values for γ, V_{TH_0} and $2\phi_F$ are all process dependent variables.

$$V_{TH} = V_{TH_0} + \gamma(\sqrt{V_{SB} + 2\phi_F} - \sqrt{2\phi_F}) \tag{2.24}$$

As an example, the Taiwan Semiconductor Manufacturing Company (TSMC) .18 μm CMOS process, taught in many universities throughout the United States, has the following parameters for an N-Type MOSFET:

$$\begin{aligned} \gamma &= 0.52\ (V^{0.5}) \\ V_{TH_0} &= 0.356\ (V) \\ 2\phi_F &= 0.7 \end{aligned}$$

With the schematic in Figure 2.14(a), the threshold voltage of the N-type MOSFET is determined by the process dependent V_{TH_0}, while the schematic in Figure 2.14(b) has a non-zero V_{SB} and thus has a modified threshold voltage. Using the parameters for the TSMC .18 μm CMOS process and the schematic shown in Figure 2.14(b), Figure 2.15 shows a graphical representation of the V_{Bias} effect on the threshold voltage V_{TH}. Table 2.1 summarizes the RBB effect on the threshold voltage on a limited number of V_{Bias} voltages versus the threshold voltage V_{TH} from Figure 2.15.



(a) Typical Configuration

(b) RBB Configuration

Figure 2.14: MOSFET Configurations

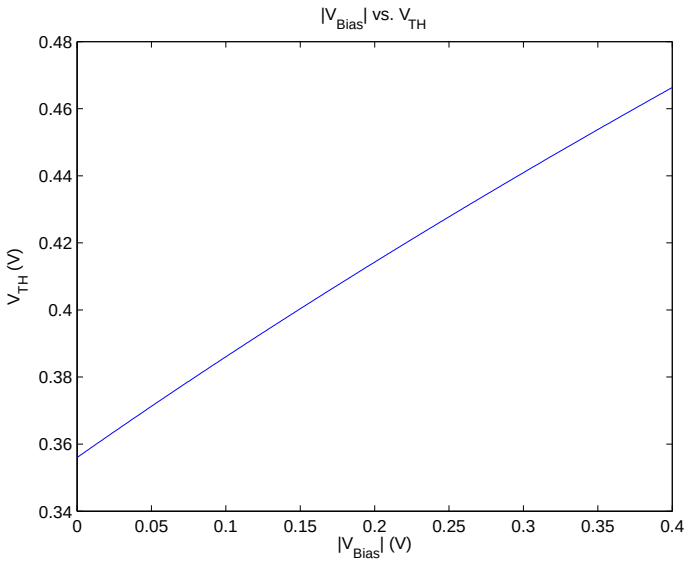
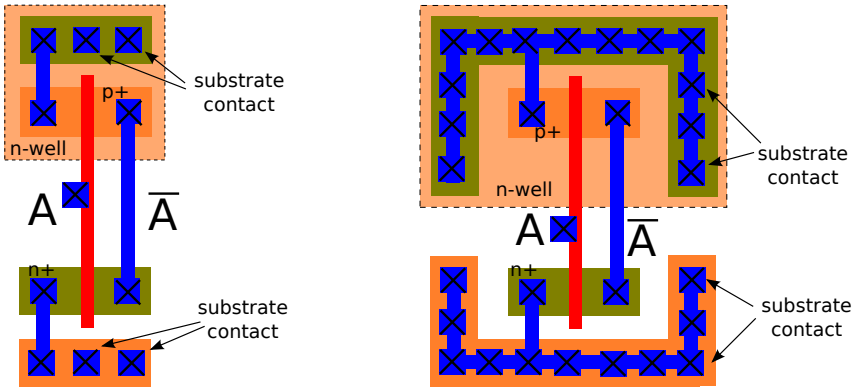


Figure 2.15: Reverse Body Bias Voltage Versus Transistor Threshold Voltage

Table 2.1: RBB Effects on Transistor Threshold Voltage

V_{Bias} (V)	V_{TH} (V)
0.0	0.356
-0.1	0.386
-0.2	0.4143
-0.3	0.4409
-0.4	0.4663

Lastly, for the guard ring technique, the driving principle is to increase the area of the substrate contacts to allow for more separation of adjacent digital cells and also to increase the current capability of the digital cell such that a large energy strike can be safely discharged without any damage to the transistors in the cell. Figure 2.16 shows a 2D drawing of a CMOS inverter with and without the added guard ring.



(a) Without Guard Ring

(b) With Guard Ring

Figure 2.16: CMOS Inverter on P Substrate

2.4.5.2 Gate Level Radiation Hardening. There exist various methods to prevent a possible erroneous output from one transistor to propagate further through an IC. The key methods for this involve spatial redundancy, error checking and correction (ECC) and adding temporal redundancy [22]. Each of these methods will be discussed in further detail.

For spatial redundancy, voting circuits are used to determine the appropriate output. Many voting circuit techniques exist, but the most common is the simple

majority voting circuit, with the simplest two-thirds majority gate shown in Figure 2.17. This circuit provides spatial redundancy in the device datapath that will counteract an SEE. This is due to two of the three redundant cells required to output erroneous data for the erroneous value to propagate. To increase the hardness of the device, larger voting circuits such as the three-fifths and four-sevenths may be used for further redundancy.

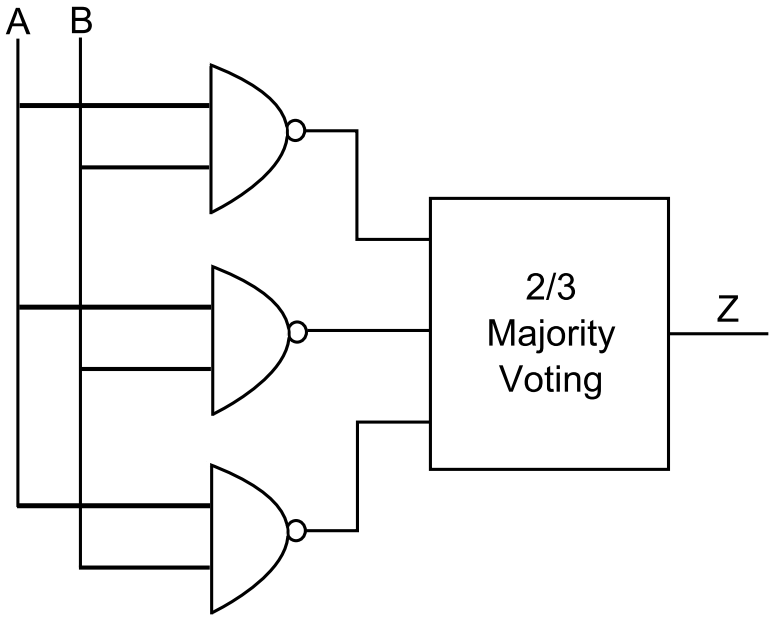


Figure 2.17: Two-Thirds Voting Circuit using NAND2 gate

For the ECC method of hardening, there exists several methods for detecting an error and similarly for correcting an error. The primary method for catching errors in combinational circuits is the Cyclic Redundancy Check (CRC) category of error checking. The CRC algorithm involves a specific polynomial chosen based on the number of bits being presented serially or in parallel and the number of bit errors that the CRC should detect [19]. The key behind the CRC is in the hamming distance [12] of the data itself. The larger the hamming distance, the more bit errors that can be detected, while having a large hamming distance increases the algorithm complexity.

Similarly, there are several methods for error correction algorithms, with the most famous being the Hamming encoder [25], and others being BCH, Reed-Solomon

and Reed-Muller encoding schemes. The key to each of the encoding schemes is to embed the error correction bits into the data. This increases the total number of bits needed for storage as well as interconnections between modules. The key to any error correction circuit is the hamming distance. Just like the CRC error checking, the larger the hamming distance the more complex the algorithm and also the more bits needed for the error correction. The simplest Hamming ECC encoding scheme is the Hamming(7,4), which receives four data bits and encodes three bits for the error detection and correction. This scheme is capable of correcting a single bit error and detecting double bit errors.

The temporal latch is a patented design, that implements temporal redundancy. This technique adds time delays between a digital gate and a memory element that allows for the correction of SET errors caused by radiation strikes. The time delays added between the gate and the memory element can be adjusted for different clock rates and radiation SET target pulse width correction. Figure 2.18 shows the basic idea behind this technique.

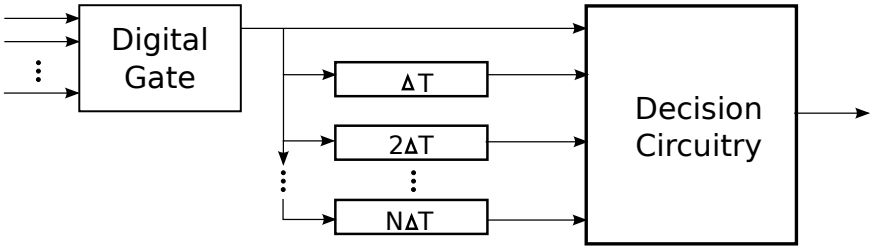


Figure 2.18: Basic Temporal Latch Idea

2.5 Structured ASIC

Traditionally, ASIC's are used when power, area and speed are an issue and FPGA's are used during prototyping and in situations when there will be low volumes and there is a need to save cost. ASIC's provide the best power, area and speed performance over FPGAs and are limited mainly by the cost of fabrication. FPGA's, on the other hand, are drastically cheaper to purchase and come with built-in features

for memory and IO communications. An S-ASIC falls somewhere between FPGA's and ASIC's. S-ASICs fall closer to an ASIC in performance and closer to an FPGA in the design flow and overall structure. The key features of an S-ASIC is that the layer of transistors and bottom metal layers have been designed and fabricated with only the top metal layers permitting any changes. The basic structure utilizes standard digital gates such as AND and OR with a D-type Flip Flop (DFF). These basic building blocks allow for larger, more complex devices to be created, but at the cost of more area and power. An FPGA on the other hand may include specialized IP multipliers, adders and other macro structures for various purposes that can be synthesized during the design flow that may not be present on the S-ASIC.

As can be seen in Figure 2.19, the basic structure of an S-ASIC is broken up into rows and columns of logic and memory elements distributed across the die area. The figure also shows the basic building block of an S-ASIC. The building blocks have the bottom layers of metal fully routed and allow the top metal layers to be added so that the unit is connected. This is similar to the way the routing of an FPGA is accomplished, although the FPGA uses memory elements to assert or un-assert the connection to the basic units that can be reconfigured by software. The S-ASIC does not permit any changes after fabrication. Figure 2.20 shows a comparison of the difference between the FPGA routing and the S-ASIC routing.

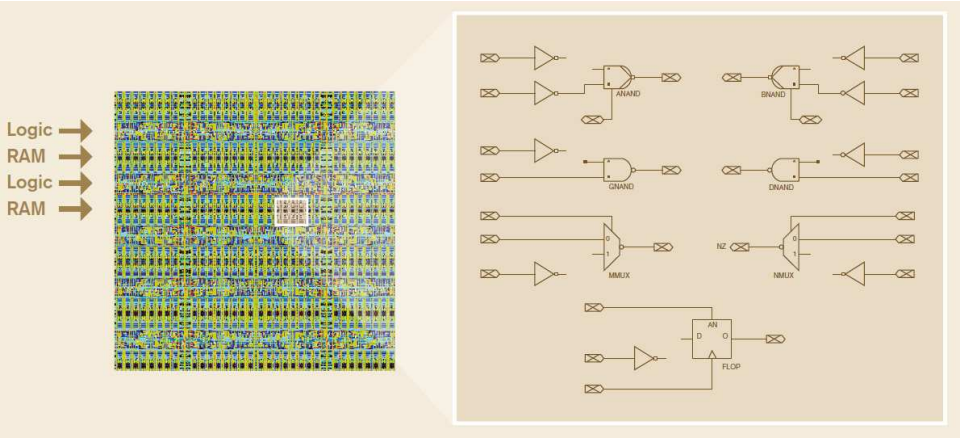
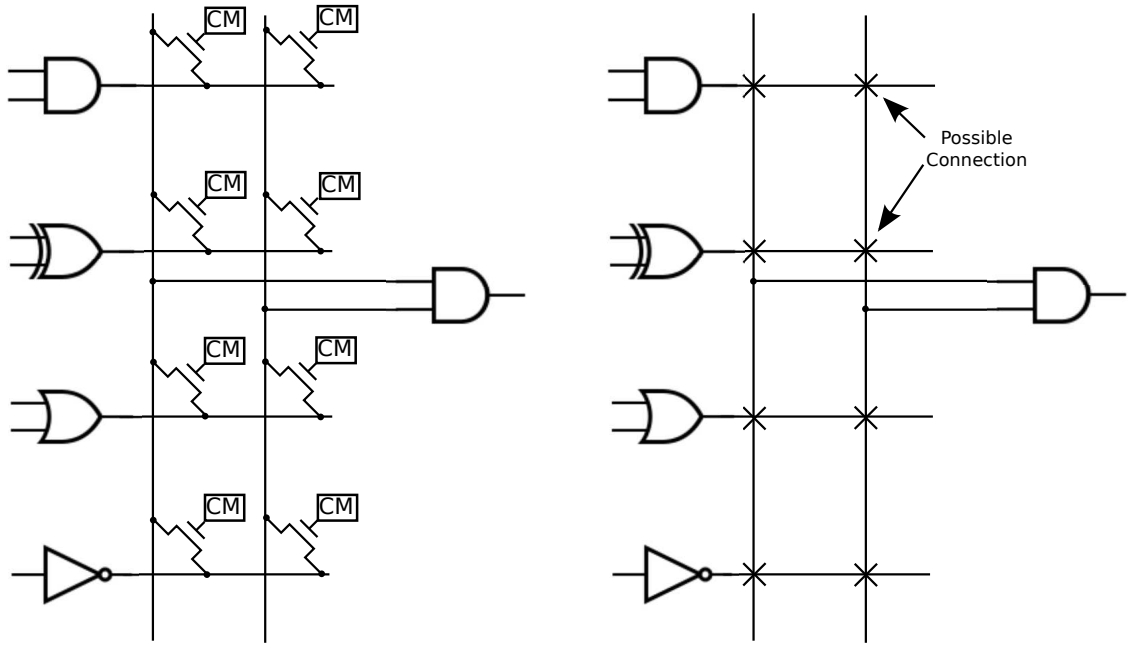


Figure 2.19: Distributed Logic and Memory Structure with Basic Logic Cell [31]



(a) FPGA with Configuration Memories(CM) (b) S-ASIC with Possible Via Connections

Figure 2.20: Interconnections Between Gates

2.6 Digital Radio Frequency Memory Designs

Some of the key features of a DRFM are the abilities to capture part of a RF spectrum of interest, store digitized samples and then re-transmit an identical copy of the original spectrum or create a frequency shifted version of the spectrum. Each of the modules needed to create a DRFM are not unique and can be replaced with different components depending on the requirements of the system. The ADC, storage and DAC are needed by the DRFM before any work can be accomplished. As shown in Figure 2.21, the DSSM is only a small portion of the entire system, but incorporates the digital processing portion of the system with the mathematical calculations and control circuitry [28].

Several methods exist in which to create a DRFM from a digital channelized approach [11, 34] to creating a DRFM on a Gallium Arsenide (GaAs) Monolithic Microwave IC (MMIC) [20]. While each of these different DRFM systems have their merits, the target of this research is the DSSM. This DSSM neglects any signal storage

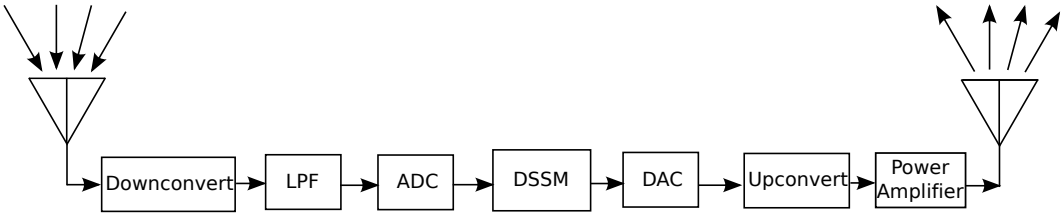


Figure 2.21: DRFM System Including Antennas, ADCs and DACs

and retrieval and is targeted for the S-ASIC using a radiation hardened by design CMOS process.

III. Methodology

The DSSM presented in this research represents only a small portion of the overall DRFM system shown previously in Figure 2.21. This research explores different hardware utilization scenarios on multiple platforms to demonstrate successful implementation of the DSSM on an S-ASIC. This research also explores the benefits of radiation hardening by design and presents a tradeoff from a standard non-hardened DSSM designed for the IBM 9SLP 90nm CMOS process as a standard ASIC [16]. The key goal of this research was to create semi-generic VHDL source code that could be modified quickly but still be as area efficient as possible to fit within the bounds given by the S-ASIC design being targeted.

Since several pieces of the DSSM have been left out during this research a separate system level view of the DSSM design created during this research is needed. Figure 3.1 shows the toplevel view of the DSSM designed during this research. Each of the different modules will be discussed in detail later. The DSSM takes as input an RF spectrum of interest that has been down-converted to base band and then digitized and creates the in-phase (I) and quad-phase (Q) signals using a digital Hilbert filter. It then mixes the I channel with a *cos* of a given frequency and mixes the Q channel with a *sin* of the same frequency, then adds to two products together to create a signal with a frequency spectrum that resembles Figure 2.5 on Page 9.

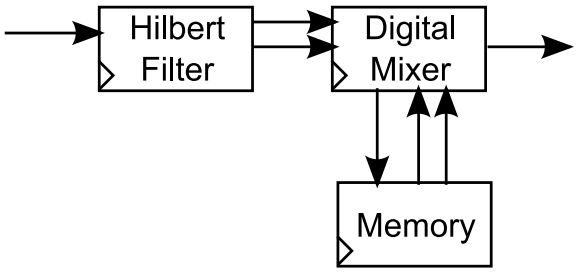


Figure 3.1: DSSM Created During this Research

3.1 I/Q Creation using Digital Hilbert Filter

The successful operation of the DSSM in this research relies on the digital Hilbert filter which creates the in-phase (I) and quad-phase (Q) signals from the incoming digital signal. The digital Hilbert filter discussed in Section 2.3.1 was designed as shown in Figure 3.2 using a previously developed Hilbert filter template [1]. Since all of the odd numbered Hilbert coefficients are zero, a more optimized Hilbert was created as shown in Figure 3.3. This design was accomplished by use of the DFF memory element for storing 1-bit every clock cycle.

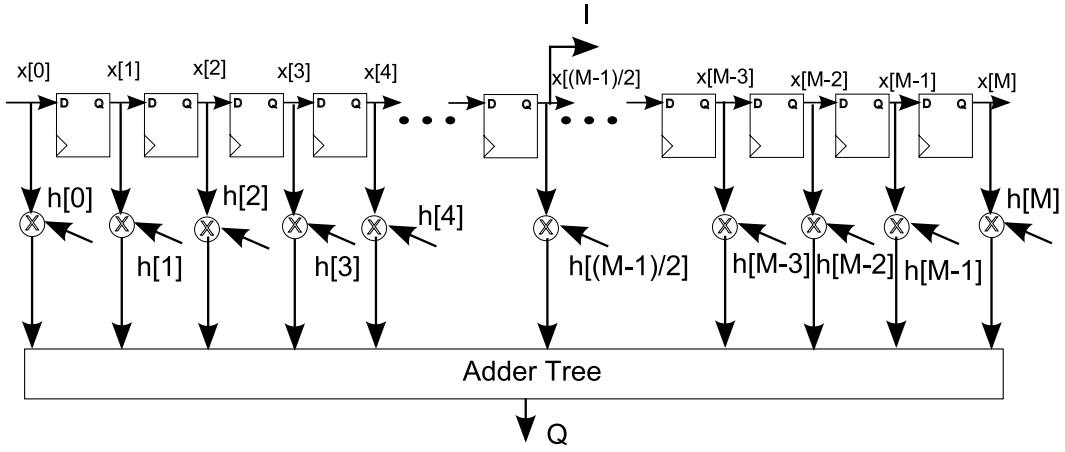


Figure 3.2: Digital Hilbert Filter Design

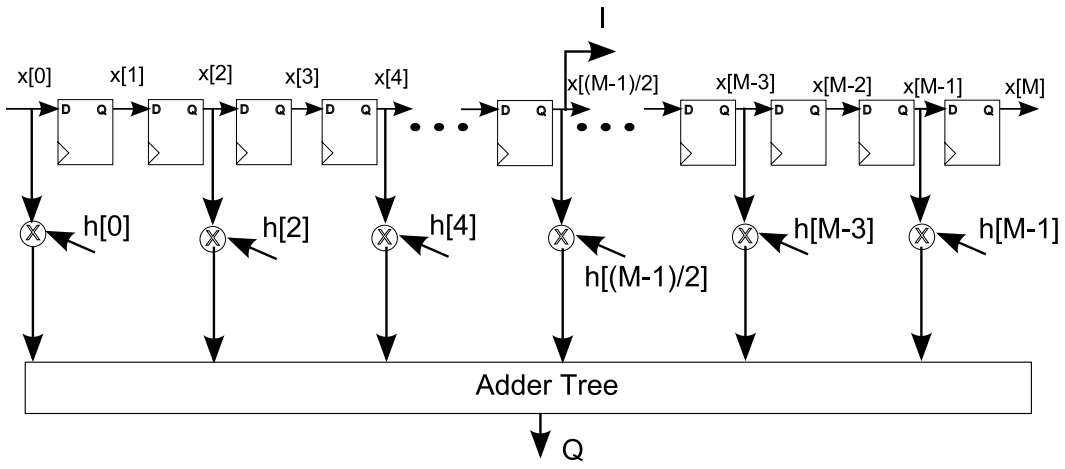


Figure 3.3: Optimized Hilbert Filter Design

3.2 Design Flow

The flow of the DSSM design in this research was targeted at creating a semi-generic VHDL design file that can be synthesized to an S-ASIC, but many other steps needed to be taken first. As shown in Figure 3.4, the design flow began with a model of the algorithm, then moved into the VHDL design and simulated using both the Cadence SimVision simulation suite and an FPGA. After successful simulations, the design could be synthesized, placed, routed and then re-simulated to verify correctness before fabricating the design. Upon receiving the fabricated design, further tests must then be completed to test its functionality.

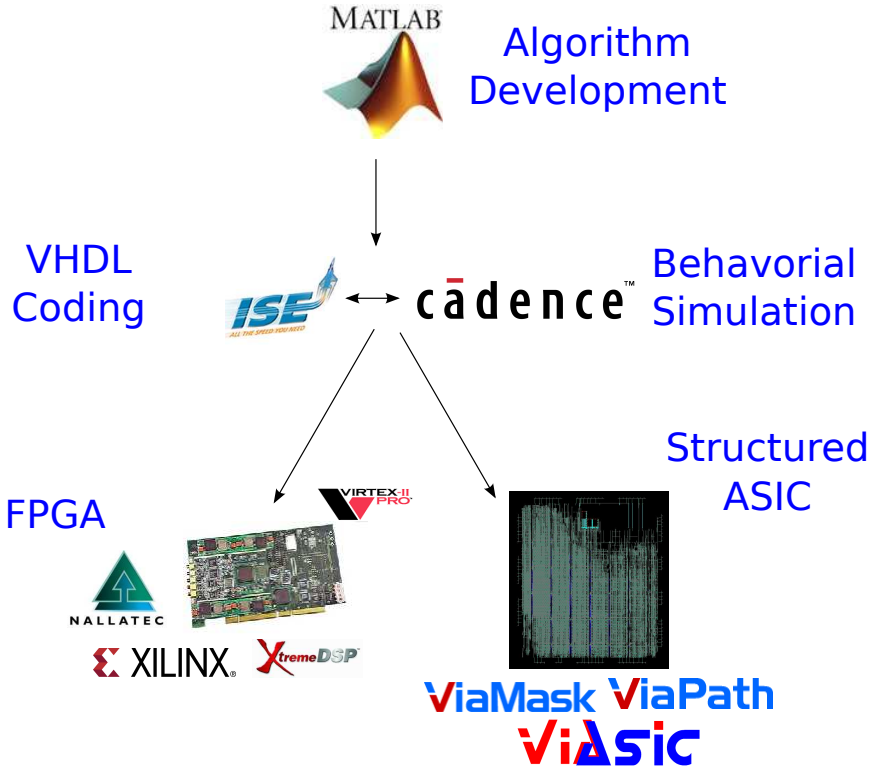


Figure 3.4: Design Flow

3.3 Matlab® Simulations

Two Matlab® simulation files were created to demonstrate the operation of the DSSM, one floating-point and the other a fixed-point representation of the designed

DSSM in hardware. Each of the two scripts can be found in Appendix A on page 80. The floating-point simulation uses the conceptualized Hilbert transform for generating the I and Q data then does the mixing with idealized sine and cosine signals to produce the final DSSM output. This version of the simulation helps to test the operation of the DSSM under ideal test conditions to verify correct operation. The fixed-point version of the simulation takes into account the various parameters of the DSSM such as the ADC and DAC bit widths, ROM size and word bit widths, and the Hilbert filter length and coefficient bit width. Each of these parameters are defined at the beginning of the script for easy modification. The addition of the fixed point parameters allows for different test scenarios based on the different design options available. An increase in the bit widths of either the Hilbert coefficients or the ADC will increase the number of gates needed in a digital circuit. Testing the increase in bit width in a simulation allows for careful calculation of the SFDR of the output as well as other design goals. A top-level simulation file was also created to allow for the testing between the floating- and fixed-point versions and displaying the output spectrum of the DSSM output, which can also be found in Appendix A.

3.4 VHDL

Among the choices of Hardware Description Languages (HDL) for designing the DSSM from this research were Verilog and VHDL with different styles in each. The choice was made to go with VHDL since the author is most familiar with VHDL. Along with VHDL being chosen as the HDL of choice, the Xilinx software was chosen as the editor of choice. The VHDL source code can be created using a typical word processor or text editor, but the Xilinx software allowed for syntax highlighting as well as the option to synthesize to the specific FPGA demo board used for prototyping. The use of the Xilinx software also allowed for field testing of changes on the FPGA demo board without further modifications to the code. A figure showing the Xilinx environment with the DSSM VHDL code is shown in Figure 3.5.

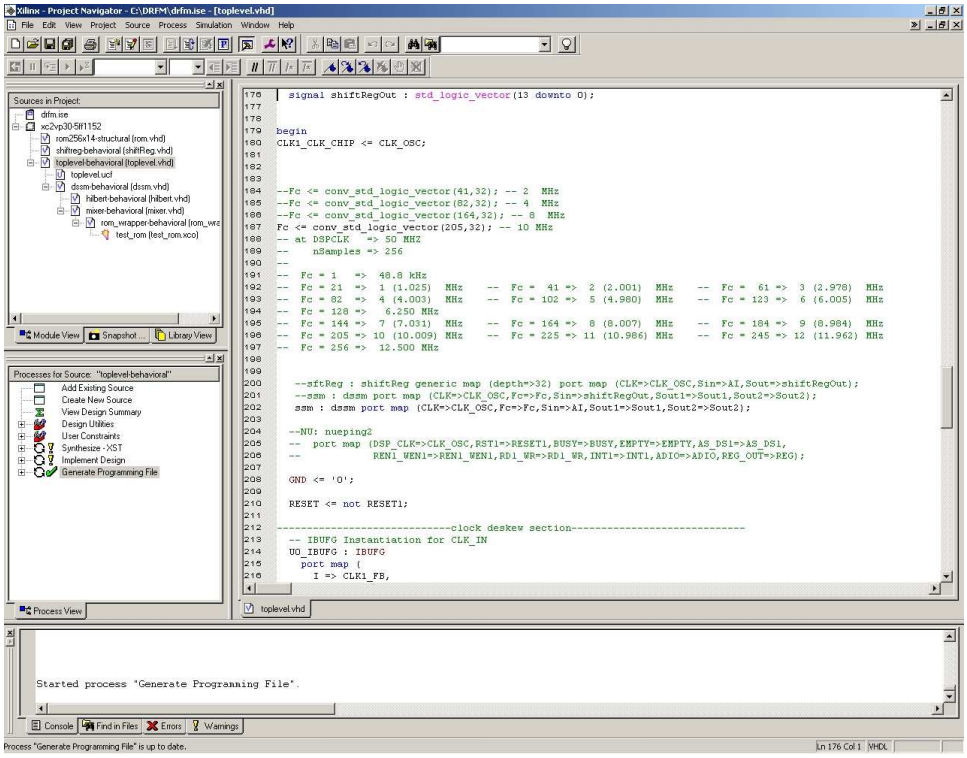


Figure 3.5: Xilinx Software Used In VHDL Development

3.5 Digital Hilbert Filter

For the digital Hilbert filter, the construction was simply a behavioral model of the structure previously shown in Figure 3.3. The Hilbert filter was created as a variable length and variable width shift register that captures data from the input on every rising edge of the clock input. Every two outputs of the shift register were then used in constant multiplication with the Hilbert coeffecients and then summed to generate the Q output signal. Along with the summation of the multiplication operations for the Q output, the I output signal was generated directly from the shift register. Separate VHDL process blocks were written to create the processes described. These processes create the I and Q signals from the input signal, as a digital Hilbert filter should. An addition the author made was to include a third output, a shift register output. This output is simply the last input sample stored in the variable length shift register which is fed out of the filter entity for use as a feedback signal to simulate a DRFM memory structure. This addition added only a

few additional combinational resources while allowing several additional operations to be implemented in the DRFM.

3.6 Digital Mixer

The digital mixer is not as complex as the Hilbert filter, but involves several steps nonetheless. The digital mixing involved multiplying the I signal by a *cos* of a given frequency and the Q signal by a *sin* of the same frequency, then sum the two results together. Simple as it may be, several considerations were taken in order for the digital mixer to behave correctly and the output to be the correct DSSM operation intended. Figure 3.6 shows the basic block diagram of the digital mixer. After several simulations, it was determined that depending on the relationship between the incoming signal frequency and the shifting frequency, the output result could either be an addition or subtraction operation. If certain conditions arise that either the incoming signal will change or the shifting frequency changes and a certain operation is desired, separate control circuitry must be implemented in order for this feature to be available. Figure 3.7 shows the additional circuitry needed to implement the new features.

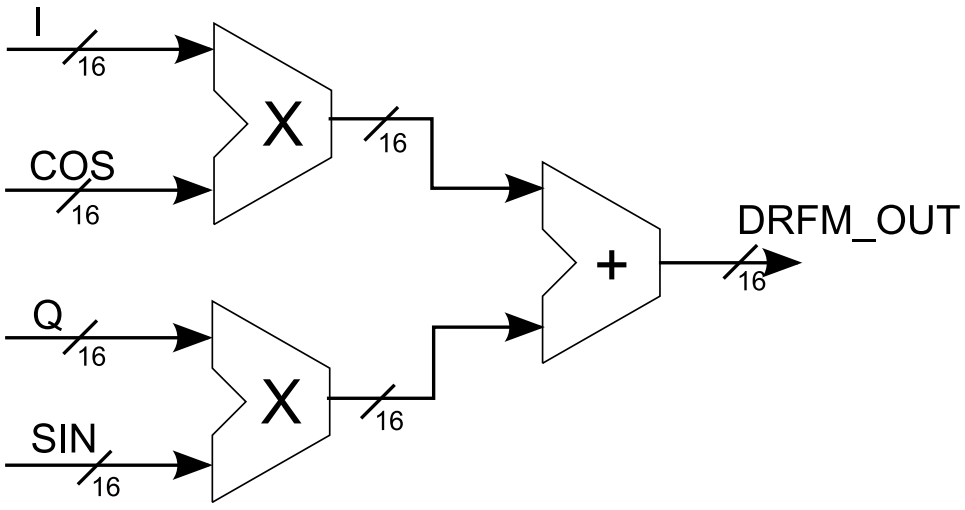


Figure 3.6: Digital Mixer

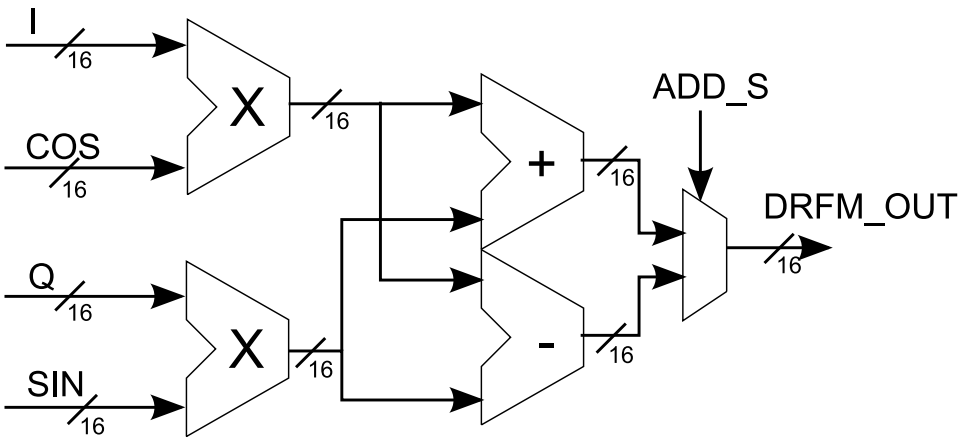


Figure 3.7: Digital Mixer With Additional Control Circuitry

3.6.1 Wallace-Tree Multiplier. During the development phase of the DSSM VHDL design, the opportunity became available to remove control from the Cadence RTL Compiler for the structure of the signed multipliers used in the digital mixer. This opportunity arose from the fact that the number of DFFs available on the chip were limited, but the number of logic cells utilized thus far was approximately 50%. Since the design would work with whichever signed multiplier was utilized in the digital mixer and the power and timing of the overall DSSM design was not limited, the author chose to utilize a Wallace-Tree Multiplier (WTM) for the signed multiplier. This multiplier was created using a VHDL generator program that the author had created previously for a separate project. This generator takes into account the input size of each of the two signed input bit-widths and whether or not each stage of the adder tree produced would include a pipeline register. For this design, the author chose to utilize a pipelined 16×16-bit signed WTM. The number of stages needed in this multiplier was limited, so the number of DFFs needed for the pipeline registers added to the number already used by the Hilbert filter did not occupy every DFF available in the chip. The Carry Save Adder (CSA) is the key component used in the WTM as it creates a 3:2 compressor tree down to the bottom level. The bottom level, which has two 32-bit operands, uses a Carry Look-Ahead Adder (CLA) to perform the last, large addition with minimal gate delays. The advantage of using the WTM is that it utilizes a CSA as its building block, which can be optimized for the given

architecture and a low propagation delay CLA for the last stage. This WTM also allows for fast clock frequencies due to the pipeline stages.

3.6.2 Carry Look-Ahead Adder. Since the author chose to utilize more of the logic cells in the chip for the WTM, the author also decided to utilize a CLA for the final 32-bit addition in the 16 × 16-bit WTM, as well as the 16-bit addition and subtraction at the end of the digital mixer. The CLA was created using a separate VHDL generator created by the author for a separate project. This generator takes as arguments the bit width of the two inputs. The bit widths for the two inputs are assumed to be equal. This CLA utilizes a tree structure [30] that constrains the largest digital CMOS gate to an input size of four and constrains the maximum delay of each level to be at most three gate delays. This structure not only produces a fast adder but also constrains the size of the largest CMOS gate. In terms of Big-Oh (O) notation, the CLA utilized in this design, with bit width of N, has a maximum delay equal to O($log_4(N)$) gate delays with the largest single gate delay equal to the delay of a 4-input AND gate.

The advantage of the decision to utilize the tree structured CLA in the WTM, as well as the adder and subtractor in the digital mixer, is the low propagation delays and more efficient utilization of the overall chip area. Due to the chip structure, the overall chip had minimal DFFs which limited the size of the digital Hilbert filter and pipeline stages. The CLA, with its completely combinational design, allowed for more combinational cells in the chip to be utilized without using any DFFs and also without any performance hit to the overall chip. The other possible adder choice was a Ripple Carry Adder (RCA) where the carry bit from each addition is propagated through each stage. The total delay of the RCA with bit width N is O(N). The RCA utilizes the fewest amount of resources but sacrifices speed. Another possible choice is the full CLA that looks ahead N-1 bits. The delay for this CLA is O(1) since the lookahead is done in parallel and each bit is summed in parallel. The actual delay of the circuit would be greater than one due to the very large gates needed for the lookahead logic.

The full CLA also consumes the largest number of resources for the addition. This tree-based CLA design lies close to the size of the RCA for low resource usage while being closer to the full CLA for total propagation delay.

3.7 Shifting Frequency Generation

For this DSSM the shifting frequency generation is controlled by an input signal and quarter-wave ROM-based DDS. The ROM stores samples that represent the quarter-wave samples of a *cos* or *sin* waveform. The full period of the *cos* or *sin* waveform is governed by the internal clock frequency. The input signal, with bit width NF, selects the appropriate number of samples in the ROM to skip in order to increase the shifting frequency. Figure 3.8 shows the *cos* waveform that is generated if the ROM has its values read to an output in sequential order. Placing a wrapper around the ROM in order to modify the address before collecting the sample data, as shown in Figure 3.9, can generate a full length *cos* waveform as shown in Figure 3.10. By modifying the scaling factor NF above one, the user can selectably generate different frequencies that will then be used in the DSSM shifting operation. Figure 3.11 shows four separate scaling factors and their associated full length *cos* waveforms generated by using the DDS.

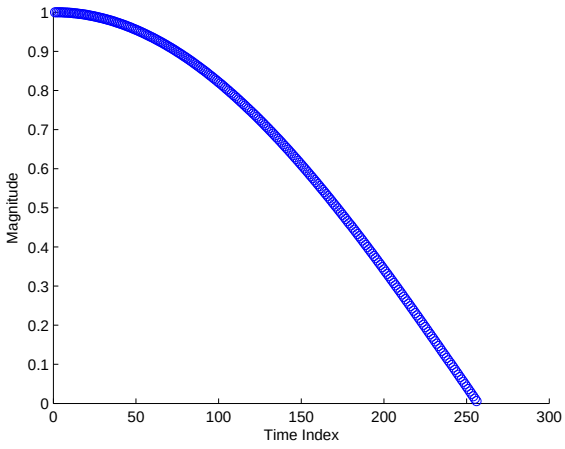


Figure 3.8: ROM Storing One−Fourth of the Cosine Waveform with 256 Samples

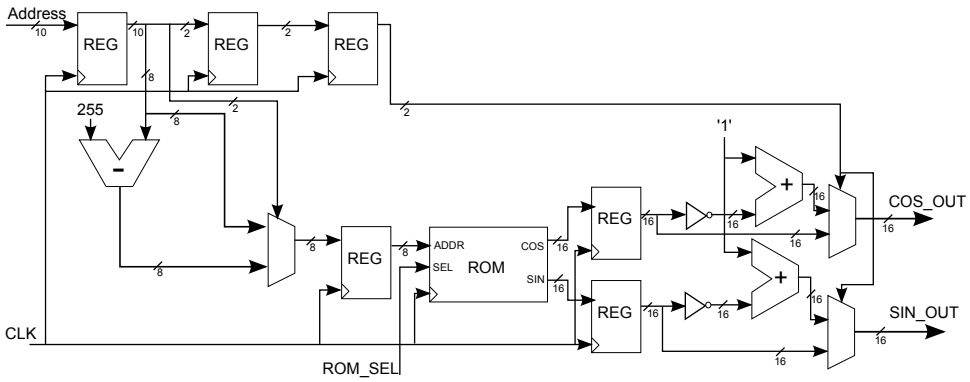


Figure 3.9: ROM-Based DDS Structure

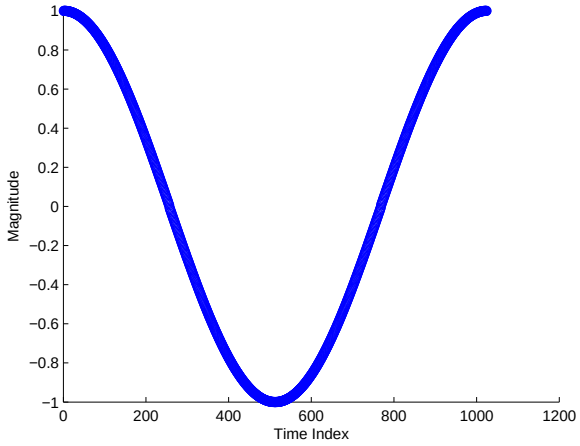


Figure 3.10: Full Length Cosine Waveform Generated by DDS

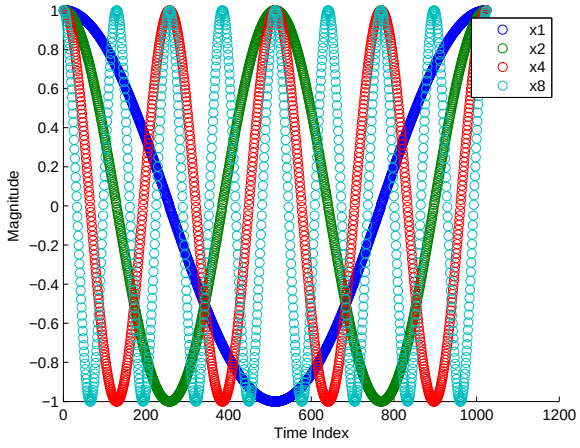


Figure 3.11: Four Separate Cosine Waveforms Using Four Different Scaling Values

By the use of the quarter-wave ROM-based DDS, different scenarios, such as using separate ROMs for the *cos* and *sin* waveforms or using a single Dual-Port ROM for both, can be used. There may also be multiple types of ROMs, such as block SRAM, distributed SRAM or a full synthesized gate implementation of a ROM. The DSSM operation is not affected by the selection of the ROM used inside of the DDS as long as each ROM behaves similar in nature to each other and the timing of each is consistent.

In this research, the DDS designed uses an IP block SRAM from a commercial company. This block SRAM has a simple ROM operation in which the ROM samples are pre-loaded into the SRAM and writing to the ROM is forbidden. Along with the IP ROM, a separate gate implementation of a ROM was created that behaves like a standard ROM, except the data samples are created using physical gates based on the address provided. This implementation was added to allow for switching between the provided IP ROM and a gate implemented ROM with known functionality in order to test the functionality of the IP ROM. The gate implementation ROM was designed by the author in VHDL and was synthesized to the S-ASIC design library and can be clocked at the same frequency as the IP ROM.

3.8 VHDL Implementation

The main focus of the VHDL design was to create a semi-generic design in which a user can modify certain parameters easily. The parameters that are modifiable by a user is given in Table 3.1 and gives a description for each parameter. While the NF parameter can be any integer, it must follow one simple rule based on the size of the ROMs shown in Equation 3.1.

$$NF < \lfloor \frac{log_2(\#\ of\ samples\ in\ ROM)}{2} \rfloor \tag{3.1}$$

Table 3.1: Generic User Modifiable Parameters

Parameter	Description
NF	integer number of bits of the shifting frequency (more bits = higher resolution)
NC	integer number of bits of the cosine/sine waveforms
NCO	integer number of bits of the cosine output (must be <= NC)
NI	integer number of bits of the input ADC signal

Other parameters that are not user changeable are the number of taps in the digital Hilbert filter, the bit width of the digital Hilbert filter coefficients, the ROM addressability and the bit width of the ROM samples. If either of the two parameters of the digital Hilbert filter need modified, a new VHDL file for the filter must be generated using those parameters and that entity would then be used with no other modifications to the design. If either of the two parameters for the ROM are modified, several parameters must be checked for possible modification. If the addressability of the ROM is modified, the NF generic parameter must be modified. If the bit width of the ROM samples is modified, the NC and NCO generic parameters must be checked. It is possible for a generating application to be created in which the given parameters for both the Hilbert filter and the ROM are provided by the user and VHDL entities are generated for shorter modification times and easy implementation. Figure 3.12 shows the dependency tree for the VHDL design files needed to simulate and synthesize the DSSM design.

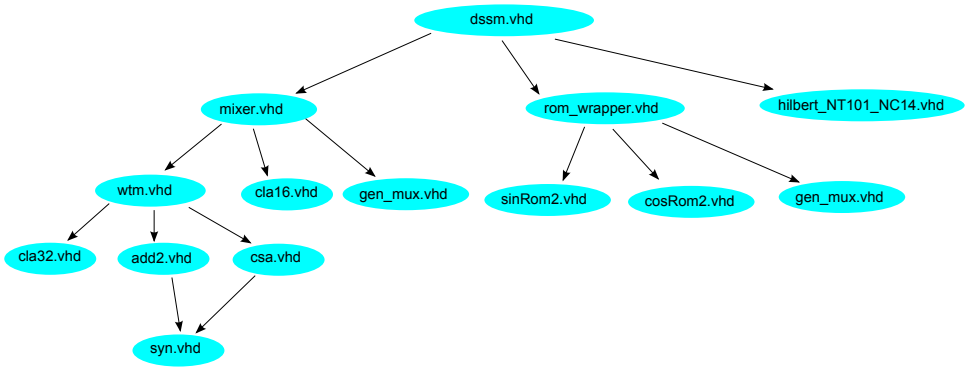


Figure 3.12: VHDL File Dependency

The top-level VHDL entity stitches together the DSSM to form the design shown previously in Figure 3.1 on page 27. The inputs and outputs are easily viewable from this entity, as are the interconnections between modules. This design was chosen to allow for quick modifications to the DSSM design without the need for multiple files to be modified. This structure also allows for different Hilbert filters, mixers and DDS topologies to be tested without a need to modify the top-level entity.

3.8.1 VHDL Simulation. For simulating the VHDL DSSM design, the Cadence SimVision simulation suite was used to generate test patterns and write the associated outputs to a file. The simulation software allows for separate VHDL and Verilog design files and either a VHDL or Verilog testbench that does the interaction with the DSSM. The designed testbench was modeled in VHDL and creates a floating-point representation of an incoming *sin* waveform, then converts it into a fixed-point representation based on the number of input bits an ADC would provide. After providing the clock and input data, the output data is written to separate files for each stage of the DSSM. These separate files are then loaded into Matlab® and their functionality is then verified by the script in Appendix A. Both the DSSM VHDL design and the synthesized Verilog DSSM design were simulated using the same simulation testbench that provides identical inputs and control circuitry to test functionality easily. The VHDL testbench used for the VHDL DSSM design is shown in Appendix B with the synthesized DSSM design testbench being a variant of the same VHDL testbench file without any generics.

3.9 FPGA Implementation

For the FPGA DSSM implementation, several considerations were taken into account before a design was created. The targeted FPGA was a Xilinx FPGA demo board inside of a desktop PC. The demo board features two 14-bit ADCs and DACs as well as an external clock input. This demo board was used on the Peripheral Component Interconnect (PCI) expansion card slot on the desktop PC as shown in

Figure 3.13. Since there were two DAC outputs available, it was decided that the Q output of the Hilbert filter would be sent to one of the DACs and the overall DSSM output would be sent to the other. This helped in determining the performance of the Hilbert filter as well as the overall DSSM operation.

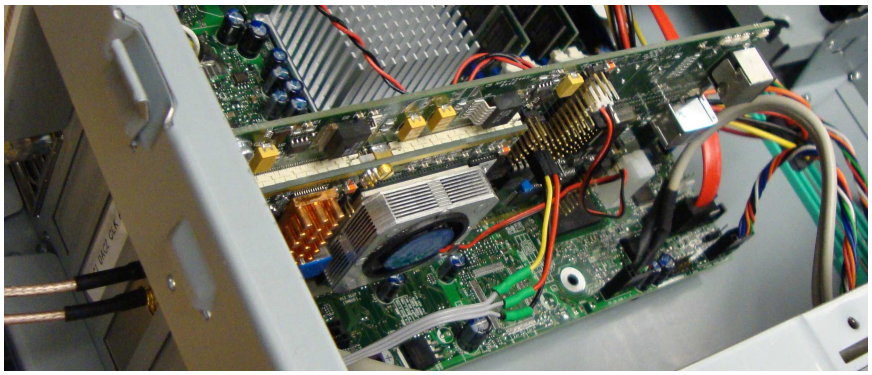


Figure 3.13: Xilinx FPGA Used In Testing

3.10 Structured ASIC Implementation

Utilizing a ViAsic ViaPath PAR tool along with a IBM 9LP 90nm CMOS radiation hardened by design standard cell library, on which the S-ASIC was targeted, the DSSM design was created. This design utilized the VHDL DSSM entities simulated for functionality in the Cadence SimVision simulation suite as well as on the FPGA, but with a few changes. The FPGA implementation utilized a single IP ROM from Xilinx that stored the *cos* and *sin* samples, while the S-ASIC implementation utilized a block IP SRAM as well as a gate implementation of a ROM. These changes, although small, required several modifications as well as several repeated simulations in order for the simulations to match. Table 3.2 shows the parameters chosen for the S-ASIC DSSM.

For the DSSM S-ASIC design the smallest size available was a 3mm $\times$ 3mm chip area, thus the approach was to utilize as much of the chip area possible, but still achieve the fastest clock frequency as possible. For utilizing the most amount of area on the chip, the design approach was to increase the length of the digital Hilbert

Table 3.2: VHDL Parameter Selections

Parameter	Value
IO Bits	16
ROM Samples	256
ROM Bit Width	16
Hilbert Filter Length	101
Hilbert Bit Width	14

filter beyond that of the FPGA tests. As previously shown in Figure 3.3, the length of the filter will increase the number of memory registers, multipliers and adders in the overall circuit. The limiting factor on the length of the digital Hilbert filter was the the total number of DFFs. The total number of DFFs in the 3mm $\times$ 3mm chip area were reduced by half based on the number of VCells, and for this chip area, the total number of DFFs is 4864. An optimum length for the digital Hilbert filter was found to be 101 filter taps. The optimum number of filter taps was determined by using Equation 3.2 to utilize approximately 50% of the chip's available DFFs. Also, the Hilbert filter length must satisfy Equations 3.3 and 3.4.

$$\text{Hilbert Length} = 50\% \times \left\lfloor \frac{\text{Total DFFs - (\# of DFFs in Mixer + \# of DFFs in DDS)}}{\text{\# of input bits from ADC}} \right\rfloor \tag{3.2}$$

$$(\text{Hilbert Length} - 1) \bmod 2 = 0 \tag{3.3}$$

$$\left(\frac{\text{Hilbert Length} - 1}{2} \bmod 2 \right) = 0 \tag{3.4}$$

The S-ASIC chip also had a total of 96 user IO pins. This in contrast to the Xilinx FPGA, which only had two DACs of 14-bits each, drastically limited the ability to debug pieces of the DSSM. Since the S-ASIC has many more IO pins available, it was determined that having an output after each stage of the DSSM process would deliver the best possible testing approach. This testing approach led to a separate DSSM design seen in Figure 3.14, which has an output after each stage in the DSSM process. This design was created using the VHDL design files and synthesized using the Cadence RTL Compiler using the provided target standard cell library. The

synthesis tool transformed the behavioral code in the DSSM VHDL design files into a structural Verilog file, ignoring the custom IP ROMs and inserting blackboxes for those entities. Given the Verilog file and the target library, a post-synthesis simulation was completed using a behavioral VHDL version of the custom IP ROMs created by the author. The simulation was completed in the same fashion as the original VHDL design simulation with the only difference being the removal of the generic bit widths of the input and outputs and the insertion of static bit widths for each input and output. The results of this simulation is featured in Chapter 4 along with the other simulations completed.

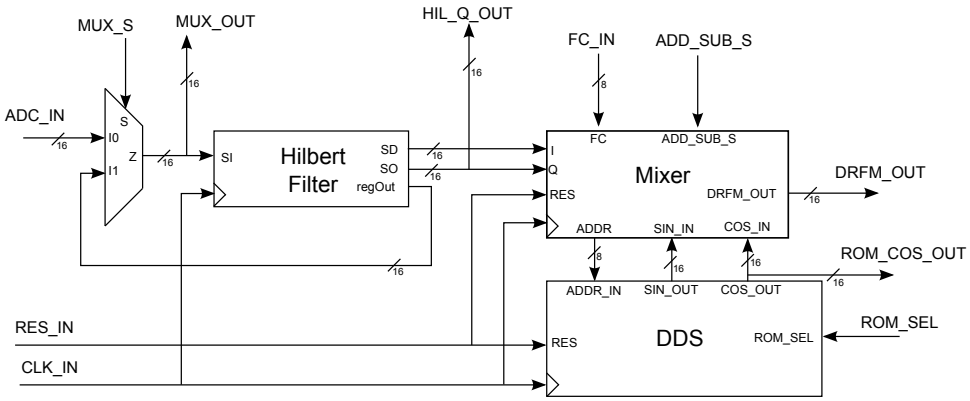


Figure 3.14: Structured ASIC DSSM Design

After the synthesized version of the DSSM had been tested for correctness, the PAR was then completed on the design using a ViAsic ViaPath PAR tool. The needed input files were the synthesized DSSM Verilog code and the custom IP ROM Verilog files generated for the targeted ViAsic ViaPath PAR tool. The program also allowed for the placement effort, target clock frequency and input and output capacitive loads to be input as parameters to the PAR program. The ViAsic ViaPath PAR tool would allocate the area described in the target chip layout file for the bottom layers and possible via locations. The ViAsic ViaPath PAR tool would then begin to place the cells into the VCell and add the connections where needed to go between metal layers. The ViAsic ViaPath PAR tool would also add the clock tree to the chip in the areas where VCells were utilized and remove clock trees where VCells were idle. Also

incorporated into the ViAsic ViaPath PAR tool is a Static Timing Analysis (STA) program that would check timing on the signal nets as well as the clocking skew introduced by the clock tree.

After the ViAsic ViaPath PAR tool had finished placing each of the cells in the design and fully routed the signal nets and clock trees the program output a post-PAR Verilog file for post-PAR simulations and a Graphic Data System II (GDSII) file which is needed for the fabrication process. This GDSII file only contained the top layers of the design and when added with the base GDSII file for the lower metal layers and transistor layer, the output would be the complete DSSM layout. The final PAR image produced from the ViAsic ViaPath PAR tool is shown in Figure 3.15 and shows only the layers that are connected by the vias that were utilized for the DSSM. The figure also shows the pin locations on the outer edge of the design for the IO pins utilized. The figure does not show the power and ground pins, nor the power and ground rails needed for the chip to operate. The low level design structure is a protected design and thus the author could not provide any additional information on the chip layout.

Since the structure of the S-ASIC was developed as a multi-project reticle and the fabrication is still in the qualification phase, the design tape-out date needed to be modified to reflect completion of each design being fabricated. This pushed the fabrication start date more than two months behind schedule, and as a result, the designed DSSM chip will not be available for testing before the completion of this thesis. Although there will not be any results from the chip itself, the post-PAR design code has been simulated using the designed filter lengths and bit widths and is presented along with other simulations in Chapter 4.

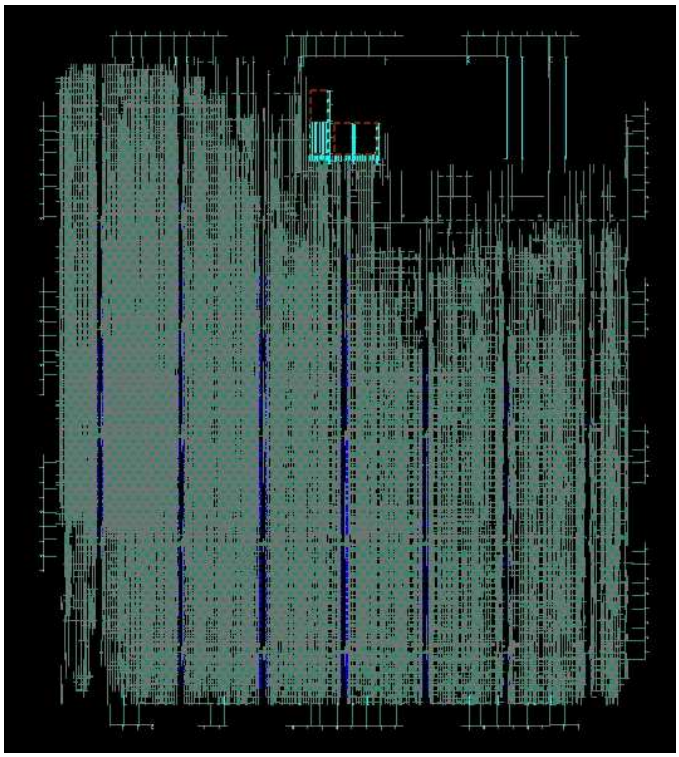


Figure 3.15: S-ASIC DSSM Placement from PAR Program

IV. Analysis and Results

The DSSM designed during this research was created through many simulations. Matlab® simulations of the DSSM system allowed for parameters to be chosen as well as provided feedback on the operation of the DSSM before the final design code was created. Through each of the simulations, the final design was created based upon different specifications revolving around the bit widths for each stage of the DSSM. Along with the Matlab® simulations, FPGA tests and VHDL simulations ranging from a behavioral simulation to a post-PAR simulation of the final design code were performed.

4.1 Choosing Parameters for Best Performance

Modifying the parameters of the DSSM designed in this research can modify the performance of the overall system, therefore it is in the best interest to test such hardware configurations for comparison purposes. Table 4.1 shows the different hardware scenarios that were simulated. The tests were divided into four sections, that test the effects of three different parameters: IO bit width, Hilbert filter length and ROM size. The last section involves the testing of the design that was sent for fabrication, which used a different bit width for Hilbert filter coefficients. Each of the eight test cases were compared using chip area, power dissipation, maximum clock frequency and dynamic range.

Table 4.1: Design Test Scenarios

Case #	IO Bits	ROM		Hilbert	
		Bits	Storage Size	Coefficient Size	Filter Length
1	8	8	256	8	101
2	16	16	256	16	101
3	24	24	256	24	101
4	16	16	256	16	33
5	16	16	256	16	153
6	16	16	128	16	101
7	16	16	512	16	101
8	16	16	256	14	101

4.2 Matlab® Simulation Description

For any signal processing design, it is best to begin with a floating-point Matlab® simulation to verify correct operation and best case results. Along with a floating-point Matlab® simulation created for this research, a fixed-point Matlab® simulation was also created. This fixed-point simulation takes into account each of the parameters from Table 3.1 on page 38. Also added to this fixed-point simulation was the bit width of the *cos* and *sin* outputs as well as the number of samples stored in the ROM. Each of the fixed-point parameters can be modified. The floating- and fixed-point Matlab® simulation code can be found in Appendix A.

4.3 FPGA Testing Description

The FPGA used in this research for prototyping was a Xilinx FPGA demo board. This demo board, as previously mentioned, has two 14-bit DACs with a maximum clock frequency of 105 MHz and two 14-bit ADCs with a maximum clock frequency of 105 MHz. The FPGA also included Xilinx primitives. The key primitives used in the FPGA were a dual-port ROM, for the ROM-based DDS, and dedicated signed multipliers for the digital mixer.

4.3.1 FPGA Test Parameters. Since the number of inputs and outputs for the FGPA testing were limited, the decision was made to use a constant for the scaling value used for the shifting frequency generation as well as a constant for the adder/subtractor, for either a positive or negative frequency shift. The Hilbert filter Q and DSSM outputs were connected to the two DAC outputs. Due to maximum bit width limitations of the ADC and DAC of the FPGA, only one modified test case was tested on the FPGA. The case tested on the FPGA was a modified case #8 with 14-bit IO bit widths, a Hilbert length of 33 taps with 14-bit coefficients and a ROM size of 256 with 14-bit coefficients. This case accounted for the fewer number of IO bits allowed by the ADC and DAC and also for faster clocking due to the short Hilbert

filter. This case was chosen as test case # 8 was the S-ASIC DSSM design sent for fabrication.

4.3.2 FPGA Test Setup. The setup for the FGPA testing was created using the block diagram shown in Figure 4.1, with one signal generator providing the single-tone input. This output is then connected through a splitter to both the FPGA ADC input and the digital oscilloscope. The digital oscilloscope used included a Fast Fourier Transform (FFT) function that would allow the oscilloscope to show the spectral content. The output of the DSSM was connected to the second channel on the oscilloscope for comparison with the input signal. The FPGA used resided on the PCI expansion slot in the Desktop PC, with the image of the complete test setup shown in Figure 4.2.

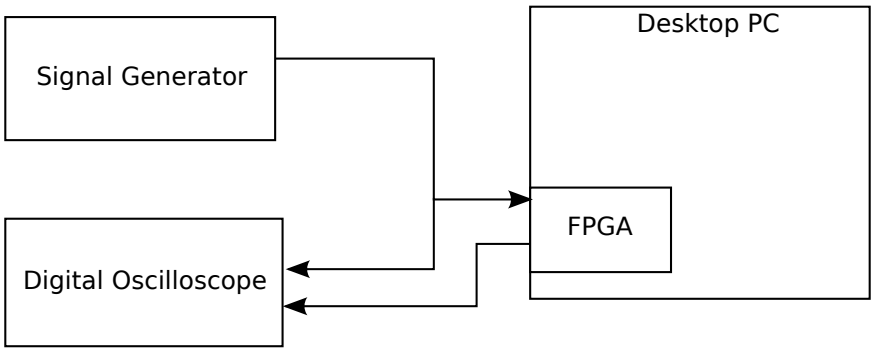


Figure 4.1: FPGA Test Setup

Due to the limited inputs and the use of constants for the shifting frequency generation, different configuration files needed to be generated for each test case. For each test case the FPGA needed to be reconfigured with the new configuration file. The software needed to accomplish this reconfiguration was the Nallatech FUSE software shown in Figure 4.3. This software allowed for the FPGA to be found and configured via the PCI bus. Also included in the FUSE software are software reset switches that were used to reset the FPGA. The FUSE software also allowed for the ADC and DAC clock frequencies to be programmed via software. These values were

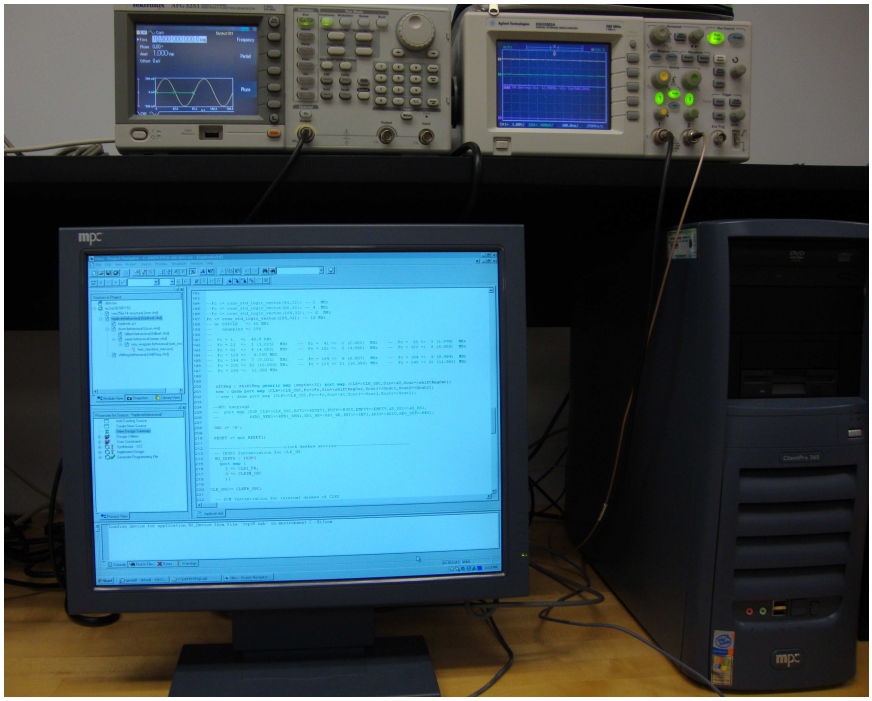


Figure 4.2: FPGA Test Setup Components

set based on the maximum clock frequency reported in the Xilinx post-PAR timing analysis.

4.4 Structured ASIC Testing Description

The S-ASIC testing began with the behavioral VHDL simulation using the Cadence SimVision simulation suite. This simulation created a real valued signal, then converted the signal to an N-bit digitized value, where N is the number of bits of the assumed ADC for the test case. The simulation also converted each output of the DSSM to an integer value. This allowed for the data to be loaded into Matlab® and processed. The processing accomplished in Matlab® was an FFT of the output data to verify the frequency shift as well as determine any spurious signals that were created in the DSSM. The behavioral VHDL simulation testbench is shown in Appendix B.

After the behavioral VHDL simulation was completed the design was synthesized using the provided library for the targeted S-ASIC design. This synthesis step

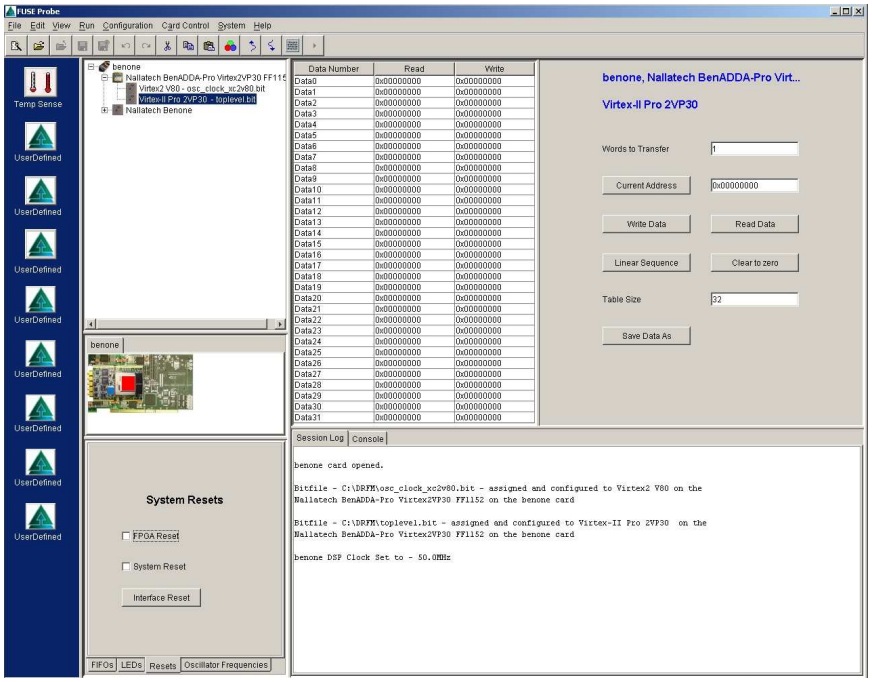


Figure 4.3: FPGA FUSE interface

produced a structural Verilog file. The Cadence RTL Compiler also optimized the design, so a simulation of this design file is crucial to verify that the Cadence RTL Compiler optimized the design incorrectly. The synthesized Verilog design file, along with the model files for the targeted library, were then simulated with a variant of the VHDL testbench used for the behavioral simulation. The generic bit widths were removed from the VHDL behavioral simulation testbench to produce the post-synthesis simulation testbench.

Lastly, after the design was fully placed and routed using the ViAsic ViaPath PAR tool, a post-PAR Verilog design file was generated. This file included the IP ROM modules and, when coupled with the library files, allowed simulation using the same approach as the post synthesis Verilog file. This simulation, in theory, should not differ much from the post synthesis simulation. The only test case for the post-PAR simulation was hardware test case #8. This test case was the only

4.5 Matlab® Floating-Point Simulation Results

Since the different hardware scenarios have no bearing on the floating-point simulation version, it will be presented first. The floating-point plot shown in Figure 4.4 has been normalized for a maximum of 0 dB and assumes a sampling frequency of 75.019 MHz. From the plot, the SFDR for the floating-point simulation is 68.56 dB.

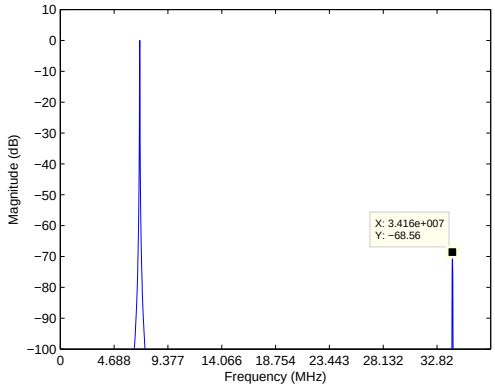


Figure 4.4: Matlab® Floating Point Simulation, f_{in} = 10.35 MHz and f_c = 3.33 MHz

4.6 Fixed-Point Simulation Results

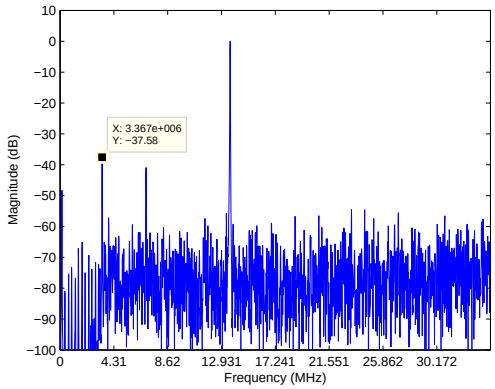
For the fixed point simulations, the goal is to accurately model the DSSM system after the Matlab® floating-point simulation. The floating point simulation proves the functionality of the DSSM, while the fixed-point simulations shows the effects due to quantization. For test cases 1-7, a Matlab® fixed point, behavioral and post-synthesis simulations were completed and are shown for comparison on the following pages. For test case #8, the DSSM designed using those parameters was the design chosen to be fabricated and thus allowed for a post-PAR simulation to be completed. The post-PAR simulation is the closest representation of the S-ASIC design. Each of the simulation plots shows the power spectrum of the first Nyquist band using a 4096-pt FFT after applying an appropriate length Hanning window. Additionally each of the Matlab® fixed point simulations, behavioral simulations and post-synthesis simulations used an input clock period of 14.5 ns (68.966 MHz) for consistent comparisons.

Also completed for the fixed-point simulations were input and shifting frequency sweeps to show the how the dynamic range is affected. The Matlab® simulations were completed with an input frequency sweep from 500 kHz to 37 MHz in 500 kHz increments, while the behavioral simulations were completed with a smaller subset of frequencies. At each input frequency the full range of the ROM-based DDS output frequencies were tested. This frequency sweep varied from test to test based on the size of the ROM. For the behavioral frequency sweep tests completed, only a select few input and shifting frequencies were used. Each of the dynamic range plots were calculated from the power spectrum generated using the same approach as taken for the single frequency simulations.

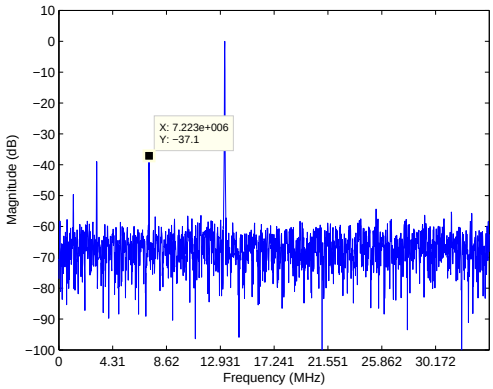
4.6.1 Hardware Test Case #1. Test #1 involved the testing of a limited IO bit width. This test case helps to see what effect an 8-bit input has on the overall DSSM system. Due to quantization limits, an 8-bit input could see at most a 48 dB dynamic range. Along with the 8-bit input, both the Hilbert filter coefficients and the ROM sample bit widths for square multipliers throughout the DSSM. The fixed-point single frequency simulations are shown in Figure 4.5 with the frequency sweep simulations shown in Figure 4.6.

As can be seen from the single frequency plots, the proposed dynamic range of the DSSM using this case is less than 40 dB. From the frequency sweep plots it can be seen that the average dynamic range result is approximately 38 dB. The single frequency as well as the frequency sweep behavioral simulations matches the Matlab® fixed-point simulations. This shows that the designed DSSM matches the analytical simulation and that the results from the Matlab® simulations are good estimates of the DSSM performance. While the simulations match, an average dynamic range of less than 40 dB is not ideal and needs to be increased. The IO bit widths must be increased in order to attain a higher dynamic range.

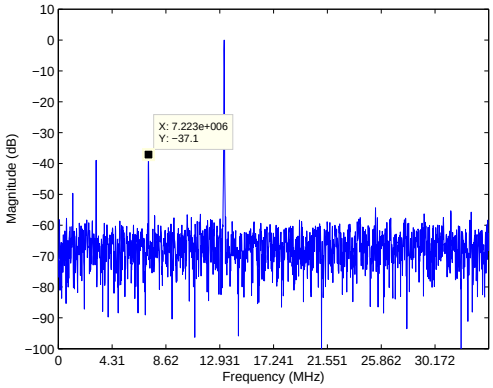
4.6.2 Hardware Test Case #2. Test case #2 further tested the DSSM system by increasing the number of input bits to 16. This increase allows the quantization



(a) Matlab® Fixed-Point Simulation

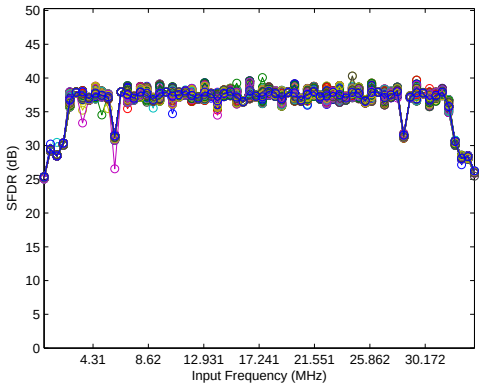


(b) Behavioral Simulation

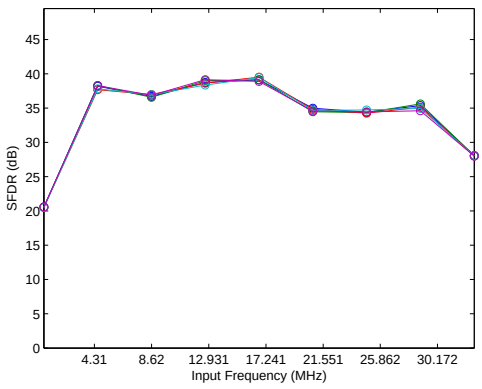


(c) Post-Synthesis Simulation

Figure 4.5: Case #1 with $f_{in} = 10.25$ MHz and $f_c = 3.33$ MHz



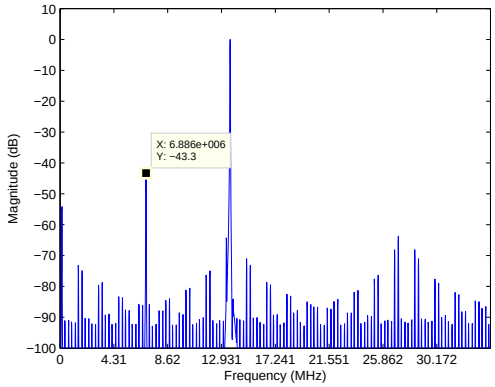
(a) Matlab® Fixed-Point Simulation



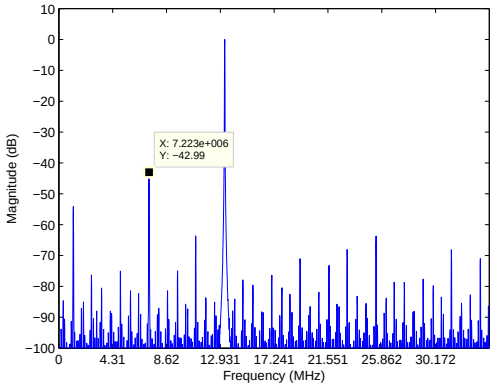
(b) Behavioral Simulation

Figure 4.6: Case #1 Frequency Sweep, $f_{in} = 500$ kHz to 33 MHz and $f_c = 73$ kHz to 9.338 MHz in Increments of (a) 500 and 73 kHz and (b) 4.055 and 2.328 MHz

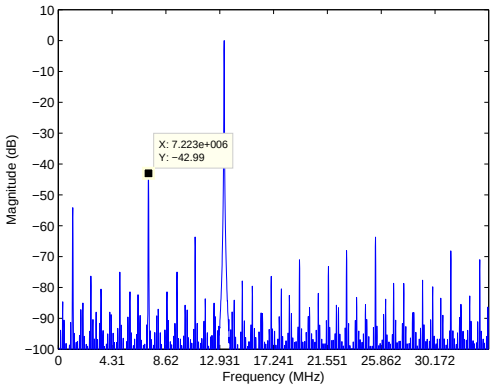
limited dynamic range to grow to a maximum of 96 dB. Similar to test case #1, both the Hilbert coefficients and ROM sample bit widths were equal to 16 to keep the square multipliers. The fixed-point single frequency simulations are shown in Figure 4.7 with the frequency sweep simulations shown in Figure 4.8.



(a) Matlab® Fixed Point Simulation



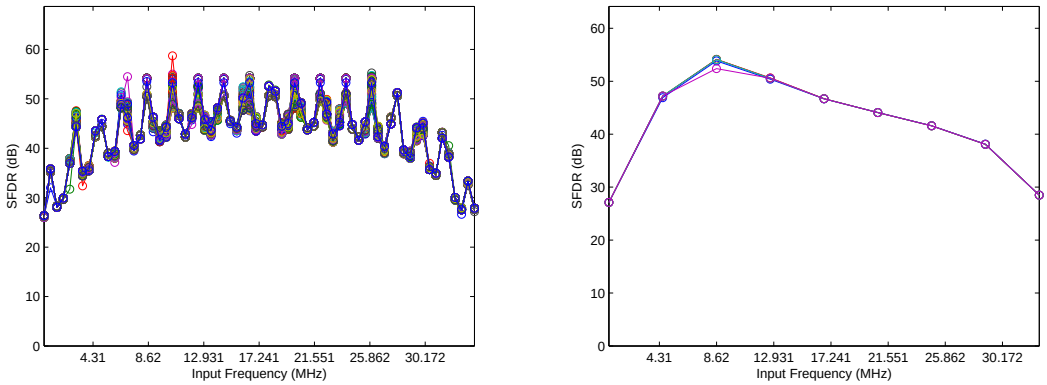
(b) Behavioral Simulation



(c) Post-Synthesis Simulation

Figure 4.7: Case #2 with $f_{in} = 10.25$ MHz and $f_c = 3.33$ MHz

From the plots, it can be seen that the increase in the IO bit width has increased the dynamic range of the system by 6 dB in the single frequency simulation. The average dynamic range found during the frequency sweep simulations was 48 dB for the Matlab® simulations and 45 dB for the behavioral simulations. Increasing the IO bits from 8 to 16 increases the average dynamic range by 8-10 dB. A 10 dB increase



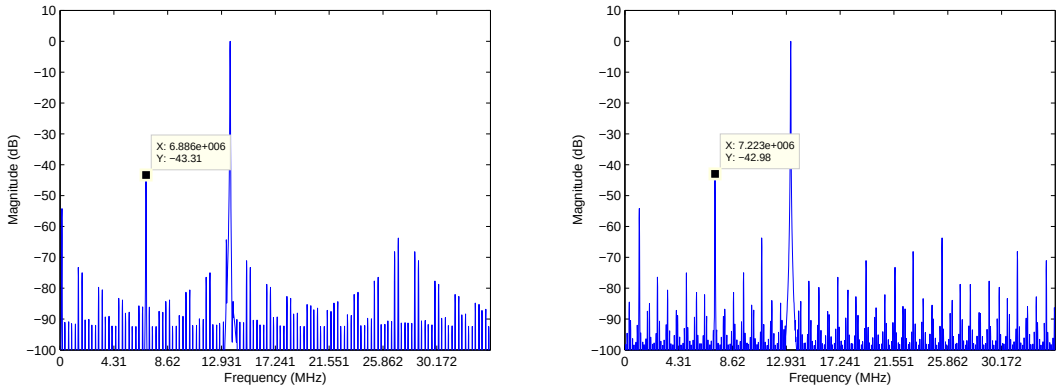
(a) Matlab® Fixed-Point Simulation (b) Behavioral Simulation

Figure 4.8: Case #2 Frequency Sweep, f_{in} = 500 kHz to 33 MHz and f_c = 73 kHz to 9.338 MHz in Increments of (a) 500 and 73 kHz and (b) 4.055 and 2.328 MHz

is substantial and pushes the dynamic range of the DSSM towards 50 dB and making the modules more useful.

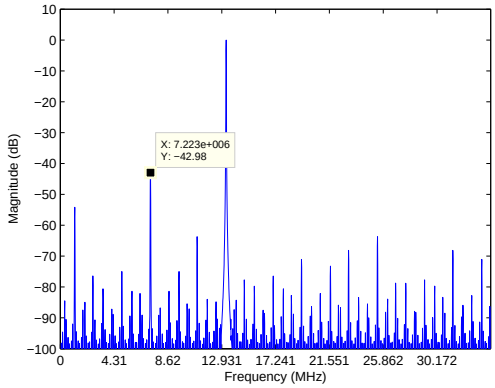
4.6.3 Hardware Test Case #3. Test case #3 took the DSSM one step further and increased the input bit width to 24. This increase allows the quantization limited dynamic range to grow to a maximum of 144 dB. Similar to test cases #1 and #2, both the Hilbert coefficients and ROM sample bit widths were equal to 24 to keep the signed multipliers square. The single frequency simulations completed are shown in Figure 4.9, with the frequency sweep simulation shown in Figure 4.9.

When compared to test case #2, this test case does not show a significant increase in the dynamic range to warrant an increase of IO bit width. Increasing the IO bit width from 16 to 24 only resulted in a dynamic range increase of less than 1 dB in the single-frequency simulations and in the frequency sweep simulations. This shows that there is an asymptotic region somewhere near 16 bits, where an increase in the IO bit width does not produce a noticeable increase in the dynamic range. The cause of this asymptote is in the ROM-based DDS that produces the *sin* and *cos* signals. This DDS stores a very low frequency and has a high noise floor. This high noise floor cancels any dynamic range gains from the increase of the IO bit width.



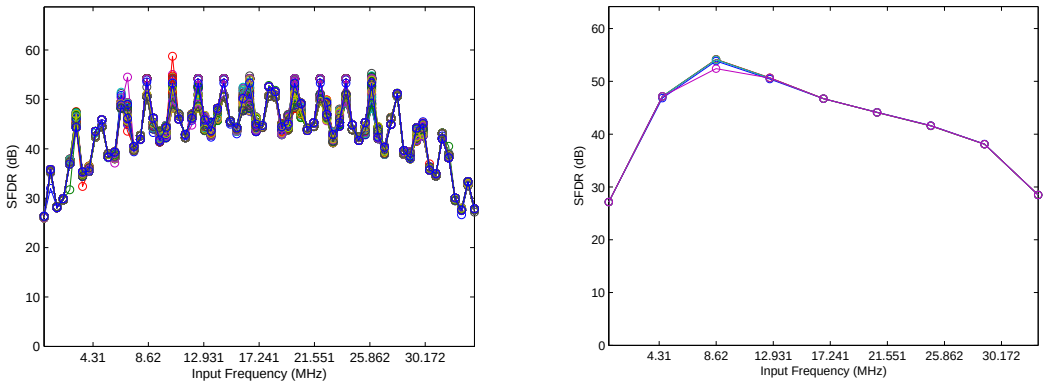
(a) Matlab® Fixed Point Simulation

(b) Behavioral Simulation



(c) Post-Synthesis Simulation

Figure 4.9: Case #3 with $f_{in} = 10.25$ MHz and $f_c = 3.33$ MHz



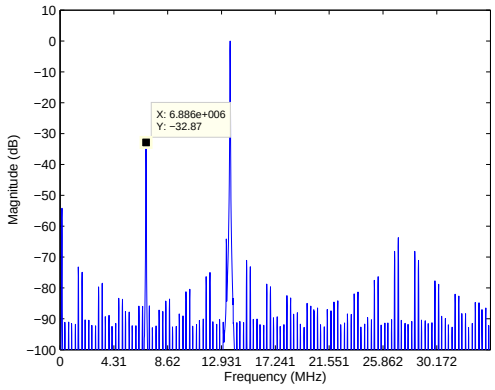
(a) Matlab® Fixed-Point Simulation

(b) Behavioral Simulation

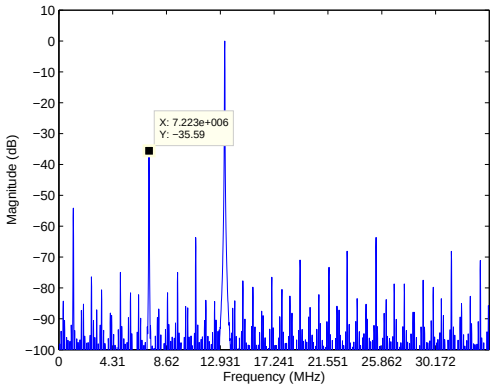
Figure 4.10: Case #3 Frequency Sweep, $f_{in} = 500$ kHz to 33 MHz and $f_c = 73$ kHz to 9.338 MHz in Increments of (a) 500 and 73 kHz and (b) 4.055 and 2.328 MHz

Also, an increase in the IO bit width can produce a situation on the S-ASIC where there is not sufficient IO pads or enough combinational resources available to process the extra bits. Since no noticeable increase was seen from this test, the IO bit width will be kept at 16 bits for the remainder of the tests.

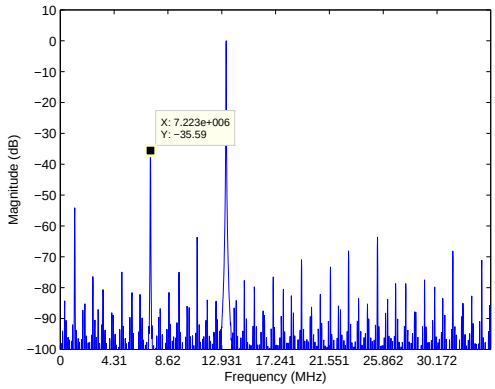
4.6.4 Hardware Test Case #4. Test case #4 returned the IO bit width to 16 and modified the Hilbert filter length to be shorter than the previous three test cases. In this test the Hilbert filter length has been reduced to 33 taps from the 101 taps previously tested. This will increase the passband ripple and show the effects caused by this. The single frequency simulations completed are shown in Figure 4.11, with the frequency sweep simulations shown in Figure 4.12.



(a) Matlab® Fixed Point Simulation

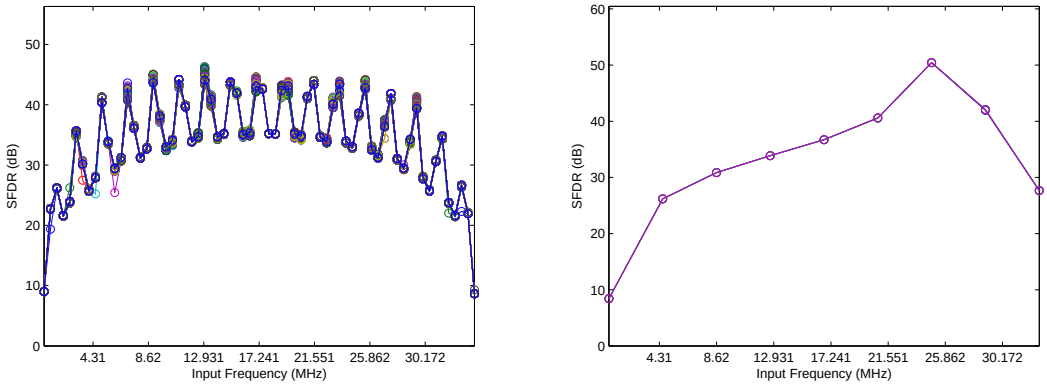


(b) Behavioral Simulation



(c) Post-Synthesis Simulation

Figure 4.11: Case #4 with $f_{in} = 10.25$ MHz and $f_c = 3.33$ MHz



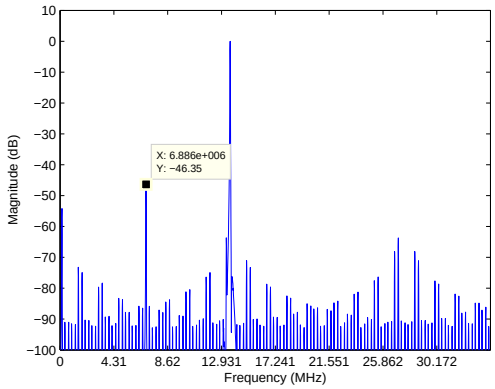
(a) Matlab® Fixed-Point Simulation (b) Behavioral Simulation

Figure 4.12: Case #4 Frequency Sweep, $f_{in} = 500$ kHz to 33 MHz and $f_c = 73$ kHz to 9.338 MHz in Increments of (a) 500 and 73 kHz and (b) 4.055 and 2.328 MHz

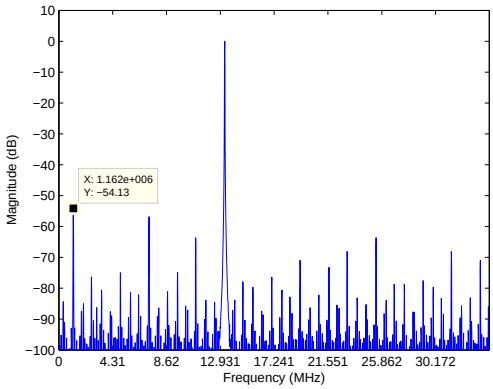
The effect can be seen that a shortened Hilbert filter reduces the dynamic range of the DSSM. Compared to test case #2, which had identical parameters with the exception of the filter length, this test case has a lower dynamic range. The single frequency simulations showed a lower dynamic range, while the frequency sweep simulations showed only a slightly lower average dynamic range. The key issue that can be seen from this test case is the large swings in the dynamic range across the useful band compared to the long filter case. This effect is caused by the Hilbert filter frequency response.

4.6.5 Hardware Test Case #5. Test case #5 kept the IO bit width set to 16 and modified the Hilbert filter length to be longer than the previous four test cases. In this test, the Hilbert filter length has been increased to 153 taps from the 33 and 101 taps previously tested. This will decrease the passband ripple and show any improvements from this change. The three single frequency simulations for this test case are shown in Figure 4.13, while the frequency sweep simulations are shown in Figure 4.14.

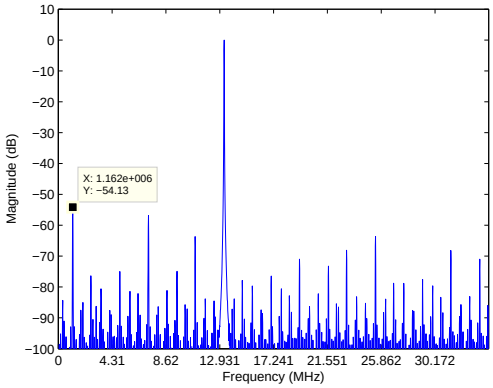
The plots for this test case show that an increase in the Hilbert filter may increase the dynamic range of a single frequency by 10 dB, but the average dynamic



(a) Matlab® Fixed Point Simulation

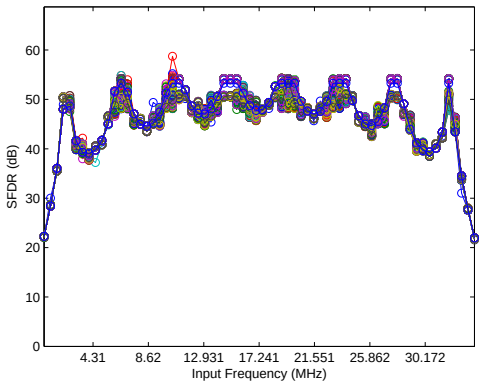


(b) Behavioral Simulation

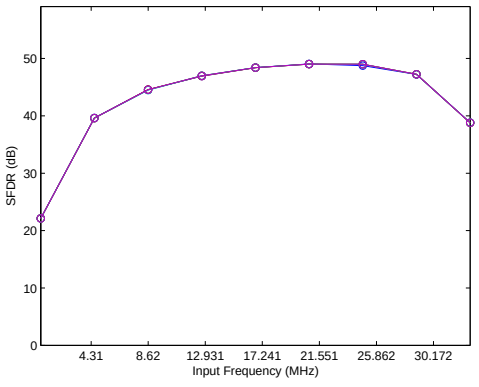


(c) Post-Synthesis Simulation

Figure 4.13: Case #5 with $f_{in} = 10.25$ MHz and $f_c = 3.33$ MHz



(a) Matlab® Fixed-Point Simulation



(b) Behavioral Simulation

Figure 4.14: Case #5 Frequency Sweep, $f_{in} = 500$ kHz to 33 MHz and $f_c = 73$ kHz to 9.338 MHz in Increments of (a) 500 and 73 kHz and (b) 4.055 and 2.328 MHz

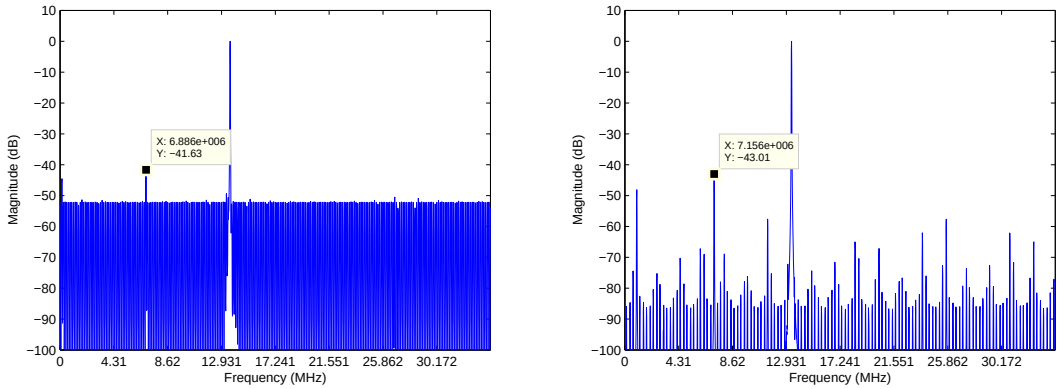
range may only increase by 1-2 dB. While the average dynamic range does not change much for this case, the key attribute of this design is the smooth output dynamic range for the DSSM. The increase in the Hilbert filter length has created fewer ripples in the passband and allows for more use of the spectrum and a slow changing dynamic range.

4.6.6 Hardware Test Case #6. Test case # 6 returned both the IO bit width to 16 and the Hilbert filter length to 101 taps. This test reduced the number of ROM samples stored from 256 to 128, which decreases the granularity of the shifting frequency. The results of this test will show any effects caused by this decrease in frequency granularity. The completed single frequency simulations for this case are shown in Figure 4.15, with the plots showing the frequency sweep simulations are shown in Figure 4.16.

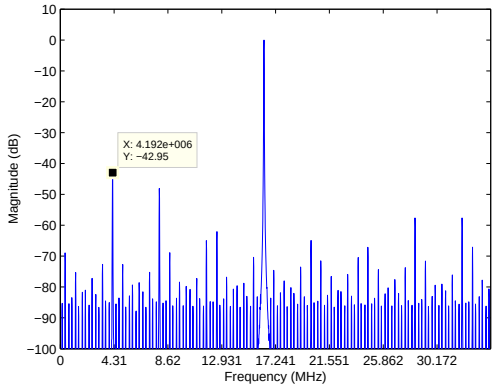
The decrease in ROM storage from 256 down to 128 in this case has reduced both the single-frequency dynamic range and the frequency sweep average dynamic range by approximately 1 dB. While this decrease is negligible, the frequency sweep plots shows the effect of decreasing the ROM size. This test case produces an output dynamic range that is relatively flat, but with a lower average dynamic range. This design can be used if a constant dynamic range is needed and a high average dynamic range is not needed.

4.6.7 Hardware Test Case #7. Test case #7 kept the IO bit width at 16 and the Hilbert filter length at 101 taps. This test increased the number of ROM samples stored from 128 in test #6 to 512, which increases the granularity of the shifting frequency. The results of this test will show any effects caused by this increase in frequency granularity. The single frequency simulations for this test case are shown in Figure 4.17, with the frequency sweep simulations shown in Figure 4.18.

This increase in ROM storage increased the shifting frequency granularity, but increased the dynamic range of the DSSM by less than 1 dB in the single-frequency

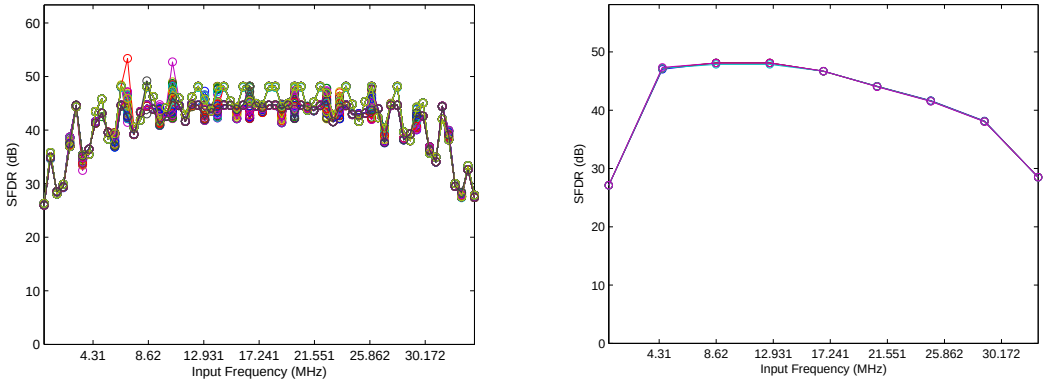


(a) Matlab® Fixed Point Simulation (b) Behavioral Simulation



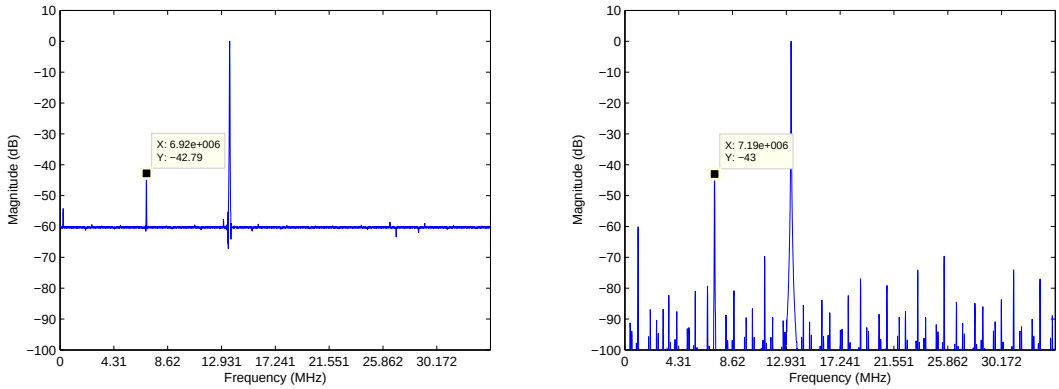
(c) Post-Synthesis Simulation

Figure 4.15: Case #6 with $f_{in} = 10.25$ MHz and $f_c = 3.33$ MHz



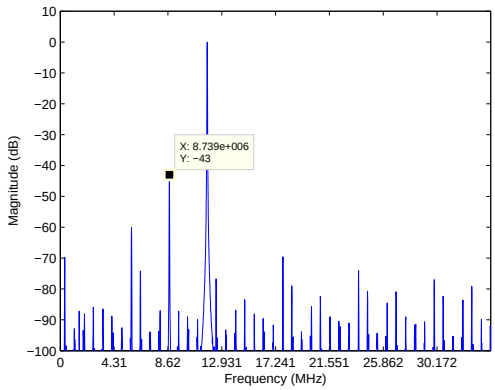
(a) Matlab® Fixed-Point Simulation (b) Behavioral Simulation

Figure 4.16: Case #6 Frequency Sweep, $f_{in} = 500$ kHz to 33 MHz and $f_c = 146$ kHz to 9.302 MHz in Increments of (a) 500 and 146 kHz and (b) 4.055 and 2.345 MHz



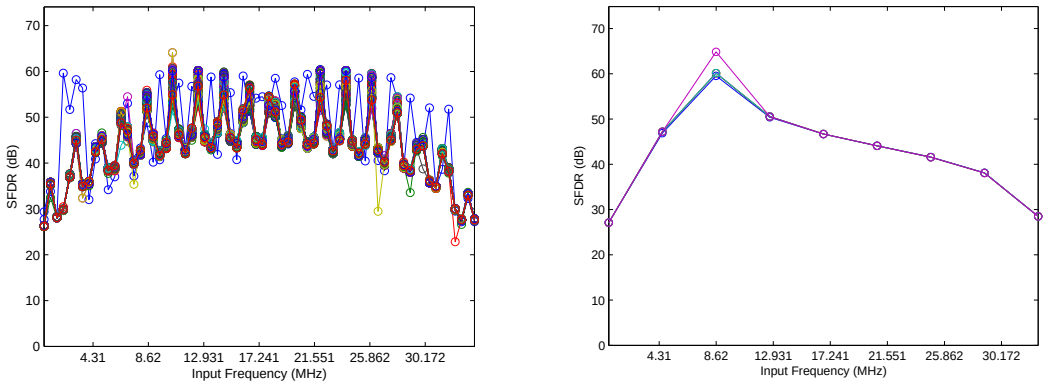
(a) Matlab® Fixed Point Simulation

(b) Behavioral Simulation



(c) Post-Synthesis Simulation

Figure 4.17: Case #7 with $f_{in} = 10.25$ MHz and $f_c = 3.33$ MHz



(a) Matlab® Fixed-Point Simulation

(b) Behavioral Simulation

Figure 4.18: Case #7 Frequency Sweep, $f_{in} = 500$ kHz to 33 MHz and $f_c = 36$ kHz to 9.374 MHz in Increments of (a) 500 and 36 kHz and (b) 4.055 and 2.254 MHz

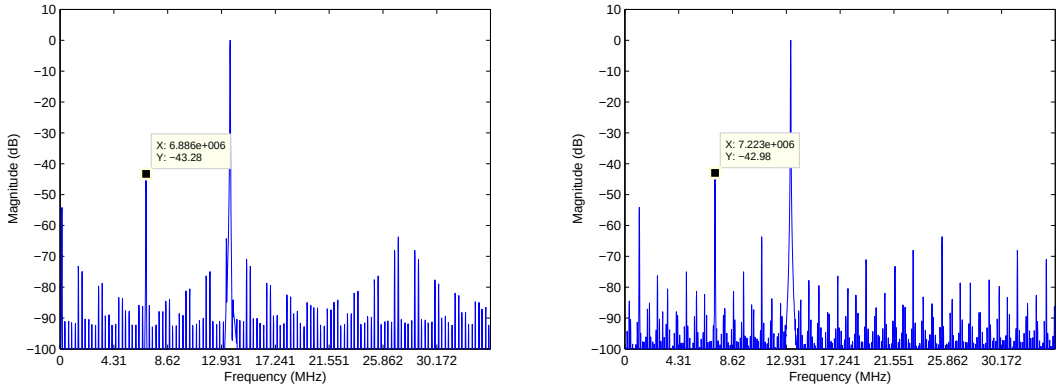
test and increased the average dynamic range by approximately 1 dB in the frequency sweep simulations. This design shows large ripples near the edge of the Hilbert pass-band cutoff as well as a higher average dynamic range. This design could be used whenever the average dynamic range must be high while the ripple in the dynamic range caused by the Hilbert filter can be neglected.

4.6.8 Hardware Test Case #8. Test case #8 kept the IO bit width at 16, the Hilbert filter length at 101 taps and reduced the ROM samples back to 256. This test reduces the number of bits for each of the Hilbert filter coefficients. This creates a non-square 16 $\times$ 14 multiplier in the Hilbert filtering process. The three single-frequency simulations that have been completed for each of the previous test cases, along with the single frequency post-PAR simulation, are shown in Figure 4.19. The frequency sweep simulations are shown in Figure 4.20.

This test case shows the effect of reducing the Hilbert coefficient bit width from 16 to 14, which modifies the size of the constant multipliers in the filter. This could potentially lower the dynamic range, but as seen from the plots, the dynamic range was reduced by less than 1 dB. Both the single-frequency and the frequency sweep simulations showed only a negligible decrease in the dynamic range. This case shows the importance of testing the IO bit width versus Hilbert filter coefficient bit widths to find the best dynamic range results for the desired DSSM, and for this design, the Hilbert coefficient bit width could safely be decreased to reduce the resource utilization, without a significant decrease in dynamic range.

4.7 FPGA Test Results

As previously mentioned, the only hardware test completed on the Xilinx FPGA was the modified test case #8. This test case was specifically designed for the particular FPGA being utilized. The IO bit width was chosen such that both the ADC and DAC were full-scale. Figure 4.21 provides the Xilinx synthesis tool printout for the utilization of the targeted FPGA.



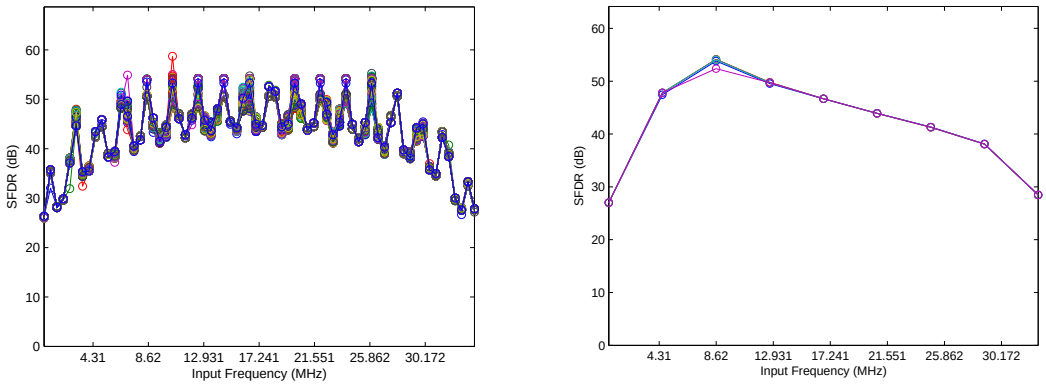
(a) Matlab® Fixed Point Simulation

(b) Behavioral Simulation

(c) Post-Synthesis Simulation

(d) Post-PAR Simulation

Figure 4.19: Case #8 with $f_{in} = 10.25$ MHz and $f_c = 3.33$ MHz



(a) Matlab® Fixed-Point Simulation

(b) Behavioral Simulation

Figure 4.20: Case #8 Frequency Sweep, $f_{in} = 500$ kHz to 33 MHz and $f_c = 73$ kHz to 9.338 MHz in Increments of (a) 500 and 73 kHz and (b) 500 and 100 kHz

```
Device Utilization Summary:

   Number of BUFGMUXs                        1 out of 16      6%
   Number of DCMs                            1 out of 8      12%
   Number of External IOBs                  60 out of 644     9%
      Number of LOCed IOBs                  60 out of 60    100%

   Number of MULT18X18s                     12 out of 136     8%
   Number of RAMB16s                         1 out of 136     1%
   Number of SLICEs                        378 out of 13696   2%
```

Figure 4.21: FPGA Resource Summary Given by Xilinx Synthesis Tool

The maximum clock frequency that can be used in the FPGA design is shown from the timing summary from the Xilinx synthesis tool in Figure 4.22. This clock frequency is the maximum allowable before timing errors begin in the form of setup time violations. The FPGA clock was also used for the ADC and DAC clocks, so the maximum input signal to the FPGA is approximately 25 MHz.

```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 19.940ns (Maximum Frequency: 50.151MHz)
   Minimum input arrival time before clock: 2.273ns
   Maximum output required time after clock: 4.061ns
   Maximum combinational path delay: No path found
```

Figure 4.22: FPGA Timing Summary Given by Xilinx Synthesis Tool

Along with the results that were produced from the Xilinx Synthesis tool, a power estimation tool was available and was used to gather the estimated power used by the FPGA. The results of the power estimation appear in Figure 4.23

Figure 4.24 shows the picture taken from the digital oscilloscope FFT output for the scenario detailed earlier for the FPGA. From this image of the FFT output of the digital oscilloscope, the dynamic range is approximately 33 dB. This approximation is due to each vertical division equal to 25 dB, with the main frequency content appearing near the center and the highest spur located approximately two divisions left of the main frequency.

Power summary		I(mA)	P(mW)
Total estimated power consumption			169

Total Vccint	1.50V	64	97
Total Vccaux	2.50V	10	25
Total Vcco25	2.50V	19	47

Clocks		7	10
Inputs		0	0
Logic		4	5
Outputs			
Vcco25		18	44
Signals		4	6

Quiescent Vccint	1.50V	50	75
Quiescent Vccaux	2.50V	10	25
Quiescent Vcco25	2.50V	1	3

Figure 4.23: FPGA Power Estimation Given by Xilinx Synthesis Tool

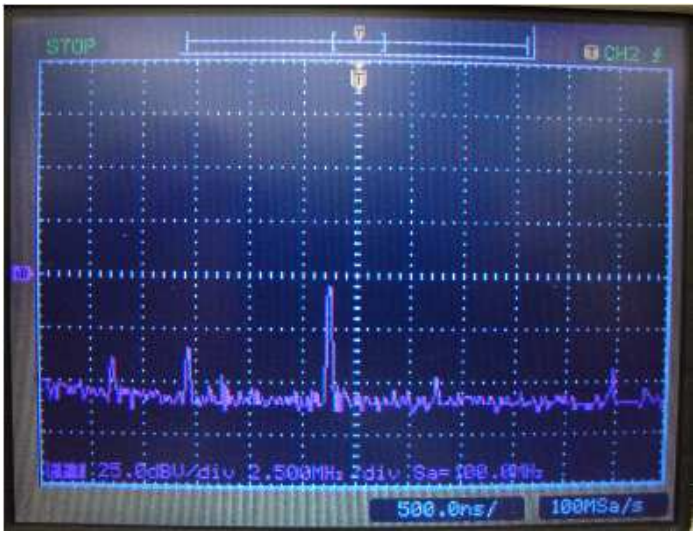


Figure 4.24: FPGA Results, $f_{in} = 10.25$ MHz and $f_c = 3.33$ MHz

4.8 Results

From each of the test cases shown previously, data was collected on the SFDR of the output signals as well as the power, maximum clock frequency and resource utilization for each of the hardware tests. For each of the hardware test case simulations completed, the dynamic range was calculated and is shown in Table 4.2. The dynamic range results from each of the simulation can be compared to the floating point Matlab® simulation completed earlier which resulted in a dynamic range of 68.56 dB.

Table 4.2: Single Frequency Dynamic Range Results (in dB)

Case #	Matlab® Fixed Pt	Behavioral	Post Synthesis	Post PAR
1	37.58	37.10	37.10	N/A
2	43.30	42.99	42.99	N/A
3	43.31	42.98	42.98	N/A
4	32.87	35.59	35.59	N/A
5	46.35	54.13	54.13	N/A
6	41.63	43.01	42.95	N/A
7	42.79	43.00	43.00	N/A
8	43.28	42.98	42.98	42.99

Table 4.3 shows the results of the input and shifting frequency sweeps for both the Matlab® fixed-point simulations and behavioral simulations that were completed. Each of the simulations were compared based on the minimum and maximum dynamic range that was calculated as well as the average dynamic range throughout the sweep, as well as the standard deviation of the dynamic range The input and shift values represent the input and shifting frequency that exhibited either the minimum or maximum SFDR.

Table 4.3: Frequency Sweep Dynamic Range Results

Case #	Matlab® Fixed-Point Simulation								Behavioral Simulation							
	Minimum			Maximum			Average	Standard Deviation	Minimum			Maximum			Average	Standard Deviation
	SFDR	Input	Shift	SFDR	Input	Shift			SFDR	Input	Shift	SFDR	Input	Shift		
1	26.556	6.0	5.994	40.292	24.5	6.061	38.831	1.363	34.265	24.83	4.378	39.533	16.72	4.378	36.742	1.898
2	37.141	6.0	5.994	58.705	10.5	3.502	48.033	4.350	38.088	28.885	8.553	54.141	8.61	4.378	45.979	4.971
3	37.113	6.0	5.994	58.735	10.5	3.502	48.043	4.361	38.091	28.885	8.553	54.168	8.61	2.223	45.993	4.980
4	25.429	6.0	5.994	46.306	13.0	7.274	38.768	4.591	26.182	4.555	2.223	50.445	24.83	8.553	37.24	7.503
5	40.787	5.0	7.206	58.734	10.5	3.502	51.020	3.096	39.609	4.555	6.466	49.053	20.775	0.067	46.392	3.160
6	36.826	6.0	5.792	53.390	7.0	7.004	46.091	2.502	38.034	8.61	0.135	48.132	8.61	8.486	44.823	3.567
7	29.462	26.5	7.981	64.097	10.5	3.468	48.971	5.742	38.089	28.885	4.344	64.859	8.61	8.587	47.012	6.950
8	37.259	6.0	5.994	58.705	10.5	3.502	48.033	4.358	38.098	28.885	8.553	54.135	8.61	2.233	45.878	4.938

All SFDR Results are in dB while all frequencies are in MHz

From synthesis to placement, routing and post-PAR static timing analysis, several timing reports were given. The Cadence RTL Compiler provided an estimate on the maximum clock frequency as did the ViAsic ViaPath PAR tool and Incentia STA tool. Also included in the timing reports is the maximum clock frequency obtained through post-synthesis and post-PAR simulations using the Cadence SimVision simulation suite. The Cadence RTL Compiler and ViAsic ViaPath PAR tool both used estimates on the path lengths to determine the maximum clock frequency. One would expect each of the reported clock frequencies to be clustered together near a common frequency. For the timing reports shown in Table 4.4, they are clustered together in a 23 MHz range. This table also shows the downward progression of the reported clock frequency from synthesis to post-PAR static timing analysis.

Table 4.4: S-ASIC Timing Reports

Source	Maximum Clock Reported
Post-Synthesis Timing Report	90.114 MHz
Post-Synthesis Simulation	66.667 MHz
Post-PAR Timing Report	80.431 MHz
Post-PAR Simulation	68.966 MHz
Post-PAR STA Timing Report	76.045 MHz

4.9 Comparison

The hardware comparison of the different hardware test cases for the S-ASIC is shown in Table 4.5. The power values shown in the table were gathered from the post-synthesis power report given by the Cadence RTL Compiler. The maximum clock frequency shown in the table was gathered from the synthesis tool output while SFDR was calculated using the post-synthesis simulation data shown earlier. Since the limiting factor on the S-ASIC was the amount of DFFs present, the resource utilization shown in the table represents the percentage of DFFs used out of the total number of DFFs available on the S-ASIC chip.

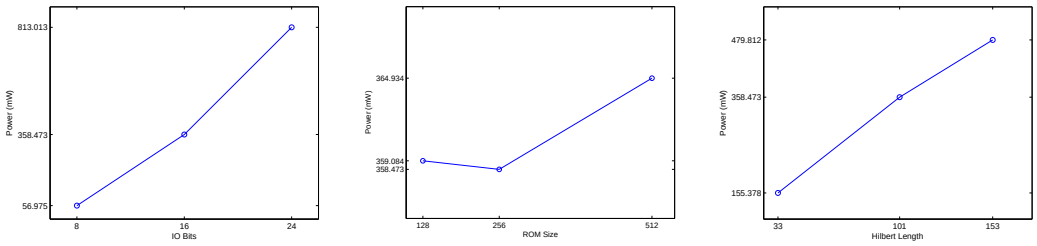
4.9.1 Design Parameter Effects. When varying the design parameters, several output parameters were modified. The parameters in question are power, maxi-

Table 4.5: S-ASIC Hardware Comparison

Parameter	Test Case #							
	1	2	3	4	5	6	7	8
Power (mW)	56.975	358.473	813.013	155.378	479.812	359.084	364.934	328.533
max f_{clk} (MHz)	92.199	90.025	87.382	90.074	89.622	90.025	90.057	90.114
Resource Utilization (%)	24.97	64.08	113.5	41.7	81.18	64.08	64.06	64.7
SFDR (dB)	37.1	42.99	42.98	35.59	54.13	42.95	43.0	42.98

mum clock frequency, resource utilization and SFDR. This is not an all inclusive list of different parameters, but were chosen for comparison purposes as they fit into many engineers' calculations. Each of the parameters are featured in their own figure based on the three design parameters that were varied around the design that was sent for fabrication. These parameters were the IO bit width, the ROM storage capacity for the DDS and the Hilbert filter length. Each of the three different design parameters modified the designed DSSM in some fashion, and Figures 4.25-4.28 show this.

The first to look at is the power for the S-ASIC design and was calculated from the Cadence RTL Compiler output files. As can be seen from the the plots in Figure 4.25, both the length of the Hilbert filter as well as the IO bit width greatly influenced the power needed for the DSSM.

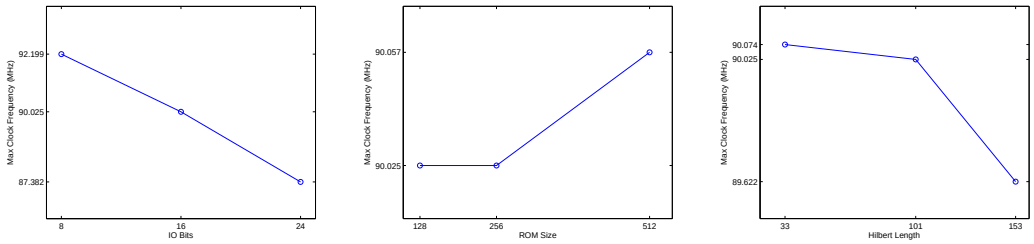


(a) IO Bits (b) ROM Storage (c) Hilbert Length

Figure 4.25: Design Parameters vs. Power Comparison

The second parameter focused on was the maximum clock frequency of the DSSM, which was captured from the Cadence RTL Compiler output files. From the plots it is easy to see that the key parameter that effects the clock speed is the IO bit width. The reason is in the size of the needed multipliers and adders in the DSSM to

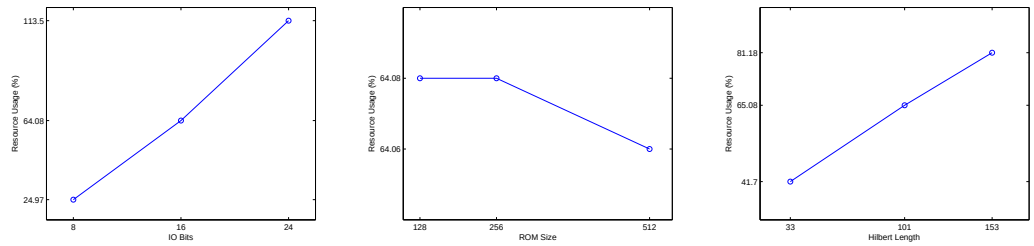
support an increase in bit widths. The plots showing the comparison for the maximum synthesized clock frequency are shown in Figure 4.26.



(a) IO Bits (b) ROM Storage (c) Hilbert Length

Figure 4.26: Design Parameters vs. Maximum Clock Frequency Comparison

The resource utilization parameter comes from the availability of DFFs on the S-ASIC. The measurement used for this comparison was to show the percentage of used DFFs to the number of available DFFs. This comparison shows that both the IO bit width and the Hilbert filter length contribute to the resource usage of the DSSM. The IO bit width has slope greater than one on the the resource usage while the Hilbert filter has a slope less than one for the percentage of used DFFs. The plots in Figure 4.27 shows the effects of the parameters on the resource usage of the DSSM.

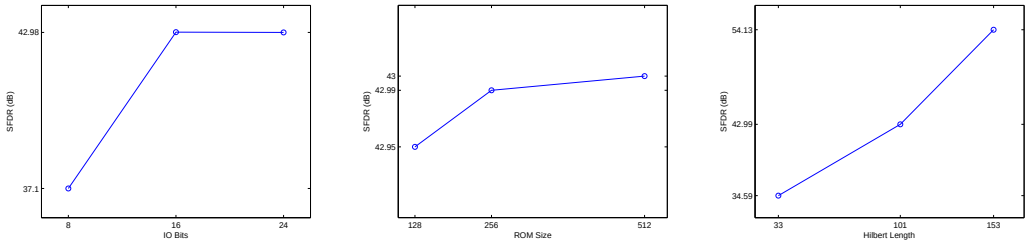


(a) IO Bits (b) ROM Storage (c) Hilbert Length

Figure 4.27: Design Parameters vs. Resource Utilization Comparison

Lastly, the SFDR was looked at from the post-synthesis simulations of each of the design parameter cases. The design parameters that contribute the most to the dynamic range performance of the DSSM is the IO bit width and the Hilbert filter length. An increase in the IO bit width from 8 bits to 16 bits increased the dynamic

range by 6 dB, while an increase in filter length from 33 taps to 101 taps increased the dynamic range by 8.4 dB, and an increase from 101 taps to 153 taps increased the dynamic range further by 11.1 dB. An increase in the IO bit width from 16 bits to 24 bits did not increase the dynamic range. From the plots in Figure 4.28, it is easy to see the Hilbert filter length effects the dynamic range more than the ROM size or the IO bit width.



(a) IO Bits (b) ROM Storage (c) Hilbert Length

Figure 4.28: Design Parameters vs. SFDR Comparison

From the preceding plots, one can conclude that the most influential parameter to the size, power and speed of the DSSM is the IO bit width. An increase in bit width increases power, decreases the clock frequency and increases the resource utilization greater than linear fashion. The second parameter that plays a role in the DSSM performance is the Hilbert filter length. The filter's length contributes to the resource utilization and power, more than it does to the clock frequency.

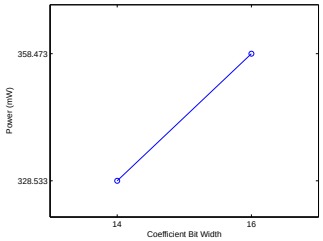
The key parameter that effects the dynamic range of the DSSM is the Hilbert filter length more than the IO bit width. Due to limitations on the number of bits for ADCs and DACs, the number of IO bits has a maximum, while the length of the Hilbert filter is limited by the number of DFFs available and the target clock frequency for the DSSM. If an increase in the Hilbert filter, that is, an increase in power and an increase in resource usage, is possible, the dynamic range of the DSSM can be increased.

The size of the ROM used in the DDS does not effect the DSSM performance in either a positive or negative way. The ROM size effects on power, clock frequency,

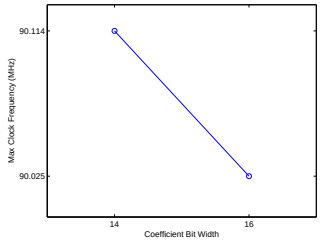
resource usage and dynamic range are negligible compared to the other design parameters. While the effects cannot be seen from the results presented, the effects of the ROM size can be seen on the frequency resolution of the shifting frequency. As the ROM size decreases the frequency resolution decreases and alternatively, when the ROM size increases, so does the frequency resolution. The ROM size should be chosen based on the desired frequency resolution more than any of the parameters compared here.

Test case #8 involved modifying the Hilbert filter coefficient bit width in case #2 to 14 bits. This creates non-square multipliers in the Hilbert filter and, as discussed previously, should decrease the power and resource usage while increasing the clock frequency. The comparison of cases #2 and #8 are shown in Figure 4.29. It can be seen that the decrease in Hilbert filter coefficient bit width decreased power by 8%, increased the maximum clock frequency by approximately 1%, increased resource usage by approximately 1% and increased the dynamic range of the DSSM by 8%. The modification from 16 to 14 bit Hilbert coefficients contributed to a more useful DSSM with reduced power and increased dynamic range without a major hit to resource usage and clock frequency.

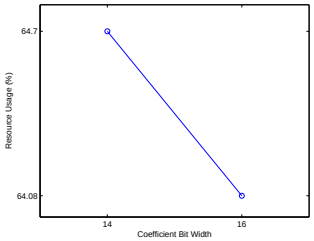
4.9.2 S-ASIC Versus FPGA. To compare the S-ASIC to the FPGA used for prototyping involved a few changes. The changes made involved the IO bits and the Hilbert length. The ADC and DAC on the FPGA board were both limited to 14 bits that limited the IO bits to 14 and the speed of the FPGA could not handle a large Hilbert filter length. The filter length used was 33 taps. From the results shown in the previous section, the dynamic range of the FPGA will be lower due to the shorter Hilbert filter, which must be accounted for in a fair comparison. The maximum clock frequency for the S-ASIC was also taken from maximum clock frequency achieved through the post-PAR simulation while the SFDR calculation for the S-ASIC was also taken from the post-PAR simulation. The comparison values are shown in Table 4.6.



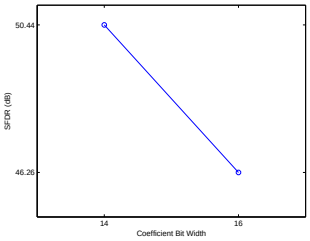
(a) Power



(b) Clock Frequency



(c) Resource Utilization



(d) SFDR

Figure 4.29: Comparison of Test Cases # 2 and 8

Table 4.6: FPGA vs. S-ASIC Comparison

Parameter	S-ASIC	FPGA	Improvement
Power (mW)	328.533	169	-1.94x
max f_{clk} (MHz)	68.965	50.151	1.37x
Resource Utilization (%)	64.7	2	-32.3x
SFDR (dB)	42.99	33	9.99 dB

As can be seen from the preceding table, the S-ASIC power consumption increased by approximately a factor of 2 over the FPGA and had a resource utilization increase by a factor of 30. Due to the Hilbert filter length difference addressed earlier for the FPGA DSSM design, the results presented earlier show that the SFDR would have increased by a factor of 1.39, the power consumption would be increased by a factor of 2.3, the maximum clock frequency would have decreased by a factor of 0.01 and the resource usage would have increased by a factor of 1.56. When accounting for these modifications, the comparison translates into what is seen in Table 4.7.

As can be seen from this compensated table, the resource usage of the FPGA was drastically lower than that of the S-ASIC design, yet the power and maximum clock frequencies were higher and the dynamic range differed only by 2.88 dB. This suggests

Table 4.7: FPGA vs. S-ASIC Comparison With Compensation

Parameter	S-ASIC	FPGA	Improvement
Power (mW)	328.533	388.7	1.18x
max f_{clk} (MHz)	68.965	46.649	1.47x
Resource Utilization (%)	64.7	3.12	-20.7x
SFDR (dB)	42.99	45.87	-2.88 dB

that even though this design has been radiation hardened, it can still perform as well as an FPGA while also keeping the power lower and increasing the clock frequency.

4.9.3 Hardened Versus Non-Hardened DSSM. Also for comparison purposes, the DSSM designed through this research has been compared to a non-hardened DSSM using similar 90nm CMOS technology. The three designs were compared using identical hardware and input test conditions as detailed eariler. The results gathered for this comparison were from test case #8, the design being fabricated on the S-ASIC. The power parameter was captured from Cadence RTL Compiler output, the area from the ViAsic ViaPath PAR tool output, the SFDR from the post-PAR simulation and the maximum clock frequency from the Incentia STA tool output using the post-PAR design file. Table 4.8 shows the results gathered for this comparison. The non-hardened DSSM process library did not have accurate timing models for the digital gates and could not be simulated the same as the DSSM from this design. Both the DSSM from this research and the non-hardened DSSM used a 20 MHz clock frequency as it was the fastest clock frequency the non-hardened DSSM could be simulated at. The dynamic range result from the table shows the average dynamic range found among a series of input frequencies.

Table 4.8: Hardened vs. Non-Hardened DSSM Comparison

Parameter	This Research	Non-Hardened [16]	Improvement
Power (mW)	328.533	54.535	-6.06x
Area (mm^2)	11.225	5.045	-2.23x
max f_{clk} (MHz)	76.045	78.49*	-0.04x
SFDR (dB)	44.574	50.698	-6.124 dB

*Data from synthesis timing report

It can be seen that this DSSM design has decreased performance versus the non-hardened DSSM. From the radiation hardening by design process, an increase in area and power and a decrease in the maximum clock frequency should be seen compared to the non-hardened DSSM, which was the case for this test. This DSSM design also showed a small decrease in the dynamic range from the non-hardened design. This decrease in dynamic range was expected due to the radiation hardened library. While the maximum clock frequency of the non-hardened DSSM was comparable to the radiation hardened DSSM from this research, the timing models for the non-hardened were not accurate. This inaccuracy created a non-ideal comparison between the two DSSM designs. Had more accurate timing models been available for the non-hardened DSSM design, a better comparison could have been completed.

V. Conclusions

A radiation hardened by design DSSM has been successfully designed, implemented and tested for an S-ASIC. Through the application of the theory behind the frequency shift principle of the SSM in the analog domain and transitioning to the digital domain, a tested design was created that would implement a functioning DSSM. Through the utilization of an S-ASIC, the design cycle was drastically reduced by the use of a standard cell library for synthesis and a one mask approach to the fabrication of the design. This design has shown how a radiation hardened DSSM can be constructed using commercially available tools and processes and how it compares to a non-hardened DSSM.

5.1 Lessons Learned

Through the theory, methodology and simulations, many lessons were learned through this research. The key lesson learned from this research is that the STA completed by the synthesizing and PAR software may not be accurate. The timing estimates provided by the software tools may give operating clock speeds higher or lower than what was expected depending on the design, software version and in some cases, the computing system the software tools are executing on. This creates a situation where each synthesized, placed and routed design must be simulated several times to ensure that the circuit can operate at the clock speed intended. Also involved in the timing estimates is the library timing information. This timing information provides the capacitive and resistive values for the designed digital gates used in the design. These parameters are sufficient enough to calculate the propagation delay of the digital gate and provide the STA with the timing information it needs for its timing estimate. If the library timing information is not accurate the STA tool timing estimate cannot be trusted until the designed chip can be fabricated and then functionally tested for timing.

5.2 Future Work

As with any research, there is always more that can be done in terms of testing and/or modifications. Due to the delay in the fabrication of the DSSM chip, the final functional testing on the DSSM, as well as the radiation testing of the DSSM, would need to be completed. To aid in processing more RF spectrum, bandpass filters can be added after the ADC to create a filterbank DSSM. This filterbank would separate sections of the RF spectrum into manageable slices for each individual DSSM. To aid in the speed of the DSSM, a pipelined digital Hilbert filter can be utilized. Lastly, adding a signal storage and retrieval system to control which signals are processing and output could be added to create the overall DRFM structure.

5.2.1 Filterbank Approach. Due to the Nyquist theorem on sampling rates being twice that of the bandwidth of interest, the DSSM speed must increase as the RF spectrum of interest increases. The filterbank approach adds the ability to utilize the DSSM designed in this research with its given clock speed multiple times to create the same effect. The basic idea of the filterbank is to use BPFs to select a portion of the RF spectrum, digitized by the ADC, such that the bandwidth can be processed in the DSSM at a reduced frequency. The larger the captured RF spectrum, the more filterbank paths need to be created. Figure 5.1 shows the structure of such a filterbank using the DSSM from this research.

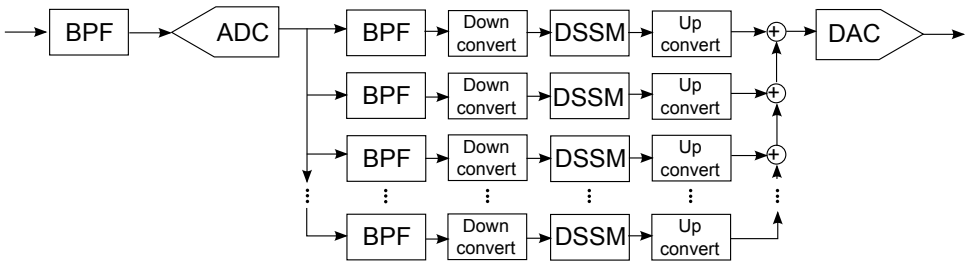


Figure 5.1: Filterbank DRFM Structure

5.2.2 Pipelined Digital Hilbert Filter. While the filterbank approach can help increase the RF spectrum examined by the entire system, adding a pipelined

digital Hilbert filter could increase the RF spectrum examined by a single DSSM. The same basic filter structure can be utilized with the addition of DFFs between the adders in the adder tree. If a new design includes more DFFs, then the entire tree may be pipelined as shown in Figure 5.2. If the number of DFFs is limited, as it was in this DSSM design, there should be several iterations completed to insert the needed pipeline stages in strategic locations such that the speed of the DSSM can be increased without the over utilization of the available DFFs. This addition to the DSSM designed from this research would allow the DSSM clock rate to be increased and would allow more of the RF spectrum to be processed. If coupled with the filterbank approach, the complete system would be capable of a much larger section of the RF spectrum and could be used in many applications.

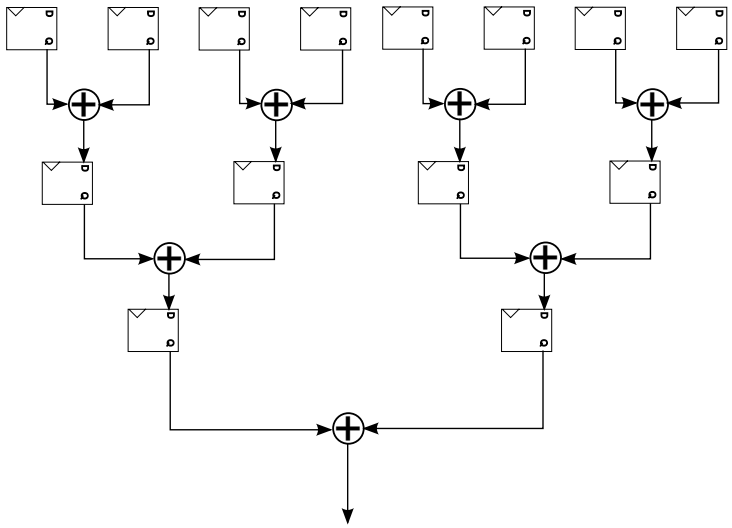


Figure 5.2: Pipelined Adder Tree for Hilbert Filter

This addition to the digital Hilbert filter would require a complete revision of the VHDL code for the Hilbert filter. The VHDL code designed in this research was written as a behavioral design, and with the addition of the pipeline stages, the design would need to be converted to a structural design that would allow for different placements of the pipeline stages. The different placements would arise whenever there are not enough DFFs in the targeted design.

5.2.3 Signal Storage and Retrieval System. Due to the limited memory elements available on the S-ASIC chip, it was decided to neglect any control structure on the storage and retrieval of digitized signals, and focus on the DSSM portion of the DRFM. This storage system would allow for a signal to be stored for a defined amount of time and retransmitted at the discretion of the controller. This feature would allow a captured signal to be stored and a frequency shifted version of that signal transmitted at various intervals in time. Storing a signal for later retrieval allows for the reproduction of a signal that has since disappeared and also to store the signal for more detailed signal processing at a later time.

Appendix A. Matlab® Simulations and Comparison Scripts

The Matlab® simulation functions and script as well as the Matlab® comparison script is found in this appendix. The simulation script creates a double-precision representation of an ideal input and output signal as well as a fixed-point representation of the same signals. The comparison script reads in the simulation data files for the VHDL design simulation, the post-synthesized VHDL simulation and the post-Place and Route simulation and plots the corresponding RF spectrum for comparison against the ideal Matlab® simulation. The dynamic range helper function *findDR* used in the *drfm_sim* simulation script is also found in this appendix.

A.1 Matlab® Floating-Point DSSM Simulation

```
function [OUT,COS,SIN,I,Q] = drfm_sim_float(sig,Fc,Fs,N,addSub)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DRFM Matlab Simulation
%
% Created by: Thomas Pemberton
%
% Last Updated on: 10 Nov 2009
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = (0:N-1)*1/Fs;

%% create the shifting signals
COS = cos(2*pi*Fc*n);
SIN = sin(2*pi*Fc*n);

%% perform Hilbert transform on data
h = hilbert(sig);

%% extract the Real and Imag from the output
I = real(h);
Q = imag(h);

%% perform the mixing
Imix = I.*COS;
Qmix = Q.*SIN;

%% subtract Imix and Qmix for the final output
if (strcmp(addSub,'sub') == 1), OUT = Imix+Qmix; end;
if (strcmp(addSub,'add') == 1), OUT = Imix-Qmix; end;
```

A.2 Matlab® Fixed-Point DSSM Simulation

```
function [OUT,sig,COS,Q] = drfm_sim_fixed(sig,Fs,params,addSub)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DRFM Matlab Simulation
%
% Created by: Thomas Pemberton
%
% Last Updated on: 10 Nov 2009
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% check if all parameters were given
if (nargin < 4), error('Requires 4 arguments'); end;

%% generate the fixed point input signal
sig = ceil(sig*2^(params.ADC_bits-1));

%% set the shifting frequency signal
Fc = ceil(params.Fc/((Fs/4)/params.ROM_size));

%% create the shifting signals
ROM = ceil(2^(params.ROM_bits-1)*cos(2*pi*(0:params.ROM_size-1)*.25/
   params.ROM_size));

%% create the hilbert coeffecients
h = zeros(1,params.HILBERT_length);
i = 1;
while i<= params.HILBERT_length
    h(1,i) = 2/(pi*(i-(params.HILBERT_length-1)/2));
    i=i+2;
end
h = ceil(2^(params.HILBERT_bits-1)*h);

%% filter the incoming signal and scale the signal appropriately
a = conv(sig,h);
a = a/(2^(params.HILBERT_bits-1));

%% extract the I and Q channels
I = sig((params.HILBERT_length-1)/2:end);
Q = a(params.HILBERT_length-2:end-params.HILBERT_length);
I = I(1:length(Q));

%% create the appropriate COS and SIN for mixing
ii=1;
address = 1;
while ii<=length(I)
    if (address > params.ROM_size*4)
        address = address-params.ROM_size*4;
    end;
    if ( (address >= 1) && (address < params.ROM_size) )
        addr = address;
        COS(ii) = ROM(addr+1);
        SIN(ii) = ROM(params.ROM_size-addr);
    elseif ( (address >= params.ROM_size) && (address < params.
       ROM_size*2) )
        addr = address-params.ROM_size;
        COS(ii) = -1*ROM(params.ROM_size-addr);
        SIN(ii) = ROM(addr+1);
    elseif ( (address >= params.ROM_size*2) && (address < params.
       ROM_size*3) )
        addr = address-params.ROM_size*2;
        COS(ii) = -1*ROM(addr+1);
        SIN(ii) = -1*ROM(params.ROM_size-addr);
    elseif ( (address >= params.ROM_size*3) && (address < params.
       ROM_size*4) )
        addr = address-params.ROM_size*3;
        COS(ii) = ROM(params.ROM_size-addr);
        SIN(ii) = -1*ROM(addr+1);
    end;
    address = address+Fc;
    ii = ii+1;
end

%% perform the mixing
Imix = I.*COS; Imix = Imix/(2^params.DAC_bits);
Qmix = Q.*SIN; Qmix = Qmix/(2^params.DAC_bits);
```

```
%% add/subtract Imix and Qmix for the final output
if (strcmp(addSub,'sub') == 1)  OUT = Imix-Qmix; end;
if (strcmp(addSub,'add') == 1)  OUT = Imix+Qmix; end;
```

A.3 Matlab® Floating-Point vs. Fixed-Point DSSM Simulation

```
%% clean environment
clear all; close all; clc;

%% setup parameters
Ts = 14.5e-9;
Fs = 1/Ts;
N = 4096;   % N-pt FFT
n = (0:N-1)*Ts;
f = 0:Fs/N:Fs-Fs/N;
F1 = 10.25e6;   P1 =   0;    % frequency and power(dBm) of Tone 1
A1 = sqrt(10^(P1/10)*50*1e-3)/0.707;
Fc = 3.33e6;

%% define the upper (add) or lower (sub) SSM
addSub = 'sub';
%addSub = 'add';

%% define plotting for fixed and floating point
plotFloat = 0;
plotFixed = 0;
plotSFDR = 1;
normalize = 1;
windowData = 1;
power = 1;
freqSweep = 1;
sigFloor = -100; % floor of the sfdr plots

%% set the fixed point parameters
params.ADC_bits = 16;
params.DAC_bits = params.ADC_bits;
params.HILBERT_length = 101;
params.HILBERT_bits = 16;
params.ROM_size = 512;
params.ROM_bits = 16;

%% define the frequency sweep
startFreq = 0.5e6;
endFreq = 34e6;
freqRes = 0.5e6;
if (freqSweep == 1)
    freq = startFreq:freqRes:endFreq;
else
    freq = F1;
end

%% define the shifting frequency sweep
shiftFreqRes = (Fs/4)/params.ROM_size;
startShiftFreq = shiftFreqRes;
endShiftFreq = (params.ROM_size/2-1)*shiftFreqRes;
if (freqSweep == 1)
    shiftFreq = startShiftFreq:shiftFreqRes:endShiftFreq;
else
    shiftFreq = Fc;
end

%% define the output sfdr found
if (freqSweep == 1)
    OUT_FLOAT_sfdr = zeros(length(freq),length(shiftFreq));
    OUT_FIXED_sfdr = zeros(length(freq),length(shiftFreq));
end

%% loop through all of the input frequencies
```

```
for jj=1:length(freq)

    %% create the test signal
    sig = A1*cos(2*pi*freq(jj)*n);

    %% loop through all of the shifting frequencies
    for kk=1:length(shiftFreq)

        %% set the Fc value
        Fc = shiftFreq(kk);
        params.Fc = Fc;

        %% now test both the fixed and floating point versions
        [out_float,COS_f,SIN_f,I_f,Q_f] = drfm_sim_float(sig,Fc,Fs,N,
           addSub);
        [out_fixed,sig_out,COS,Q] = drfm_sim_fixed(sig,Fs,params,
           addSub);

        %% lower the output of the floating point version for
           matching to the
        %fixed point version
        out_float = out_float/(2^2); % 4 bits (2 for each I and Q)

        %% convert the fixed point outputs back to floats
        out_fixed = out_fixed/2^params.DAC_bits;
        sig_out = sig_out/2^params.ADC_bits;

        %% generate the FFTs of each and scale to 0 dB
        if windowData == 1
            OUT_FIXED = abs(fft(out_fixed.*hanning(length(out_fixed))
               ',N));
        else
            OUT_FIXED = abs(fft(out_fixed,N));
        end
        if power == 1
            OUT_FIXED = OUT_FIXED.^2/N^2;
        end
        if normalize == 1
            OUT_FIXED = OUT_FIXED/max(OUT_FIXED);
        end;
        if power == 1
            OUT_FIXED = 10*log10(OUT_FIXED);
        else
            OUT_FIXED = 20*log10(OUT_FIXED);
        end
        if windowData == 1
             OUT_FLOAT = abs(fft(out_float.*hanning(length(out_float)
                )',N));
        else
             OUT_FLOAT = abs(fft(out_float,N));
        end
        if power == 1
             OUT_FLOAT = OUT_FLOAT.^2/N^2;
        end
        if normalize == 1
             OUT_FLOAT = OUT_FLOAT/max(OUT_FLOAT);
        end;
        if power == 1
             OUT_FLOAT = 10*log10(OUT_FLOAT);
        else
             OUT_FLOAT = 20*log10(OUT_FLOAT);
        end

        %% put the outputs into the storage array
        if (freqSweep == 1)
            OUT_FIXED_sfdr(jj,kk) = findDR(OUT_FIXED(1:N),Fs,
               normalize);
```

```
        end
    end % end shifting frequency loop
end % end input frequency loop

%% create the x-axis labels for the following plots
a = 0:Fs/16:Fs/2; labels = []; ii = 1;
while (ii <= length(a))
    x = a(ii)/1e6;
    % trim to 3 decimal places
    x = x*10^3; x = floor(x); x = x/10^3;
    labels = [labels x]; ii=ii+1;
end

%% now plot the floating and fixed point sfdr's found
if (( plotSFDR == 1) && (freqSweep == 1) )
    figure; plot(freq,OUT_FIXED_sfdr,'-o');
    high = max(max(OUT_FIXED_sfdr))+10;
    axis manual; axis([min(freq),max(freq),0,high]);
    set(gca, 'XTick', 0:Fs/16:Fs/2);
    set(gca, 'XTickLabel', {labels(1:length(a))});
    xlabel('Input Frequency (MHz)'); ylabel('SFDR (dB)');
end
if ( (plotFixed == 1) && (freqSweep == 0) )
    figure; plot(f(1:N/2),OUT_FIXED(1:N/2));
    high = max(OUT_FIXED(1:N/2))+10;
    axis manual; axis([min(f(1:N/2)),max(f(1:N/2)),sigFloor,high]);
    set(gca, 'XTick', 0:Fs/16:Fs/2);
    set(gca, 'XTickLabel', {labels(1:length(a))});
    xlabel('Frequency (MHz)'); ylabel('Magnitude (dB)');
end
if ( (plotFloat == 1) && (freqSweep == 0) )
    figure; plot(f(1:N/2),OUT_FLOAT(1:N/2));
    high = max(OUT_FLOAT(1:N/2))+10;
    axis manual; axis([min(f(1:N/2)),max(f(1:N/2)),sigFloor,high]);
    set(gca, 'XTick', 0:Fs/16:Fs/2);
    set(gca, 'XTickLabel', {labels(1:length(a))});
    xlabel('Frequency (MHz)'); ylabel('Magnitude (dB)');
end

%% now figure out the min, max and avg SFDR
if (freqSweep == 1)
    %% remove the bottom and top 5 MHz from the scan for min and avg
    ss = round(5e6/(freqRes));
    tt = round((freq(end)-5e6)/freqRes);

    %% now tabulate the min, max and avg SFDR results
    maxSFDR = max(max(OUT_FIXED_sfdr));
    minSFDR = min(min(OUT_FIXED_sfdr(ss:tt,:)));
    avgSFDR = sum(sum(OUT_FIXED_sfdr(ss:tt,:)))/((length(freq)-(
       length(freq)-tt)-ss)*length(shiftFreq)-length(shiftFreq));

    %% now find out where the exact min and max occurred
    minSFDR_input=0; minSFDR_shift=0;
    maxSFDR_input=0; maxSFDR_shift=0;
    ii=1;
    while ii<=length(freq)
        jj=1;
        while jj<=length(shiftFreq)
            if (OUT_FIXED_sfdr(ii,jj) == maxSFDR)
                maxSFDR_input = freq(ii);
                maxSFDR_shift = shiftFreq(jj);
                ii=length(freq);
                jj=length(shiftFreq);
```

```
            end
            jj = jj+1;
        end
        ii=ii+1;
    end
    ii=1;
    while ii<=length(freq)
        jj=1;
        while jj<=length(shiftFreq)
            if (OUT_FIXED_sfdr(ii,jj) == minSFDR)
                minSFDR_input = freq(ii);
                minSFDR_shift = shiftFreq(jj);
                ii=length(freq);
                jj=length(shiftFreq);
            end
            jj = jj+1;
        end
        ii=ii+1;
    end

    %% now find the standard deviation
    stdDevSFDR = (std(OUT_FIXED_sfdr(ss:tt,:)));
    stdDevSFDR = sum(stdDevSFDR)/length(stdDevSFDR);

    %% now print the conditions for the plot
    fprintf(1,'Input Sweep Range:\n');
    fprintf(1,'  Starting Frequency   : %3.3f MHz\n',startFreq/1e6);
    fprintf(1,'  Ending Frequency     : %3.3f MHz\n',endFreq/1e6);
    fprintf(1,'  Frequency Resolution : %3.3f MHz\n',freqRes/1e6);
    fprintf(1,'  Number of Frequencies: %d\n',length(freq));
    fprintf(1,'\n');
    fprintf(1,'Shifting Frequency Range:\n');
    fprintf(1,'  Starting Frequency   : %3.3f MHz\n',startShiftFreq/1
       e6);
    fprintf(1,'  Ending Frequency     : %3.3f MHz\n',endShiftFreq/1e6
       );
    fprintf(1,'  Frequency Resolution : %3.3f MHz\n',shiftFreqRes/1e6
       );
    fprintf(1,'  Number of Frequencies: %d\n',length(shiftFreq));
    fprintf(1,'\n');
    fprintf(1,'SFDR Calculations\n');
    fprintf(1,'  Minimum SFDR : %3.3f dB @ %3.3f MHz input and %3.3f
       shift\n',minSFDR,minSFDR_input/1e6,minSFDR_shift/1e6);
    fprintf(1,'  Maximum SFDR : %3.3f dB @ %3.3f MHz input and %3.3f
       shift\n',maxSFDR,maxSFDR_input/1e6,maxSFDR_shift/1e6);
    fprintf(1,'  Average SFDR : %3.3f dB\n',avgSFDR);
    fprintf(1,'  Std Deviation: %3.3f dB\n',stdDevSFDR);
end
```

A.4 Matlab® Comparison Script

```
%%
close all;
clear all; clc;

%% setup the simulatoion type
type = 1; % 0 - behavioral
          % 1 - post-synthesis
          % 2 - post-par

%% setup parameters
N = 4096;
if (type == 0)
  Ts = 14.5e-9;
else
  Ts = 14.5e-9;
```

```
end;
Fs = 1/Ts;                    % set frequency based on period
fstep = Fs/N;                 % frequency step size
f = 0:fstep:Fs-fstep; % frequency range -Fs/2->Fs/2
dbFloor = -100;
power = 1;
plotSig = 1;

%% output file name
if (type == 0)
  file = 'sim_out_output.txt';
elseif (type == 1)
  file = 'sim_out_output_syn.txt';
else
  file = 'sim_out_output_par.txt';
end;

%% gather the output
output = load(file);

%% truncate the output and window it
output = output(1:N).*hanning(N);

%% perform N-pt FFT and normalize the output and put in dB
ff_out = abs(fft(output,N));
if (power == 1)
    ff_out = ff_out.^2; ff_out = 10*log10(ff_out/max(ff_out));
else
    ff_out = 20*log10(ff_out/max(ff_out));
end

%% design the xlabels for the plots
labels = [];
for ii=0:Fs/16:Fs/2
    a = floor(((ii/1e6)*1e3))/1e3;
   labels = [labels  a];
end

%% plot the output
if (plotSig == 1)
    figure; plot(f(1:N/2),ff_out(1:N/2)); %title('Behavioral
       Simulation');
    axis([min(f(1:N/2)) max(f(1:N/2)) dbFloor 10]);
    set(gca,'XTick',0:Fs/16:Fs/2);
    set(gca,'XTickLabel',labels);
    xlabel('Frequency (MHz)');
    ylabel('Magnitude (dB)');
end

sfdr = findDR(ff_out,Fs,1)
```

A.5 Dynamic Range Helper Function

```
function dr = findDR(FFT,Fs,norm)
%%
% This function finds the dynamic range of the given:
%
%  FFT - FFT data (in dB)
%  Fs  - sampling frequency used

%% figure out how many points there are
N = length(FFT)/2;

%% internally normalize the fft data
FFT = FFT-min(FFT);

%% if the data has not been normalized, then normalize it
if (norm == 1)
```

```
    FFT = FFT+-1*min(FFT);
end

%% now find the maximum signal of interest
freqBin = find(FFT(1:N) == max(FFT(1:N)));

if length(freqBin) > 1 freqBin = freqBin(1); end;

%% now start there and go forward and backwards to find a
%% trough, if it exists
leftTrough=0; rightTrough=0;
ii=freqBin-1; prev=max(FFT(freqBin-1:1:freqBin+1));
while ii>1
    %find point where the current FFT point is larger than the
       previous
    if (prev == 0)
        prev = FFT(ii);
    elseif (FFT(ii) > prev)
        leftTrough = ii;
        ii = 0;
    else
        prev = FFT(ii);
    end
    ii = ii-1;
end

ii=freqBin+1; prev=max(FFT(freqBin-1:1:freqBin+1));
while ii<=N
    if (prev == 0)
        prev = FFT(ii);
    elseif (FFT(ii) > prev)
        rightTrough = ii;
        ii = N+1;
    else
        prev = FFT(ii);
    end
    ii = ii+1;
end

%% now find the largest spur
if ( (leftTrough == 0) && (rightTrough == 0) )
    dr = 0;
elseif ( (leftTrough ~= 0) && (rightTrough == 0) )
    dr = max(FFT(1:N))-max(FFT(1:leftTrough));
elseif ( (leftTrough == 0) && (rightTrough ~= 0) )
    dr = max(FFT(1:N))-max(FFT(rightTrough:N));
else
    dr = max(FFT(1:N))-max(max(FFT(1:leftTrough)),max(FFT(rightTrough
       :N)));
end
end
```

The top-level VHDL source file for the DSSM is found in this appendix. This top-level file utilizes several user-changeable entries as generics that will alter the behavior of the DSSM. Other changes such as the length of the Hilbert filter, the bit width of the Hilbert filter coefficients, the size of the ROM as well as the ROM coefficient bit width must be changed either manually or through the use of a VHDL generator.

B.1 DSSM VHDL File

```
library IEEE,work;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;
  use IEEE.math_real.all;
  use work.all;

entity DSSM is
  generic (
     NF  : integer := 8;     -- number of bits for the Fc scaling
                             -- log2(number of ROM samples)
     NC  : integer := 16;    -- number of bits for sine and cosine
     NCO : integer := 16;    -- number of bits of output cosine
     NI  : integer := 16 ); -- number of bits coming from ADC
  port (
     ADC_IN : in std_logic_vector(NI-1 downto 0);
     CLK_IN : in std_logic;
     MUX_S : in std_logic;
     RST_IN : in std_logic;
     ADD_SUB_S : in std_logic;
     ROM_SEL : in std_logic;
     Fc_IN : in std_logic_vector(NF-1 downto 0);
     MUX_OUT : out std_logic_vector(NI-1 downto 0);
     HIL_Q_OUT : out std_logic_vector(NI-1 downto 0);
     ROM_COS_OUT : out std_logic_vector(NCO-1 downto 0);
     DRFM_OUT : out std_logic_vector(NI-1 downto 0) );
end DSSM;

architecture Behav of DSSM is

  component HILBERT_NT101_NC14
    generic (
      NI : integer := 16;
      NE : integer := 6 );
    port (
      SI : in std_logic_vector(NI-1 downto 0);
      CLK : in std_logic;
      SD : out std_logic_vector(NI-1 downto 0);
      SO : out std_logic_vector(NI-1 downto 0);
      regOut : out std_logic_vector(NI-1 downto 0) );
  end component;

  component HILBERT
    port (
      SI : in std_logic_vector(NI-1 downto 0);
      CLK : in std_logic;
      SD : out std_logic_vector(NI-1 downto 0);
      SO : out std_logic_vector(NI-1 downto 0);
      regOut : out std_logic_vector(NI-1 downto 0) );
```

```
  end component;

  component rom_wrapper
    generic (
      NC : integer := 16;
      NF : integer := 8 );
    port (
      CLK : in std_logic;
      RST : in std_logic;
      SEL : in std_logic;
      addr : in std_logic_vector(NF+1 downto 0);
      sin : out std_logic_vector(NC-1 downto 0);
      cos : out std_logic_vector(NC-1 downto 0) );
  end component;

  component mixer
    generic (
      NI : integer := 16;
      NC : integer := 16;
      NF : integer := 8 );
    port (
      clk : in std_logic;
      RST : in std_logic;
      ADD_SUB_S : in std_logic;
      Fc : in std_logic_vector(NF-1 downto 0);
      Isig_in : in std_logic_vector(NI-1 downto 0);
      Qsig_in : in std_logic_vector(NI-1 downto 0);
      Cos_in : in std_logic_vector(NC-1 downto 0);
      Sin_in : in std_logic_vector(NC-1 downto 0);
      addr_out : out std_logic_vector(NF+1 downto 0);
      Mix_out : out std_logic_vector(NI-1 downto 0) );
  end component;

  component gen_mux
    generic (
      NI : integer := 16 );
    port (
      A : in std_logic_vector(NI-1 downto 0);
      B : in std_logic_vector(NI-1 downto 0);
      S : in std_logic;
      Z : out std_logic_vector(NI-1 downto 0) );
  end component;

  signal GND : std_logic;

  signal shiftRegOut : std_logic_vector(NI-1 downto 0);
  signal I,Q : std_logic_vector(NI-1 downto 0);
  signal mixed : std_logic_vector(NI-1 downto 0);
  signal cos,sin : std_logic_vector(NC-1 downto 0);
  signal addr : std_logic_vector(NF+1 downto 0);
  signal muxOut : std_logic_vector(NI-1 downto 0);

begin

  -- begin instantiating the components

  -- mux the input from the ADC with the shift register output
  im1 : gen_mux
      generic map (NI => NI)
      port map (A=>ADC_IN,B=>shiftRegOut,S=>MUX_S,Z=>muxOut);

  -- hilbert transform from sig to I and Q
  hil : HILBERT_NT101_NC14
      generic map (NI=>NI, NE=>6)
      port map (SI=>muxOut, CLK=>CLK_IN, SD=>I, SO=>Q, regOut=>
          shiftRegOut);
```

```
  -- digital mixer
  mx1 : mixer
      generic map (NI=>NI, NC=>NC, NF=>NF)
      port map (clk=>CLK_IN, ISig_in=>I, QSig_in=>Q, Fc=>Fc_IN,
                RST=>RST_IN, Cos_in=>cos, Sin_in=>sin, addr_out=>addr
                  ,
                Mix_out=>Mixed, ADD_SUB_S=>ADD_SUB_S);

  -- wrapper to interface with ROM
  -- given the address, will query ROM appropriately and get both
  -- the sine and cosine for that particular address
  -- rom_wrapper contains 2 256x16 sequentially addressed ROMs
  rw1 : rom_wrapper
      generic map (NF=>NF, NC=>NC)
      port map (CLK=>CLK_IN,addr=>addr,RST=>RST_IN,
                sin=>sin, cos=>cos, SEL=>ROM_SEL);

  -- now send outputs to the OUTSIDE WORLD
  MUX_out <= muxOut;
  HIL_Q_OUT <= Q;
  ROM_COS_OUT <= cos(NC-1 downto NC-NCO);
  DRFM_OUT <= Mixed;

end Behav;
```

B.2 VHDL Design Testbench

```
-- run_sim
library ieee, work;
  use std.textio.all;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_arith.all;
  use ieee.std_logic_signed.all;
  use ieee.math_real.all;
  use work.all;

entity TB_DSSM is
  generic (
    NC      : integer := 16;
    NI      : integer := 16;
    NF      : integer := 8;
    FC_int : integer := 67; -- 4.362 MHz
    N_ITS   : integer := 10001;  -- number of test sets
    CP      : time    := 15.0 ns ;  -- 66.667 MHz clock period
    CPR     : real    := 15.0e-09 ; -- 66.667 MHz clock period
    f1      : real    :=  10000.0e03 ; -- 10 MHz
    f2      : real    :=  10000.0e03 ; -- 10 MHz
    P1      : real    := 0.0 ;      -- [dB] power levels
    P2      : real    := 0.0        -- [dB] power levels
  );
end ;

architecture TB of TB_DSSM is
  file OUT_FILE: text is out "sim_out.txt";
  file OUT_FILE1: text is out "sim_out_all.txt";
  file OUT_FILE2: text is out "sim_out_input.txt";
  file OUT_FILE3: text is out "sim_out_muxout.txt";
  file OUT_FILE4: text is out "sim_out_Q.txt";
  file OUT_FILE5: text is out "sim_out_cos.txt";
  file OUT_FILE6: text is out "sim_out_output.txt";
  file OUT_FILE7: text is out "sim_out_sin.txt";

  component DSSM
    generic (
        NF : integer := 8;
        NI : integer := 16;
        NC : integer := 16 );
    port (
        ADC_IN : in std_logic_vector(NI-1 downto 0);
        CLK_IN : in std_logic;
```

```
        MUX_S : in std_logic;
        RST_IN : in std_logic;
        ADD_SUB_S : in std_logic;
        ROM_SEL : in std_logic;
        Fc_IN : in std_logic_vector(NF-1 downto 0);
        MUX_OUT : out std_logic_vector(NI-1 downto 0);
        HIL_Q_OUT : out std_logic_vector(NI-1 downto 0);
        ROM_COS_OUT : out std_logic_vector(NC-1 downto 0);
        DRFM_OUT : out std_logic_vector(NI-1 downto 0) );
  end component;

  type TYPE_INPUT_ARRAY is array(0 to N_ITS) of std_logic_vector(NI-1
      downto 0);
  type TYPE_REAL_ARRAY is array(0 to N_ITS) of real;
  signal CLK,CLK_ROM : std_logic;
  signal RESET : std_logic;
  signal INDEX : integer := 0;
  signal ADC : std_logic_vector(NI-1 downto 0);
  signal INPUT_V: TYPE_INPUT_ARRAY;
  signal FC : std_logic_vector(NF-1 downto 0);
  signal muxSelect : std_logic;
  signal muxOUT,drfmOUT: std_logic_vector(NI-1 downto 0);
  signal add_sub : std_logic;
  signal hilQOUT: std_logic_vector(NI-1 downto 0);
  signal romCosOUT: std_logic_vector(NC-1 downto 0);
  signal romSelect : std_logic;

begin
  COMP_DSSM: DSSM
    generic map (NF=>NF,NC=>NC,NI=>NI)
    port map (CLK_IN=>CLK, ADC_IN=>ADC, RST_IN=>RESET,
              MUX_S=>muxSelect, Fc_IN=>FC, MUX_OUT=>muxOUT,
              HIL_Q_OUT=>hilQOUT, ROM_COS_OUT=>romCosOUT,
              DRFM_OUT=>drfmOUT,ADD_SUB_S=>add_sub,ROM_SEL=>romSelect
                 );

  Fc <= conv_std_logic_vector(FC_int,NF);
  muxSelect <= '0';
  romSelect <= '1';
  add_sub <= '1';
  -- functionality of add_sub
  -- if add_sub = '1' and if F_in < F_c then F_out = F_in+F_c
  --                      if F_in > F_c then F_out = F_in-F_c
  -- if add_sub = '0' and if F_in < F_c then F_out = F_in-F_c
  --                      if F_in > F_c then F_out = F_in+F_c

  process (CLK)
   variable tmp : integer := 0;
  begin
    if CLK'event and CLK = '1' then
      if tmp = 0 then
        RESET <= '1';
        tmp := tmp+1;
      else
        RESET <= '0';
      end if;
    end if;
  end process;

  CLK_PROC: process
  begin
    wait for CP/2;
    CLK <= '0';
    wait for CP/2;
    CLK <= '1';
  end process;
```

```
process(CLK)
begin
  --if CLK'event and CLK='1'then
  -- note we have to delay the input relative to clk arrival for
     setup time
  if CLK'event and CLK='0'then
    ADC <= INPUT_V(INDEX);
    if INDEX = N_ITS-1 then
      INDEX <= 0;
    else
      INDEX <= INDEX + 1;
    end if;
  end if;
end process;

FFT_PROC: process
  variable W1,W2,A1,A2,Amax,P_dym,fsample:real:=1.0; -- amplitudes
  variable S_REAL,VAL_COMP,VAL_DIV : real;
  variable INPUT_R: TYPE_REAL_ARRAY;
  variable OUT_LINE: line;
  variable J,K: integer;
  variable S_VECTOR : TYPE_INPUT_ARRAY;
begin
  fsample := 1.0/real(CPR);
  ----------------------------------------
  --vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
  -- generate input data ---------------
  P_dym := P1 - P2;
  A2 := Amax / (1.0 + exp(log(10.0)*(P_dym)/10.0));
  A1 := Amax-A2; -- Amax - A2
  W1 := 2.0 * 3.1415927 * real(f1);  -- sets W
  W2 := 2.0 * 3.1415927 * real(f2);  -- sets W
  K:= 0;WH_LOOPK0: while K < N_ITS-1 loop
    --S_REAL := A1*sin(W1*REAL(K)*CPR);                    -- one
       tone
    S_REAL := A1*sin(W1*REAL(K)*CPR)+A2*sin(W2*REAL(K)*CPR);-- two
       tones

    -- n bit ADC
    VAL_COMP := 0.0; VAL_DIV := 0.0;
    if S_REAL < VAL_COMP then S_VECTOR(K)(NI-1):='1';VAL_DIV:=-0.5;
       VAL_COMP:=-0.5;
    else S_VECTOR(K)(NI-1):='0';VAL_DIV:=0.5;VAL_COMP:=0.5;end if;
    for IIII in 1 to NI-1 loop
      if S_REAL < VAL_COMP then
        S_VECTOR(K)(NI-1-IIII):='0';
        VAL_DIV:=VAL_DIV/2.0;
        VAL_COMP:=VAL_COMP-abs(VAL_DIV);
      else  -- S_REAL >= VAL_COMP
        S_VECTOR(K)(NI-1-IIII):='1';
        VAL_DIV:=VAL_DIV/2.0;
        VAL_COMP:=VAL_COMP+abs(VAL_DIV);
      end if;
    end loop;

    INPUT_V(K) <= S_VECTOR(K);
    INPUT_R(K) := S_REAL;
    K := K + 1;
  end loop WH_LOOPK0;
  --^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  K:= 0;WH_LOOPK1: while K < N_ITS-1 loop
    if K >= N_ITS/2 then
      write(OUT_LINE,INDEX);
      write(OUT_LINE,string'("   "));
      write(OUT_LINE,INPUT_R(K));
      write(OUT_LINE,string'("   "));
      J:= NI-1;WH_LOOPJ2: while J > -1 loop
```

```
          write(OUT_LINE,conv_integer(INPUT_V(K)(J)));
          J := J - 1;
        end loop WH_LOOPJ2;

        write(OUT_LINE,conv_integer(INPUT_V(K)));
        writeline(OUT_FILE2, OUT_LINE);
        write(OUT_LINE,conv_integer(muxOUT));
        writeline(OUT_FILE3, OUT_LINE);
        write(OUT_LINE,conv_integer(hilQOUT));
        writeline(OUT_FILE4, OUT_LINE);
        write(OUT_LINE,conv_integer(romCosOUT));
        writeline(OUT_FILE5, OUT_LINE);
        write(OUT_LINE,conv_integer(drfmOUT));
        writeline(OUT_FILE6, OUT_LINE);

      end if;
      wait until CLK'event and CLK = '1';
      wait until CLK'event and CLK = '0';
      K := K + 1;
    end loop WH_LOOPK1;
    ----------------------------------------
    ----------------------------------------
    assert (false) report "done" severity FAILURE;
  end process;

end ;

configuration CFG_TB of TB_DSSM is
  for TB
  end for;
end ;
```

Bibliography

1. Axtell H. S. *Development of a Parameterizable, Synthesizable Real-Time Digital Single Sideband Modulator for Hardware Implementation*. MS Thesis, Wright State University, MSEgr, Dayton, OH, June 2010.

2. Barnaby H. "Total-ionizing-dose effects in modern CMOS technologies," *Nuclear Science, IEEE Transactions on*, *53*:3103 –3121 (December 2006).

3. Benedetto J., Eaton P., Mavis D., Gadlage M., and Turflinger T. "Digital Single Event Transient Trends with Technology Node Scaling," *Nuclear Science, IEEE Transactions on*, 3462 –3465 (December 2006).

4. Braga L. H. C., Domingues S., Rocha M. F., Sá L. B., Campos F., Santos F. V., Mesquita A. C., Silva M. V., and Swart J. W. "Layout techniques for radiation hardening of standard CMOS active pixel sensors," *SBCCI '07: Proceedings of the 20th annual conference on Integrated circuits and systems design*, 257 –262 (September 2007).

5. Chavez R. M., Rax B. G., Scheick L. Z., and Johnston A. H. "Total ionizing dose effects in bipolar and BiCMOS devices," *Radiation Effects Data Workshop 2005, IEEE*, 144 –148 (July 2005).

6. Claeys C. and Simoen E. *Radiation Effects in Advanced Semiconductor Materials and Devices*. Spring, October 2002.

7. Colinge J.-P. *Silicon-on-Insulator Technology: Materials to VLSI*. Springer, September 1997.

8. Engelberg S. *Digital Signal Processing, An Experimental Approach*. Springer, February 2008.

9. Foltz T., Cook G., and Meer D. "A digital single sideband modulator for a digital radio frequency memory," *Aerospace and Electronics Conference, 1989. NAECON 1989., Proceedings of the IEEE 1989 National*, *2*:926 –932 (May 1989).

10. Ghosh M., Chimakurthy L., Dai F., and Jaeger R. "A novel DDS architecture using nonlinear ROM addressing with improved compression ratio and quantisation noise," *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, *2*:705 –708 (May 2004).

11. Guoying S., Yunjie L., and Meiguo G. "An Improved DRFM System Based on Digital Channelized Receiver," *Image and Signal Processing, 2009. CISP '09. 2nd International Congress on*, 1 –5 (October 2009).

12. Hamming R. W. "Error Detecting and Error Correcting Codes," *Bell System Technical Journal*, *29*:147 –160 (1950).

13. Haykin S. and Moher M. *Introduction to Analog and Digital Communications* (2nd Edition). Wiley, 2007.

14. Haykin S. "Different forms of the Hilbert transform as applied to digital filters and sequences," *Electrical Engineers, Proceedings of the Institution of*, *121*(7):561 –567 (July 1974).

15. Hilbert D., Hallett M., and Majer U. *David Hilbert's Lectures on the Foundations of Geometry, 1891-1902*. Springer, July 2004.

16. Hopkins T. A. *An Automated Approach to a 90-nm CMOS DRFM DSSM Circuit Design*. MS Thesis, Wright State University, MSEgr, Dayton, OH, June 2010.

17. Huhtinen F. D., Faccio F., Detcheverry C., and Huhtinen M. "First evaluation of the Single Event Upset (SEU) risk for electronics in the CMS experiment," *The Compact Muon Solenoid (CMS) Experiment Note* (1998).

18. Imec B. D., Bogaerts J., and Dierickx B., "Total Dose Effects on CMOS Active Pixel Sensors," 1998.

19. Koopman P. and Chakravarty T. "Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks," *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, 145 (July 2004).

20. Lewis G. K., Bahl I. J., and Griffin E. L. "GaAs MMIC For Digital Radio Frequency Memory (DRFM) Subsystems," *1987 Microwave and Millimeter-Wave Monolithic Circuits Symposium, IEEE*, 53 –56 (June 1987).

21. Ma T. and Dressendorfer P. V. *Ionizing Radiation Effects in MOS Devices and Circuits*. Wiley, April 1989.

22. Mavis D. G. and Eaton P. H., "Temporally Redundant Latch For Preventing Single Event Disruptions in Sequential Integrated Circuits." *US Patent 6,127,864*, assigned to Micro-RDC. Filed on August 19, 1998. Application No. 09/136,872. Approved on October 3, 2000.

23. Mohr K. C., Clark L. T., and Holbert K. E. "A 130-nm RHBD SRAM With High Speed SET and Area Effecient TID Mitigation," *Nuclear Science, IEEE Transactions on*, 2092 –2099 (December 2007).

24. Mohr K. C., Clark L. T., Holbert K. E., Yao X., Knudsen J., and Shah H. "Optimizing Radiation Hard by Design SRAM Cells," *Nuclear Science, IEEE Transactions on*, 2028 –2036 (December 2007).

25. Moon T. K. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, June 2005.

26. Nyquist H. "Certain topics in telegraph transmission theory," *Bell Technical Journal*, *3*:617 –644 (April 1924).

27. Oppenheim A. V., Schafer R. W., and Buck J. R. *Discrete-Time Signal Processing* (2nd Edition). Prentice-Hall, January 1999.

28. SanGregory S. L. and Mehalic M. A. "A 2K X 8-bit single-chip digital radio frequency memory," *ASIC Conference and Exhibit, 1993. Proceedings., Sixth Annual IEEE International*, 247 –249 (27 Sep - 1 Oct 1993).

29. Shoga M. and Binder D. "Theory of Single Event Latchup in Complementary Metal-Oxide Semiconductor Integrated Circuits," *Nuclear Science, IEEE Transactions on*, *33*(6):1714 –1717 (December 1986).

30. Vahid F. *Digital Design*. Wiley, July 2006.

31. ViASIC , "ViaMask Datasheet." Internet: `http://viasic.wenderhost.com/wp-content/uploads/datasheets/ViaMask_datasheet.rev6.29.09-2.pdf`, June 2009.

32. Wirth G., Vieira M., Neto E., and Kastensmidt F. "Single Event Transients in Combinatorial Circuits," *Integrated Circuits and Systems Design, 18th Symposium on*, 121 –126 (September 2005).

33. Yi L., Tuan Y., Ningmei Y., and Yong G. "The Application of a Novel Direct Digital Synthesizer for the IP Code Design of All Digital Three Phase SPWM Generator," *Power Electronics and Motion Control Conference, 2004. IPEMC 2004. The 4th International*, *2*:730 –733 (August 2004).

34. Zongbo W., Meiguo G., Yunjie L., and Haiqing J. "Design and application of DRFM system based on digital channelized receiver," *Radar, 2008 International Conference on*, 375 –378 (September 2008).