A HIGH PERFORMANCE BASEBAND INSTRUMENT

Institution: UNIVERSITY OF CALIFORNIA, DAVIS

Author: Jeremy W. Webb- jwwebb@ucdavis.edu

Mentor: Bevan Baas - bbaas@ucdavis.edu

Project Date: March 2011 (Duration: 2 years)

Project URL: http://msee.jwebb-design.com/measbd/

Testing complex digital signal processors (DSPs), such as the *Asynchronous Array of Simple Processors* (AsAP), requires a development platform with sufficient signal bandwidth and system performance to provide and consume data to and from the DSP. Without a development platform, verification of DSPs would be limited to monitoring test output signals for an indication of performance and successful operation. This document describes the design of a General Purpose Instrument which will simplify the testing and characterization of the AsAP processor when performing real world DSP tasks. The General Purpose Instrument is a flexible platform capable of targeting a wide variety of applications, such as signal generation and signal analysis.

Project Goals

- Signal generation and analysis frequency range from DC to 110 MHz.
- A 334-processor platform with real-time I/O.
- Flexible waveform generation, loading, and capture.
- A 50 Ω input and output impedance to simplify interconnection with standard test and measurement equipment.

Project Contributions

- General Purpose Instrument system architecture design.
- Measurement board design, layout, and characterization.
- Data Path field programmable gate array (FPGA) system architecture design and SystemVerilog HDL development.
- Control FPGA embedded soft-core processor software architecture design.

Conclusion The General Purpose Instrument, shown in Figures 1 and 2, is a successful development platform that can be used for a wide variety of AsAP DSP software prototyping. This platform can be used to target applications from software defined radios to cognitive radio. The signal bandwidth of the front end designs exceeded my initial design goals of a frequency range from DC to 110 MHz by 15 MHz.

The General Purpose Instrument is a flexible platform capable of targeting a wide variety of applications, such as signal generation and signal analysis, and includes: a 12-bit, 500 $\mathrm{MS/s}$ analog-to-digital converter (ADC) input, a dual-channel, 16-bit, 1 $\mathrm{GS/s}$ digital-to-analog converter (DAC) output, a Xilinx Virtex-5 SX50T data path field programmable gate array (FPGA), a Xilinx Spartan-3A XC3S1400A control FPGA, a 36-Mbit QDR-II static random access memory (SRAM), a 2 GB DDR2 synchronous dynamic random access memory (SDRAM), a 512 Mbit DDR SDRAM, and two 2 GB microSD cards. The signal analyzer input operates with a -3 dB frequency of 134 MHz, and has a noise floor of -98 dBm. The signal source output operates with a -3 dB frequency of 138 MHz, a spurious-free dynamic range (SFDR) of 68.49 dBc at a power level of -6 dBFS, and a signal-to-noise ratio (SNR) of 101.02 dBc.

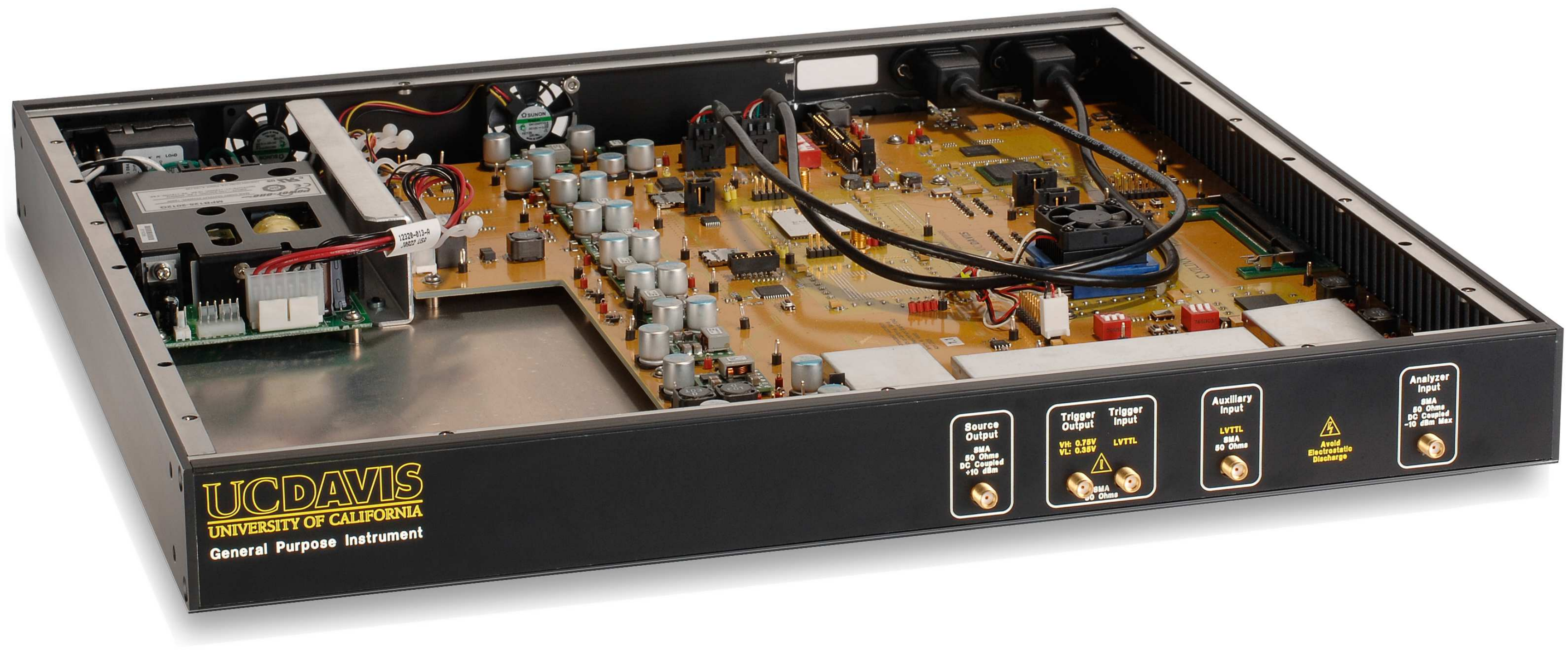
UCDAVIS
UNIVERSITY OF CALIFORNIA
General Purpose Instrument
Source Output SMA 50 Ohms DC Coupled +10 dBm
Trigger Output Vh: 0.75V VL: 0.35V
Trigger Input LVTTL
SMA 50 Ohms
Auxiliary Input LVTTL SMA 50 Ohms
Avoid Electrostatic Discharge
Analyzer Input SMA 50 Ohms DC Coupled -10 dBm Max

Figure 1: General Purpose Instrument ISO View with open top

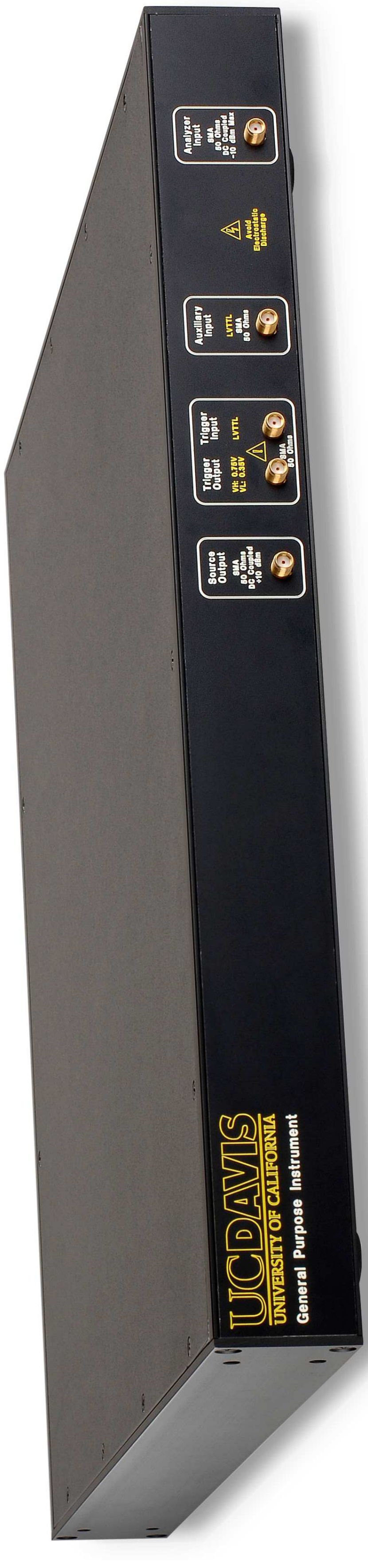

Figure 2: General Purpose Instrument ISO View

A HIGH PERFORMANCE BASEBAND INSTRUMENT

By

JEREMY WILLIAM WEBB
B.S. (University of California, Davis) December, 2000

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Bevan M. Baas, Chair

Gary E. Ford

Andre Knoesen

Committee in charge
2011

Abstract

Testing complex digital signal processors (DSPs) requires a development platform with sufficient signal bandwidth and system performance to fully exercise the DSP. Without a development platform, verification of DSPs would be limited to monitoring test output signals for an indication of performance and successful operation. In addition, a development platform with high-speed analog input and output interfaces to the DSP system allows it to be used directly in many sophisticated real-time applications. Presented here is a 334-processor development platform for testing of the *Asynchronous Array of Simple Processors* (AsAP). This platform, known as the General Purpose Instrument, is capable of generating and analyzing baseband signals. The General Purpose Instrument simplifies the testing and characterization of the AsAP processor when performing real world DSP tasks. The General Purpose Instrument is a flexible platform capable of targeting a wide variety of applications, such as signal generation and signal analysis, and includes: a 12-bit, 500 MS/s analog-to-digital converter (ADC) input, a dual-channel, 16-bit, 1 GS/s digital-to-analog converter (DAC) output, a Xilinx Virtex-5 SX50T data path field programmable gate array (FPGA), a Xilinx Spartan-3A XC3S1400A control FPGA, a 36-Mbit QDR-II static random access memory (SRAM), a 2 GB DDR2 synchronous dynamic random access memory (SDRAM), a 512 Mbit DDR SDRAM, and two 2 GB microSD cards. The signal analyzer input operates with a −3 dB frequency of 134 MHz, and has a noise floor of −98 dBm. The signal source output operates with a −3 dB frequency of 138 MHz, a spurious-free dynamic range (SFDR) of 68.49 dBc at a power level of −6 dBFS, and a signal-to-noise ratio (SNR) of 101.02 dBc.

Acknowledgments

First, I would like to thank Professor Baas for his guidance and support throughout my years at UC Davis. I would also like to thank Professor Ford and Professor Knoesen for their time in reviewing my thesis.

I'd like to thank everyone that contributed to the design of the measurement board. Application engineers at Texas Instruments, thanks for your help with simulating my signal source and signal analyzer front-end design. Gary Dooley, thanks for teaching me about spectrum analyzer digital IF design. Sam Walker, thank you for your continued support and guidance during the schematic design and PC board layout. Harrell Huckeba, thank you for your help with translating the single-ended anti-alias filter topology to a differential reconstruction filter topology. Robert Uang, thanks for your assistance with the PLL loop-filter design. Patty Walz-Hunter and Jason Thom, thank you for your help with the layout of the measurement board. And thanks to all the members of the VLSI Computation Laboratory.

I'm grateful for the financial support offered by my current and previous employers, including: Centellax, Inc., Agilent Technologies, Inc., Thomson Grass Valley, and Folsom Research.

Finally, I'd like to thank my family. To my parents, thank you for your support and encouragement throughout this time. To my wife, Julie, thank you for putting up with me for the past eight years. It's been a long journey, but it was worth it and I could not have done it without your love and support.

This work was supported in part by the National Science Foundation under Grant No. 0430090 and CAREER Award 0546907 and the Semiconductor Research Corporation under Grant No. 1598.001 and 1659.001.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Testing complex digital signal processors (DSPs), such as the *Asynchronous Array of Simple Processors* (AsAP), requires a development platform with sufficient signal bandwidth and system performance to provide and consume data to and from the DSP [1–4]. Without a development platform, verification of DSPs would be limited to monitoring test output signals for an indication of performance and successful operation. This thesis describes the design of a General Purpose Instrument which will simplify the testing and characterization of the AsAP processor when performing real world DSP tasks. The General Purpose Instrument is a flexible platform capable of targeting a wide variety of applications, such as signal generation and signal analysis.

1.1 Project Goals

- Signal generation and analysis frequency range from DC to 110 MHz.
- A 334-processor platform with real-time I/O.
- Flexible waveform generation, loading, and capture.
- A 50 Ω input and output impedance to simplify interconnection with standard test and measurement equipment.

1.2 Project Contributions

- General Purpose Instrument system architecture design.
- Measurement board design, layout, and characterization.

- Data Path field programmable gate array (FPGA) system architecture design and SystemVerilog HDL development.
- Control FPGA embedded soft-core processor software architecture design.

1.3 Organization

The remainder of this thesis is divided as follows: Chapter 2 provides an overview of the signal source design, verification, and the achieved performance. Chapter 3 discusses the signal analyzer design. Chapter 4 describes the design and turn-on process of the measurement board. Chapter 5 concludes with possibilities for future work. Supporting chapters are included in the appendix.

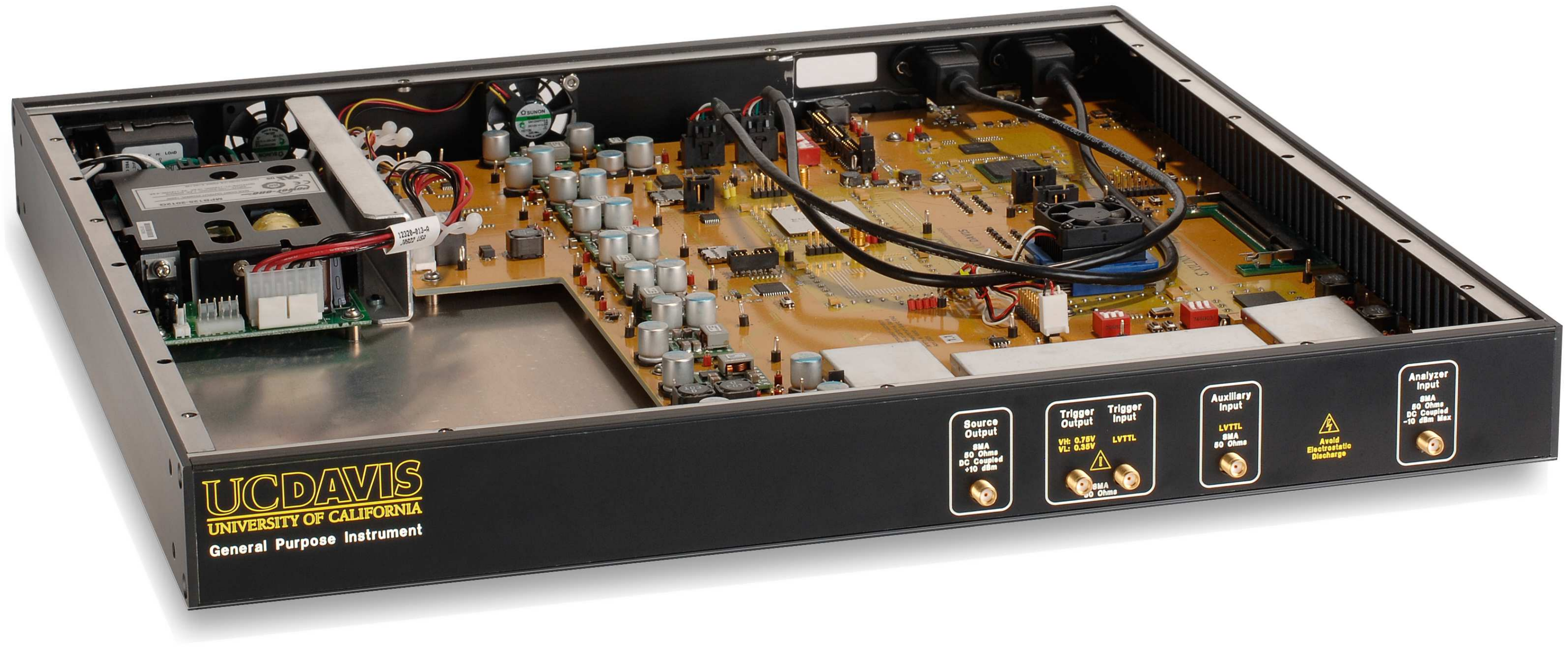

Figure 1.1: General Purpose Instrument ISO View with open top

Chapter 2

Signal Source

A signal source is a common component of a measurement setup, and is used to stimulate a device under test (DUT) with a continuous wave (CW) signal of known frequency and amplitude. In addition to CW waveforms, the following waveforms are also useful when measuring the performance of a DUT including:

- Triangle Waveform
- Square Waveform
- Arbitrary Waveform
- OFDM symbols
- CDMA symbols

The AsAP processor is well-suited for all types of signal generation applications, from arbitrary waveform generation to communication signals. The General Purpose Instrument provides a high-speed digital-to-analog converter (DAC) along with a field programmable gate array (FPGA) to demonstrate the signal generation capabilities of the two on-board AsAP DSP processors.

The requirements used for the signal source design are covered in Section 2.1. Section 2.2 describes the design of the signal source and each component in the signal path. The methods used to verify the performance of the signal source are describe in Section 2.3. And finally the performance of the signal source is summarized in Section 2.4.

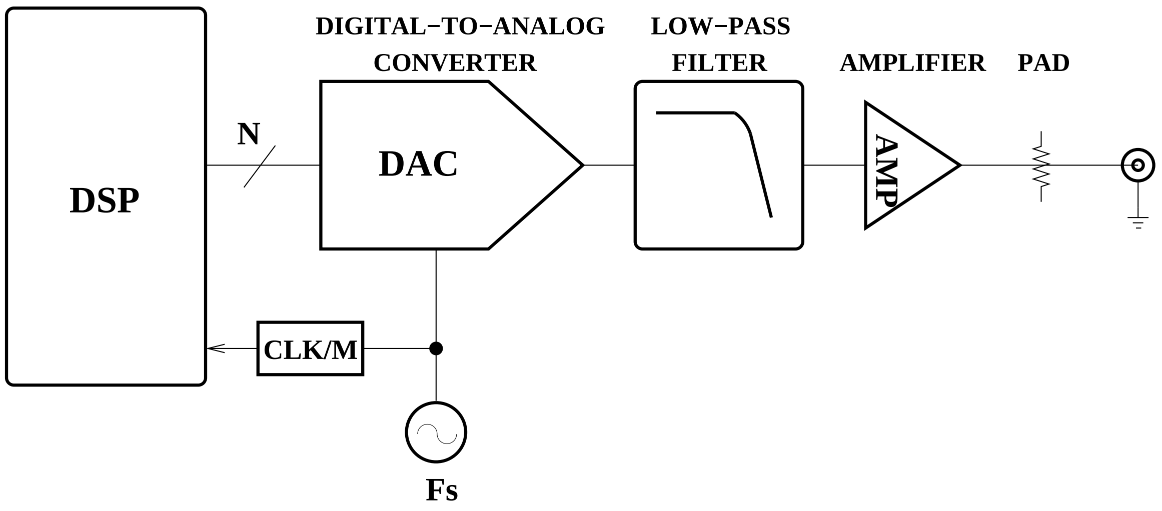

Figure 2.1: Common Signal Source Block Diagram

Figure 2.1 shows the basic block diagram of a signal source.

2.1 Requirements

A signal source can have an impact on the measurement results of a DUT. When performing measurements on a DUT, it is important to understand the capabilities of the signal source. The following parameters are typically specified for signal sources by test and measurement equipment manufacturers, and were addressed by the General Purpose Instruments signal source:

- Bandwidth
- Dynamic Range
- Distortion
- Accuracy

The baseband signal source output is intended to generate waveforms in the frequency range of DC to 120 MHz. Given the wide range of applications in this frequency range, and the desire to transmit data at several frequencies simultaneously, a wide bandwidth filter is required. The design complexity of the anti-image or reconstruction filter will be determined by the sample rate of the DAC. The goal is to provide the maximum amount of signal bandwidth (B) without violating the Nyquist sampling theorem. As the filter's cut-off frequency approaches the Nyquist frequency ($\frac{F_s}{2}$), the cost and complexity will increase. One possible solution is to oversample the DAC by a factor of two and set the filter's cut-off frequency to $\frac{F_s}{4}$. The benefit of this type of architecture is that common and inexpensive inductors and capacitors can be used along with a filter topology that is simple to design and debug.

Some signal generation applications may require signals that are closely spaced in the frequency domain. As such, these applications require excellent dynamic range performance. Dynamic range is affected by several factors including: noise present in the signal path, filter cut-off frequency, and amplifier performance. The measurement board of the General Purpose Instrument employs board level shields to enclose the signal source circuitry. The board level shields help minimize the effects of unwanted signals such as power supply switching frequencies and harmonic frequencies of the various clock sources on the board. The use of an oversampled system for the signal source design should also improve the dynamic range by further attenuating alias frequencies of the sampled system.

2.2 Design

The General Purpose Instrument signal source was designed to operate in the first Nyquist zone with a signal bandwidth of 120 MHz. Figure 2.2 shows a high-level block diagram of the signal source.

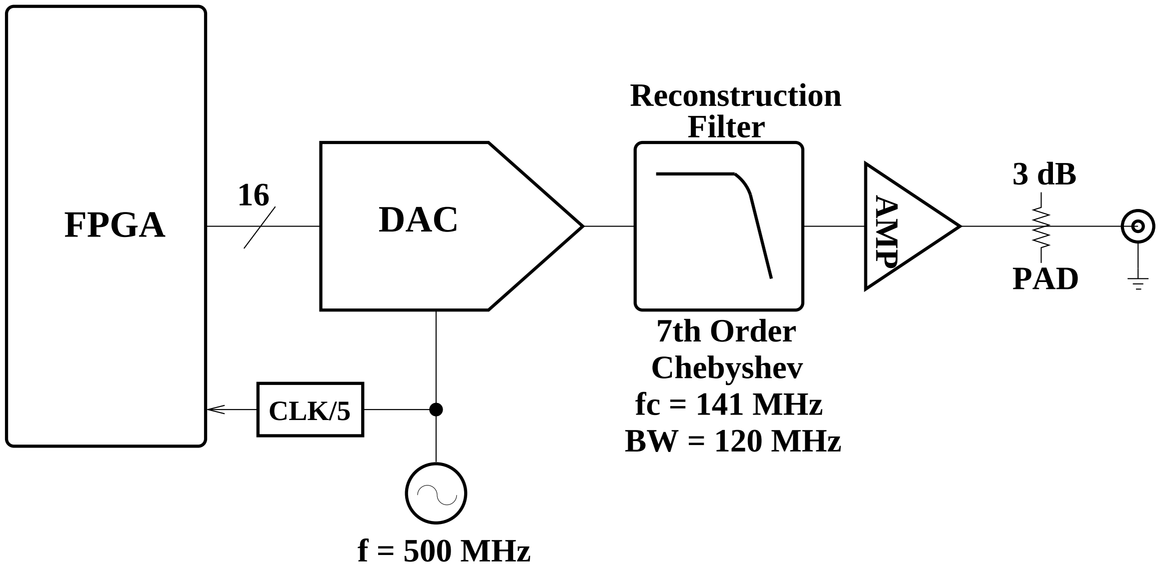

Figure 2.2: High-Level Signal Source Block Diagram

The main building blocks of the signal source are:

- Digital signal processing, implemented in an FPGA
- High-speed digital-to-analog converter
- Clock generation and distribution
- Passive low-pass reconstruction filter
- High-speed amplifier output stage

Each of the building blocks listed above is described in Subsections 2.2.1 to 2.2.6.

2.2.1 Digital Signal Processing

The digital signal processing sub-system of the signal source is made up of a high-performance field programmable gate array and two AsAP digital signal processors. The DSP sub-system is responsible for generating and transmitting digital signals to the high-speed DAC. The signal source can generate or playback signals from several sources including:

- 32-Mbit QDR-II SRAM
- 2-Mbit Block RAM
- 128-bit Static Register
- AsAP DSPs
- FPGA logic

A direct digital synthesizer (DDS) is an example of a signal source which could be implemented in FPGA logic. The current implementation of the signal source design supports only playback of signals from QDR-II SRAM, Block RAM, or a 128-bit register. Figure 2.3 shows the major components of the DSP sub-system, and Figure 2.4 shows a detailed view of the FPGA data path.

Several key design areas were addressed during the development of the DSP sub-system including:

- Waveform storage and playback
- Waveform scale, invert, and offset
- High-speed DAC interface
- Multiple clock domains

Each of the design areas listed above is described in Subsections 2.2.1.1 to 2.2.1.2.

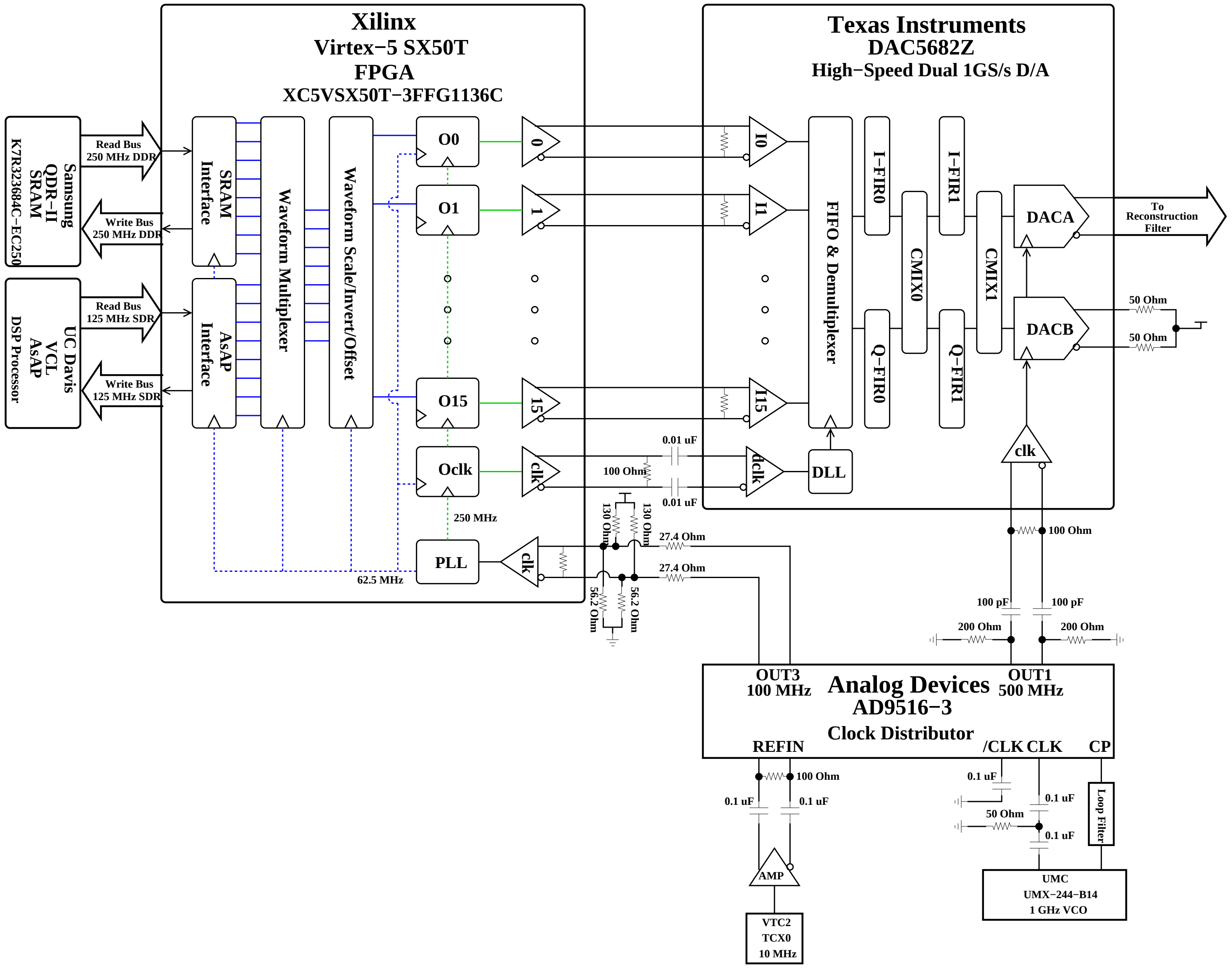

Figure 2.3: Baseband signal source DSP block diagram

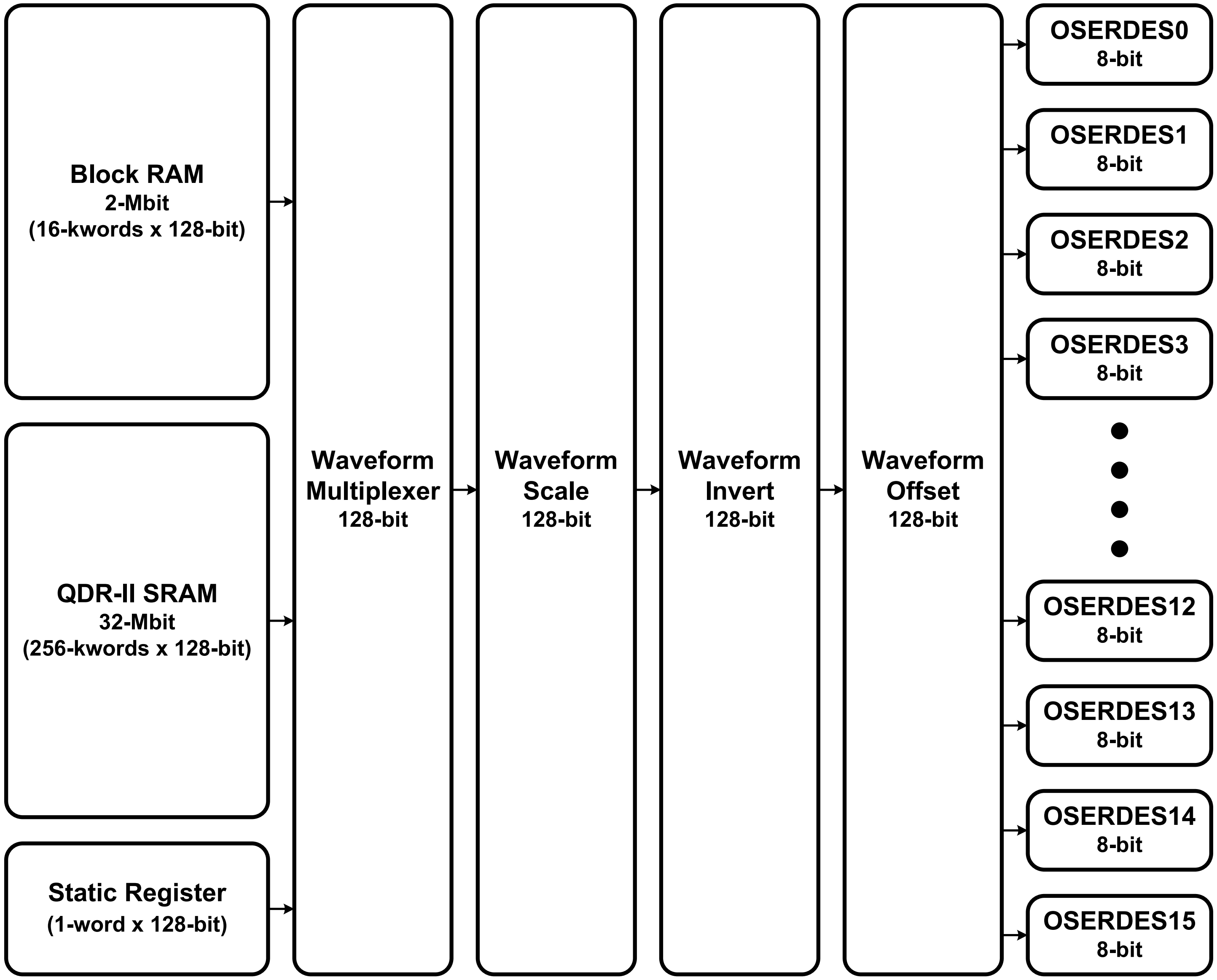

Figure 2.4: Baseband signal source data path FPGA block diagram

2.2.1.1 Waveform Storage and Playback

The signal source was designed to play back arbitrary waveforms, which include sinusoid waveforms. The amount of storage required for a sinusoid waveform of frequency F_c can be calculated using Equation 2.1, where F_s is the sample frequency of the DAC and R is the resolution of the DAC.

$$NumBits = \frac{F_s}{F_c} \cdot R \tag{2.1}$$

The waveform storage and playback sub-system uses a 128-bit data bus to transport data from each source to the high-speed DAC interface sub-system. The internal data path width places several constraints on the waveform stored in memory.

- Waveform must be a multiple of 128 in terms of bits.
- Waveform must be a multiple of 16 in terms of bits.
- Waveform must meet minimum length requirements, which vary depending on the storage memory used.
- Waveform must meet maximum length requirements, which vary depending on the storage memory used.

In the event that NumBits does not meet the length or modulo requirements, it must be replicated N times until the requirements are met. Special care must be taken when replicating waveforms, since some waveforms will require more than a single cycle to be replicated based on the starting and stopping points of the waveform. See Appendix Chapter A for a detailed description of waveform file generation.

The internal data bus width is essentially made up of eight 16-bit data samples, which lends itself to polyphase or parallel DSP operations. The data bus operates at a clock rate of 62.5 MHz.

Static Register The static register provides a means to generate either DC values or limited 8-sample waveforms with the signal source.

Block RAM The block RAM memory is capable of playing back waveforms up to 128-kSamples, and is arranged as 16-kwords x 128-bits.

QDR-II SRAM The QDR-II SRAM memory is capable of playing back waveforms up to 2-MSamples, and is arranged as 256-kwords x 128-bits.

2.2.1.2 Waveform Scale, Invert, and Offset

The baseband signal source allows the user to adjust the power level, invert the waveform, and change the offset voltage. All three of these features are implemented using a DSP48 slice in the Xilinx Virtex-5 FPGA, and take advantage of the properties of the two's complement numbering scheme. A DSP48 slice can perform a combination of 25-bit x 18-bit multiplies and 48-bit additions.

Scale A waveform signal can be scaled in increments of -6 dB by right-shifting the waveform data 1-bit for each desired attenuation setting as shown in Table 2.1. The baseband signal source implements an arithmetic shifter, or dynamic shifter, to achieve the desired scaling of the waveform data [5]. The scaling operation is performed by multiplying a one-hot encoded 18-bit value by the waveform data. Equation 2.2 describes the relationship between the attenuation power level and the scale parameter K.

$$Scale = 20 \cdot \log_{10}\left(\frac{2^{\mathrm{K}}}{2^{16}}\right) \tag{2.2}$$

The waveform data is shifted right by $(16 - \mathrm{K})$ bits resulting in a $((16 - \mathrm{K}) \cdot -6$ dB$)$ attenuation for K in the range of 0 to 16.

One-Hot Value	One-Hot Encoded Value (hex)	Attenuation (dB)
2^{16}	0x10000	0
2^{15}	0x08000	-6
2^{14}	0x04000	-12
2^{13}	0x02000	-18
2^{12}	0x01000	-24
2^{11}	0x00800	-30
2^{10}	0x00400	-36
2^{9}	0x00200	-42
2^{8}	0x00100	-48
2^{7}	0x00080	-54
2^{6}	0x00040	-60
2^{5}	0x00020	-66
2^{4}	0x00010	-72
2^{3}	0x00008	-78
2^{2}	0x00004	-84
2^{1}	0x00002	-90
2^{0}	0x00001	-96

Table 2.1: Attenuation versus one-hot scale values

The internal data bus is scaled by employing a bank of eight DSP48 slices configured as dynamic shifters in right-shift mode, which means the scaled waveform data is available in bits 31 to 16 of the 43-bit product.

Invert In some measurement cases, it can be useful to either delay by 180 degrees or invert the waveform signal. When using two's complement numbers, a negation is performed by inverting the hexadecimal value and adding one. For example, the process of negating +5 is achieved by performing the following steps:

1. Invert positive 5: $0b0101 \rightarrow 0b1010$
2. Add one to the result: $0b1010 + 0b0001 = 0b1011$

While this procedure for inverting a two's complement number is straightforward, it requires two operations to achieve the desired result. Rather than using both an inverter and an adder, the inversion can be achieved by the use of a single multiplier. The baseband signal source implements a two's complement inversion by multiplying the waveform data by -1. The internal data bus is either inverted or passed to the offset stage by changing the scale value from -1 (0xFFFF) to 1 (0x0001). A bank of eight DSP48 slices configured as 16-bit x 16-bit multipliers are used to invert the waveform signal.

Offset Depending on the DUT being stimulated, the input signal may require an offset voltage other than ground, or 0 Volts. A waveform signal can be offset by adding an offset value to the waveform data. The baseband signal source output typically swings about ground. An offset voltage is introduced to the waveform data by adding a two's complement 16-bit value, which represents every integer in the range -2^{15} to $\left(+2^{15}-1\right)$. In order to adjust the offset voltage such that the waveform signal swings about $\frac{V_{ampl}}{2}$, the waveform data can be summed with a value of $\left(2^{15}-1\right)$ or 0x7FFF. The waveform signal is offset in the baseband signal source by employing a bank of eight DSP48 slices configured as 17-bit adders, which provide a 16-bit result and a 1-bit overflow flag.

2.2.2 High-Speed Digital-to-Analog Converter

A Texas Instruments DAC5682Z dual-channel, 16-bit, 1 GS/s digital-to-analog converter is used to generate the analog waveforms for the baseband signal source. While this DAC is capable of 1 GS/s, a sample rate of 500 MS/s was used instead. Sampling the DAC at the same frequency of the high-speed analog-to-digital converter (ADC) allowed the signal source and signal analyzer DSP designs to coexist in the same FPGA. This capability is especially important when the signal source is used to stimulate a DUT and measure its performance with the same system.

The high-speed DAC digital interface is made up of 16-bits of double data rate (DDR) data. The digital interface of the DAC operates at 250 MHz, which corresponds to a data rate of 500 Mb/s for each data bit. The Xilinx Virtex-5 SX50T FPGA core logic is only capable of operating at frequencies up to 450 MHz, so a straightforward 16-bit data path could not be implemented. Instead, the width of the data path was extended from 16-bits to 128-bits in order to use the built-in output serializer/deserializer (OSERDES). The OSERDES facilitate higher external data rates, while keeping the internal data bus at a more manageable rate.

In the case of the signal source, the internal data rate is operating at 250 Mb/s divided by 4, which is equivalent to 62.5 Mb/s. The OSERDES are used in an 8:1 DDR configuration, which requires a high-speed clock of 250 MHz and a low-speed clock of 62.5 MHz. The internal data is running at a single data rate, and the external data is running at a double data rate.

The high-speed DAC has two analog outputs, which are capable of sinking a full-scale output current up to 20 mA [6]. A resistor bias network, consisting of a 62 Ω pull-up and 270 Ω pull-down resistor on each half of the differential pair, is required to achieve the maximum output current of 20 mA. The analog outputs expect to see a load 25 Ω, which is achieved by a combination of the resistor bias network and the reconstruction filter. As a result, the maximum voltage on each half of the differential pair will be 500 $\mathrm{mV_{pp}}$. The current sink structure of the DAC also requires a DC common mode of +3.3 V.

2.2.3 Clock Generation and Distribution

A key element of the baseband signal source is the clock generation and distribution scheme. The successful generation of high performance waveforms depends on synchronous clocks. An Analog Devices AD9516-3 14-output clock generator is responsible for generating all clocks for the baseband signal source. A Universal Microwave Corp UMX Series 1 GHz voltage controlled oscillator (VCO) is used to drive the AD9516-3 external RF clock input. Using the external 1 GHz VCO, the output frequency range of the AD9516-3 is 15.625 MHz to 1 GHz. Figure 2.5 describes the clock relationship between the various DSP components of the baseband signal source. The AD9516-3 clock generator is responsible for generating two clocks:

- 100 MHz data path FPGA clock
- 500 MHz high-speed DAC sample clock

Data Path FPGA Clock Generation The data path FPGA uses an internal phase-locked loop (PLL) primitive to generate several clocks from the 100 MHz clock input. The PLL primitive generates the following clocks:

- 250 MHz OSERDES high-speed clock
- 62.5 MHz internal core low-speed clock

The digital clock input (DCLK) of the high-speed DAC is generated by driving a static “10101010” sequence into the OSERDES. Using an OSERDES device to drive the DCLK input of the high-speed DAC allows the DCLK to be precisely aligned with the 16-bit DAC data bus. An alternative and equally acceptable solution to the OSERDES would have been to use an output DDR (ODDR) primitive clocked on both edges of the 250 MHz clock with the D1 input tied to a logic high and the D2 input tied to a logic low. Either solution provides a precise relationship between the clock and data because of their location in the input/output bank (IOB) of the Virtex-5 SX50T FPGA.

High-speed DAC Sample Clock Generation The high-speed DAC is sampled by the 500 MHz clock generated by the AD9516-3 clock generator. The DCLK of the high-speed DAC is driven into an internal delay-locked loop (DLL) to generate two 500 MHz clocks at 0 and 180 degree phase. The 16-bit DDR data is clocked into a FIFO using the two clocks from the internal DLL, and clocked out of the FIFO by the 500 MHz sample clock.

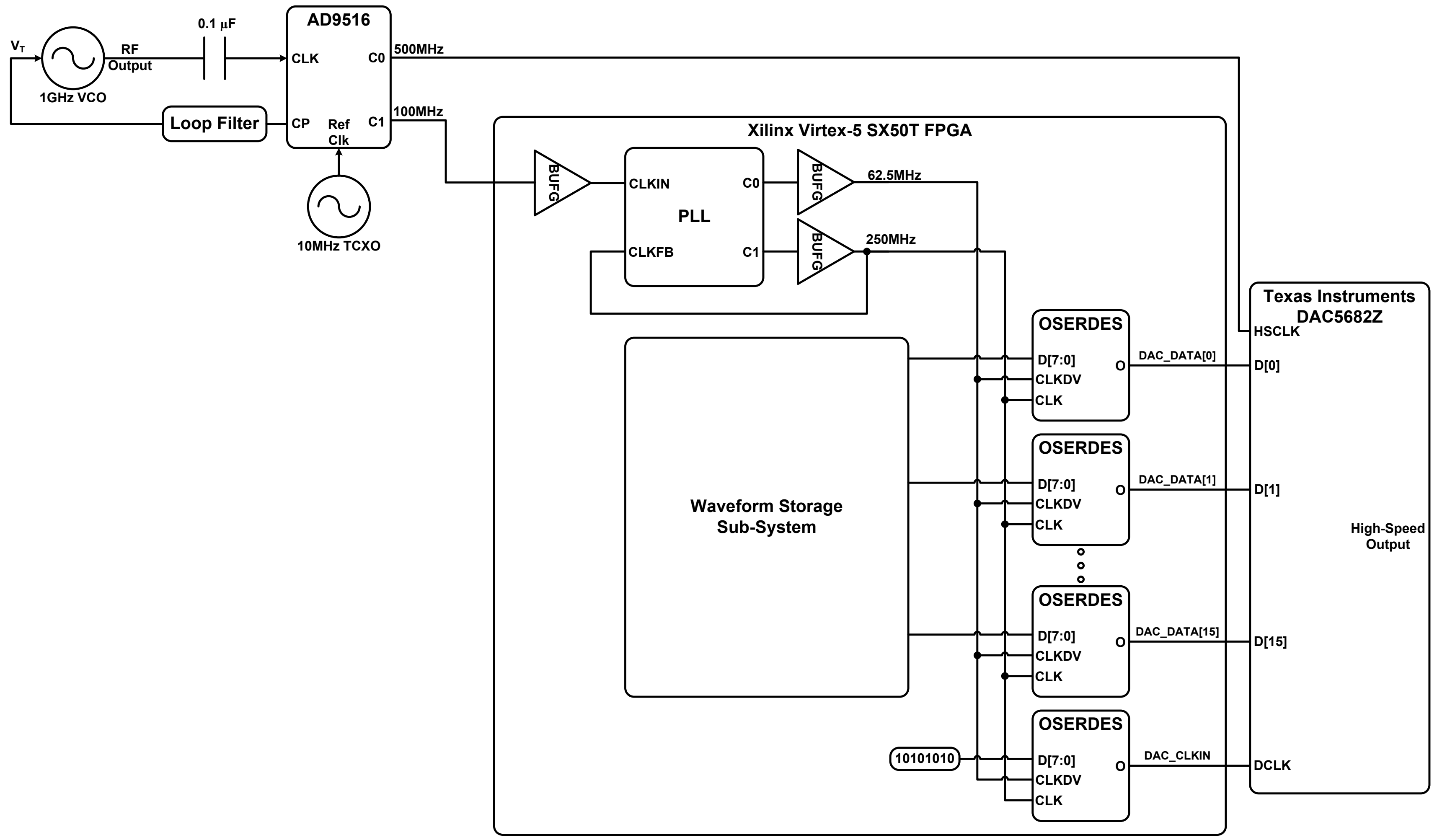

Figure 2.5: Signal Source clock distribution and generation block diagram

2.2.4 Reconstruction Filter

In general, wide-band passive filters are difficult to implement with sufficient ripple and stop-band performance [7]. The role of a reconstruction filter is to attenuate harmonic and alias frequencies to a sufficient power level before they fold back into the pass band of the filter. In a typical sampled system operating in the first Nyquist zone, the cut-off frequency of the reconstruction filter would be set to a frequency slightly less than $\frac{F_s}{2}$. As the filter's cut-off frequency approaches the Nyquist frequency the steepness of the transition band increases. Figure 2.6(a) highlights the shape of the reconstruction filter when sampling at twice the Nyquist Frequency.

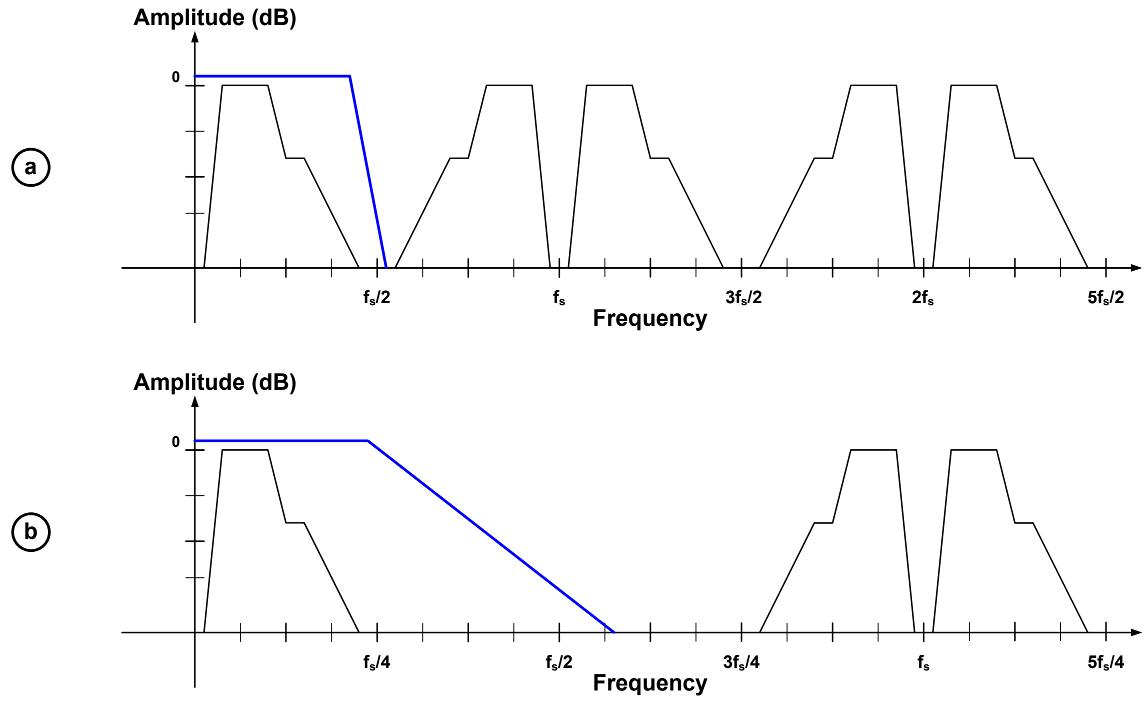

Figure 2.6: (a) Example of Nyquist sampling and the requirements for the reconstruction filter. (b) Example of reconstruction filter constraint relaxation as a result of oversampling by 2.

The baseband signal source takes advantage of oversampling to greatly simplify the implementation of the reconstruction filter. Figure 2.6(b) shows the effects of oversampling on the reconstruction filter. The high-speed DAC was oversampled by a factor of two, which allowed a -3 dB frequency well within the first Nyquist zone. As a result, the signal source can cleanly pass sinusoid waveforms up to 120 MHz with relative ease.

The reconstruction filter was designed using Agilent Technologies' Genesys Filter Synthesis software. The first step in designing the filter involved setting the desired specifications including:

- -3 dB frequency: 130 MHz
- Passband ripple: 0.5 dB
- Stopband attenuation: 60 dB

The second step was to select a filter type, filter shape, and filter topology. The baseband signal source uses a 7th-Order Chebyshev low-pass differential filter topology. Upon defining the filter specifications and topology, the Genesys Filter Synthesis software generated the filter schematic and simulated frequency response shown in Figures 2.7 and 2.8, respectively. The simulated frequency response was evaluated to ensure the desired specifications would be met by the filter design.

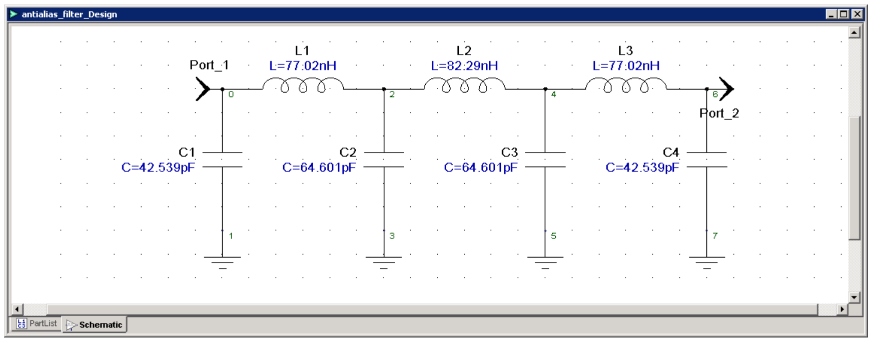

Figure 2.7: Genesys Filter Synthesis 7th-Order Chebyshev low-pass filter schematic

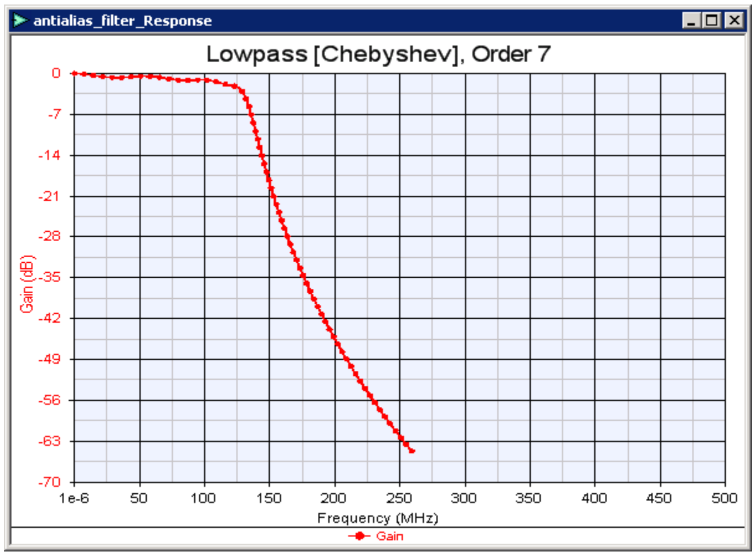

Figure 2.8: Genesys Filter Synthesis simulated frequency response for 7th-Order Chebyshev low-pass filter

The inductor and capacitor values shown in the Figure 2.7 were generated by the Genesys Filter Synthesis software, and were used as a starting point to select practical component values. When choosing component values for the reconstruction filter, it was necessary to use multiple components to achieve the correct values. In the case of inductance, the inductors were placed in series; in the case of capacitance, the capacitors were placed in parallel.

Reference Designator	Calculated	Practical	Series/Parallel
L1	77.02 nH	71 nH	22 nH+27 nH+22 nH
L2	82.29 nH	76 nH	27 nH+22 nH+27 nH
L3	77.02 nH	71 nH	22 nH+27 nH+22 nH
C1	42.539 pF	39.6 pF	1.8 pF+18 pF+18 pF+1.8 pF
C2	64.601 pF	59.9 pF	15 pF+15 pF+15 pF+15 pF
C3	64.601 pF	59.9 pF	15 pF+15 pF+15 pF+15 pF
C4	42.539 pF	39.6 pF	1.8 pF+18 pF+18 pF+1.8 pF

Table 2.2: A comparison of calculated and practical component values for the reconstruction filter

In addition to choosing practical component values, the differential nature of the high-speed DAC output required the single-ended filter design to be converted to a balanced differential topology. This transformation was performed by mirroring the single-ended design about a *virtual* ground reference. The *virtual* ground creates a circuit made up of two series capacitors which results in the capacitance value being cut in half. The value of the series inductors are identical to that of the single-ended design. The schematic shown in Figure 2.9 represents the differential low-pass filter that was simulated using LTSpice.

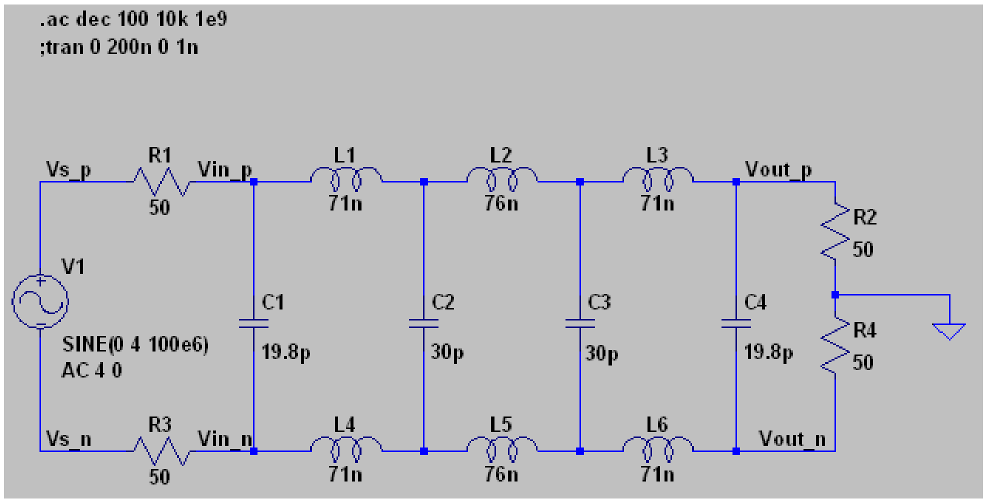

Figure 2.9: LTSpice 7th-Order Chebyshev differential low-pass filter schematic

The simulated frequency response of the Chebyshev filter, shown in Figure 2.10, has a cut-off frequency of 141 MHz, which is different from that specified in the Genesys Filter Synthesis software. However, the increased cut-off frequency provides for more signal bandwidth in the reconstruction filter.

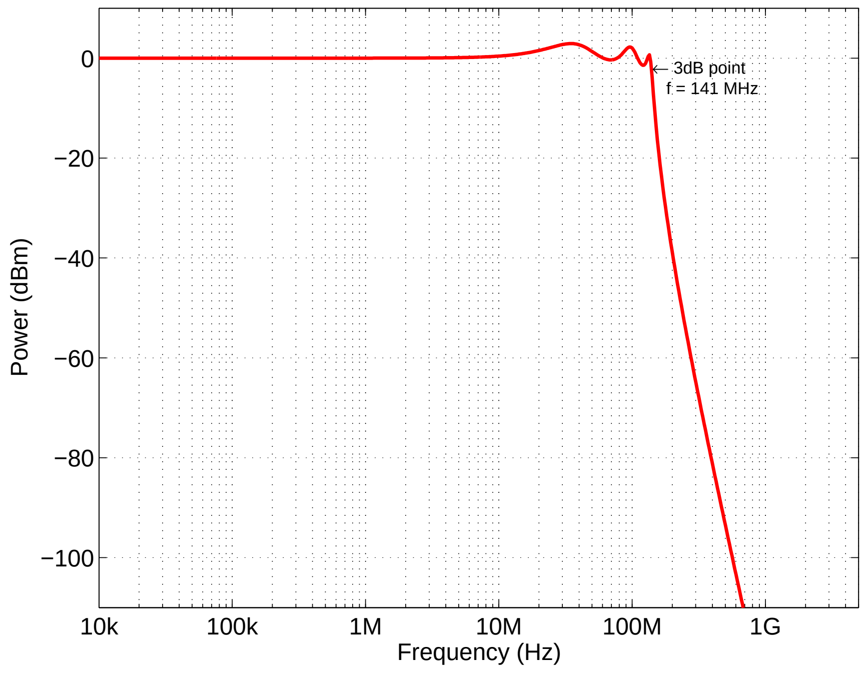

Figure 2.10: LTSpice 7th-Order Chebyshev differential low-pass filter simulated frequency response. The measured performance of the 7th-Order Chebyshev differential low-pass filter is shown in Figure 2.29.

Table 2.3 outlines the estimated performance of the 7th-Order Chebyshev low-pass reconstruction filter.

PARAMETER	**VALUE**
-3 dB Frequency	141 MHz
Bandwidth	120 MHz
Passband Ripple	0.5 dB
Signal Attenuation	2.4 dB
Stopband Attenuation	60 dB

Table 2.3: Estimated reconstruction filter specifications

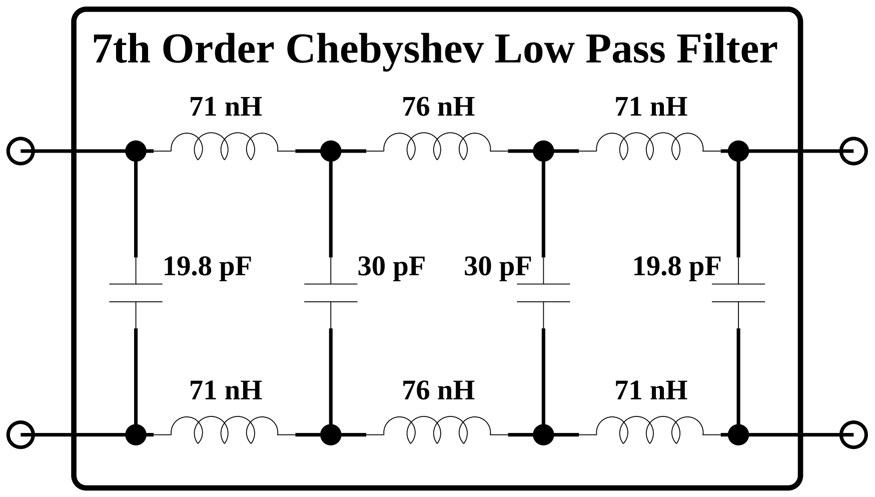

Figure 2.11: Detailed schematic of the signal source reconstruction filter

The final implementation of the reconstruction filter used in the signal source is shown in Figure 2.11. The bill of materials for the reconstruction filter is shown in Table 2.4.

Reference Designator	**Manufacturer**	**Part Number**	**Description**
C49, C50	Murata Electronics	GRM1885C1H1R8CZ01D	1.8 pF, 50 V Ceramic Capacitor
C28, C29	Murata Electronics	GRM1885C1H180JA01D	18 pF, 50 V Ceramic Capacitor
C18, C19, C23, C24	Murata Electronics	GRM1885C1H150JA01D	15 pF, 50 V Ceramic Capacitor
L82, L87, L88, L89, L90, L91, L92, L93, L94, L95	Coilcraft	0603CS-22NXJL	22 nH, 700 mA Ceramic Chip Inductor
L61, L75, L76, L77, L78, L79, L80, L81	Coilcraft	0603CS-27NXJL	27 nH, 600 mA Ceramic Chip Inductor

Table 2.4: Reconstruction filter bill of materials

2.2.5 High-Speed Amplifier Output Stage

A Texas Instruments OPA695 ultra-wideband, current-feedback operational amplifier with a gain bandwidth of 1400 MHz is used to convert the differential current output of the DAC5682Z high-speed DAC into a single-ended signal [8]. The baseband signal source was designed to drive a DUT with an input impedance of 50 Ω.

The OPA695 amplifier was configured to have a gain of +14 dB, or 2.2 times the gain of the input signal. Using a gain of approximately 2 allows the OPA695 amplifier to achieve its maximum bandwidth of 1400 MHz. The input of the OPA695 amplifier has an effective input impedance of 25 Ω on a 20 mA AC signal.

The baseband signal source output stage was designed to have a maximum gain of +10 dB. The combined signal gain of the reconstruction filter and the OPA695 amplifier equates to approximately +12 dB. A −3 dB PAD, or attenuator, was used to limit the output signal power to no more than +10 dBm. The attenuator was designed using a pi-pad configuration with discrete resistors. Figure 2.12 shows the OPA695 amplifier and attenuator circuits of the baseband signal source.

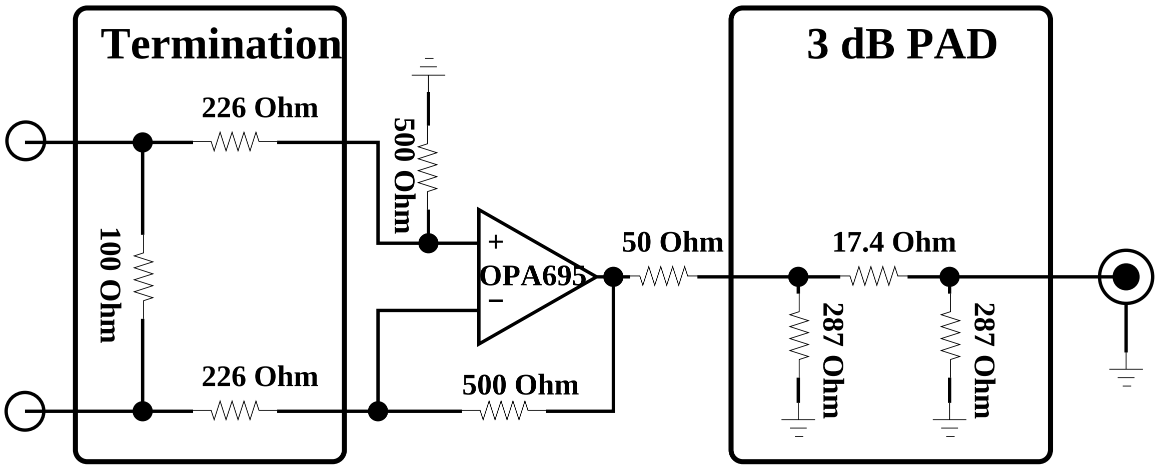

Figure 2.12: High-speed amplifier and attenuator output stage schematic

2.2.6 Waveform Generation Examples

A detailed block diagram of the baseband signal source is shown in Figure 2.14. The signal source was designed to generate both CW and arbitrary waveforms, examples of which are listed below:

- A comb signal consisting of tones across the entire Nyquist bandwidth (Figure 2.13).
- A CW signal operating at 100 MHz in the frequency domain (Figure 2.15).
- A CW signal operating at 100 MHz in the time domain (Figure 2.16).
- A chirp signal consisting of multiple sinusoid waveforms of increasing frequencies from 10 MHz to 150 MHz in 10 MHz steps (Figure 2.17).
- A sawtooth signal (Figure 2.18).
- A burst of single sinusoid cycles with a DC interval of fixed length (Figure 2.19).
- A symmetric ramp waveform at 1 MHz (Figure 2.20).
- A square waveform at 1 MHz with a 50 % duty-cycle (Figure 2.21).

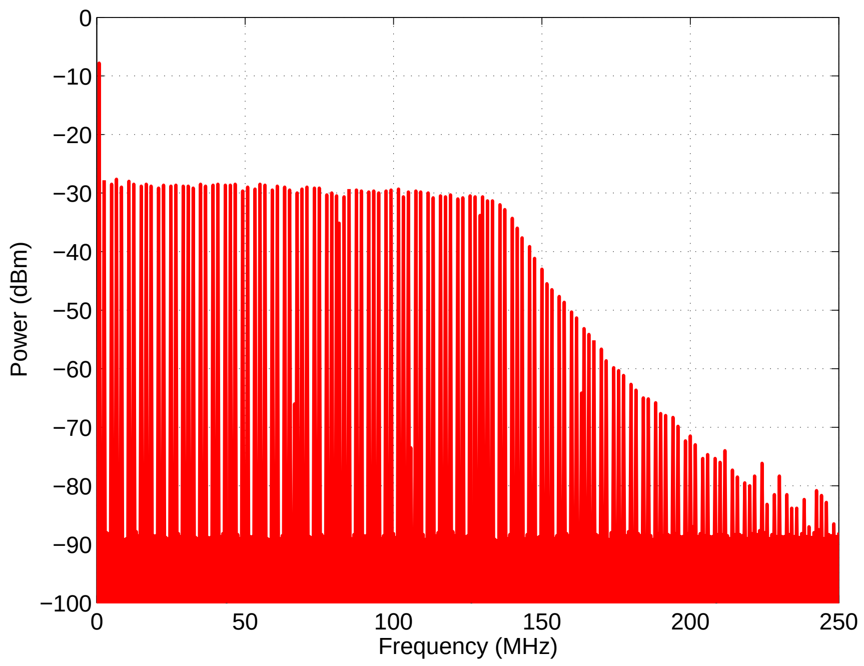

Figure 2.13: Comb Signal Frequency Response with a VBW of 3 kHz and a RBW of 3 kHz. Captured with an HP/Agilent 8562E Spectrum Analyzer.

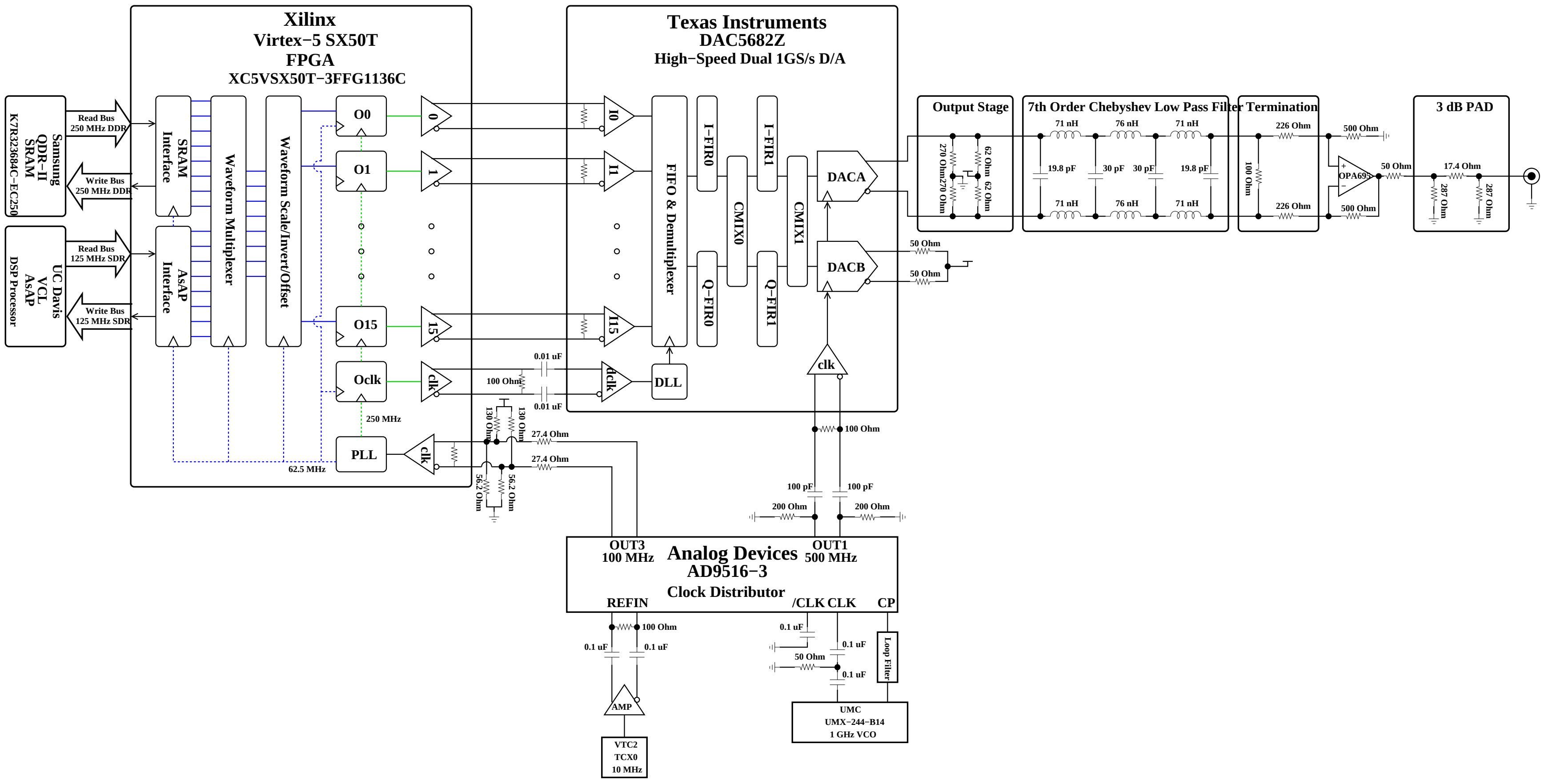

Figure 2.14: Detailed block diagram of baseband signal source

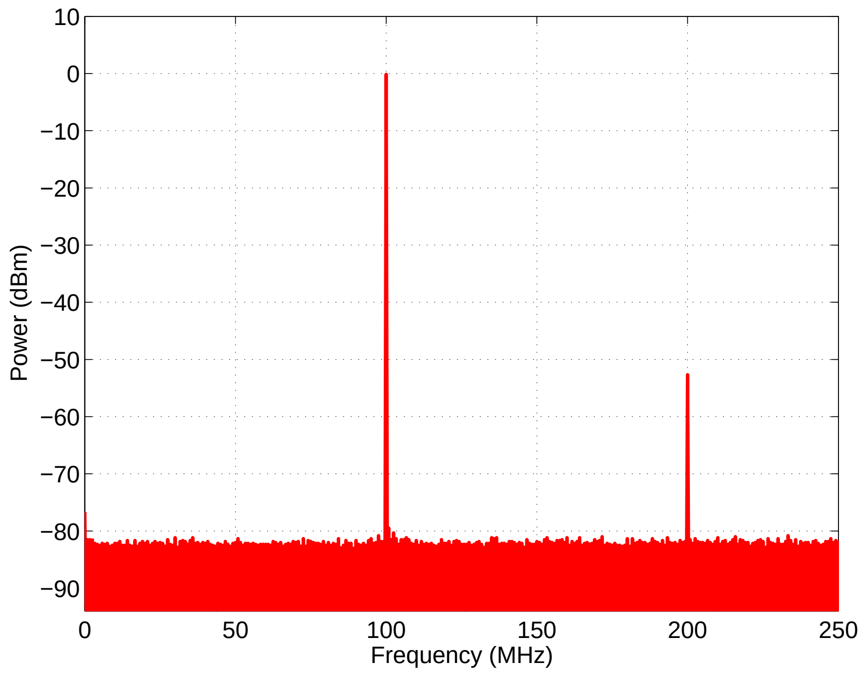

Figure 2.15: Single Tone Sine Wave at a frequency of 100 MHz with a power level of 0 dBFS, a VBW of 3 kHz, and a RBW of 3 kHz. Captured with an HP/Agilent 8562E Spectrum Analyzer.

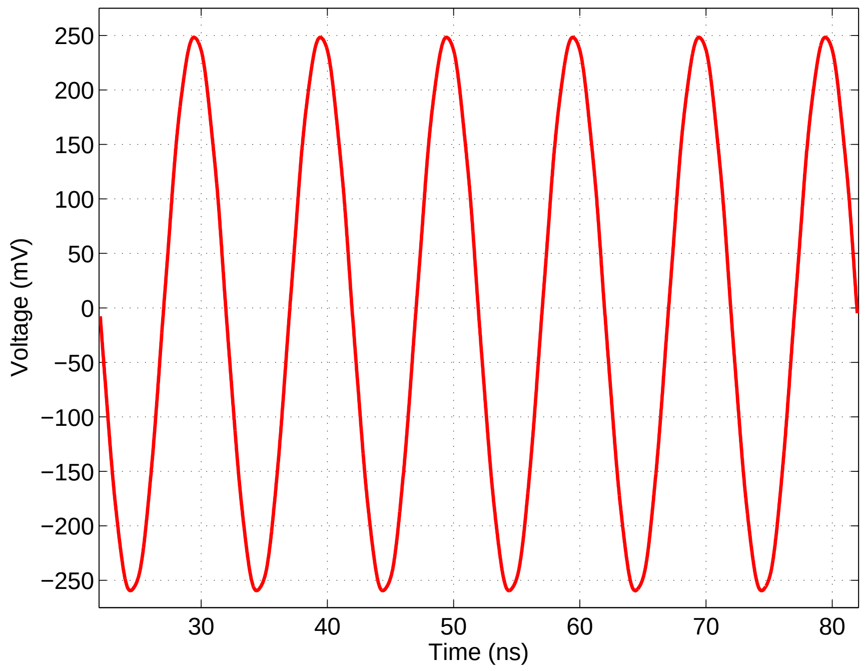

Figure 2.16: Sine Waveform at a frequency of 100 MHz with a power level of 0 dBFS. Captured with an HP/Agilent 83480A Digital Communications Analyzer and an HP/Agilent 83485A 20 GHz Electrical/Optical Plug-In Module.

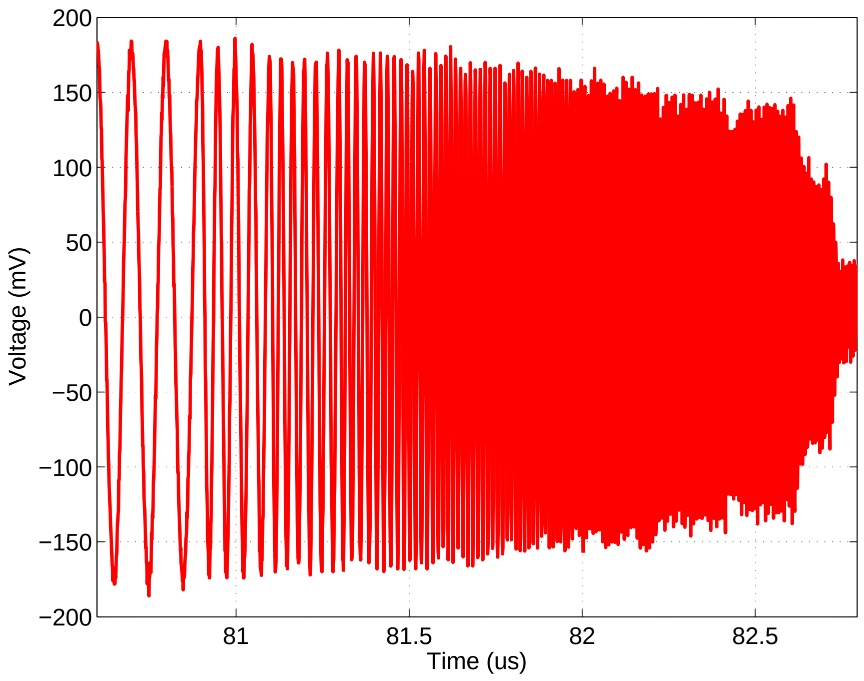

Figure 2.17: Chirp Waveform swept frequency from 10 MHz to 150 MHz in 10 MHz steps. Captured with a Tektronix TDS3054 Digital Phosphor Oscilloscope.

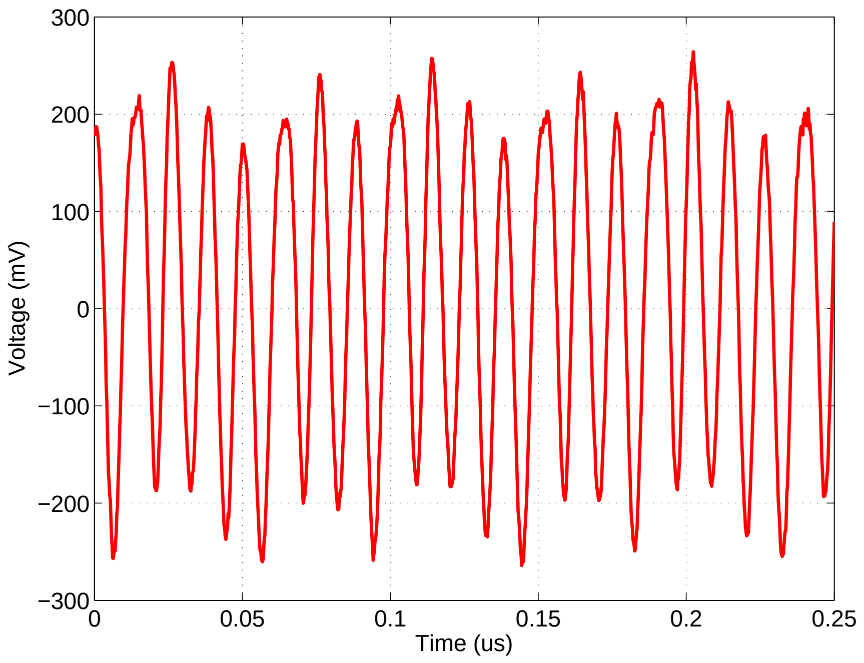

Figure 2.18: Sawtooth Waveform. Captured with a Tektronix TDS3054 Digital Phosphor Oscilloscope.

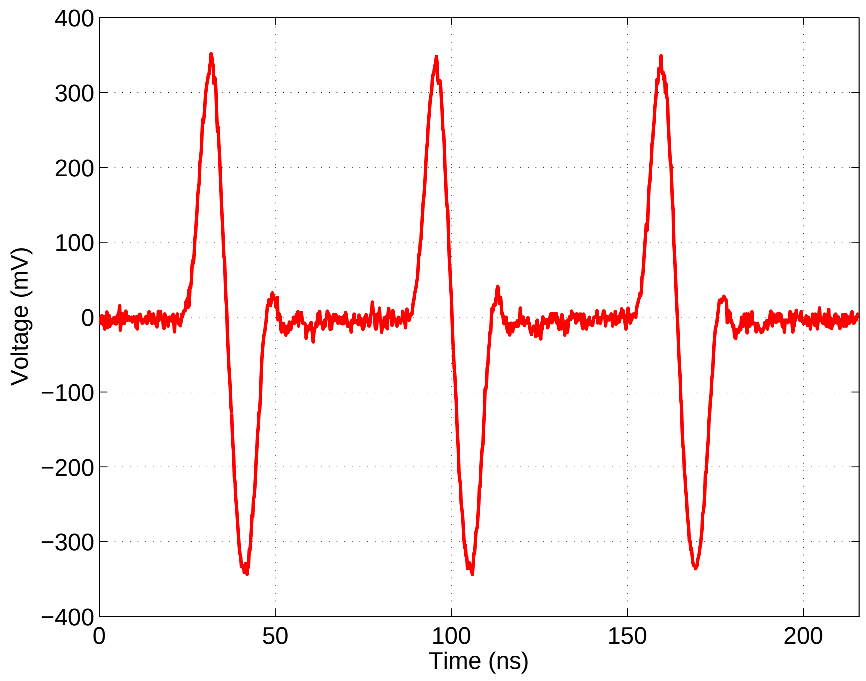

Figure 2.19: Burst Sine Waveform with 20 ns pulse width and 64 ns interval. Captured with a Tektronix TDS3054 Digital Phosphor Oscilloscope.

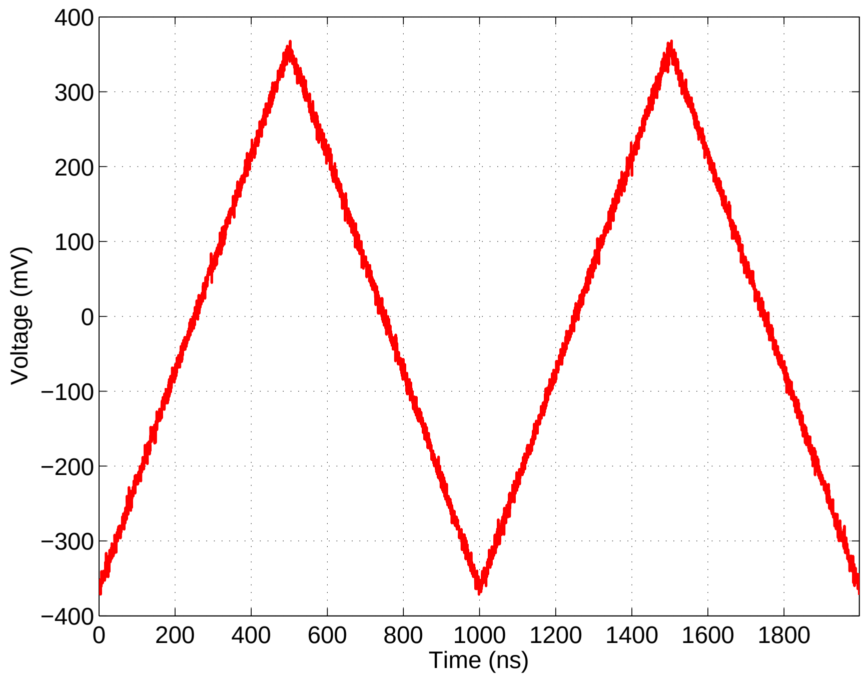

Figure 2.20: Ramp Waveform at a frequency of 1 MHz. Captured with a Tektronix TDS3054 Digital Phosphor Oscilloscope.

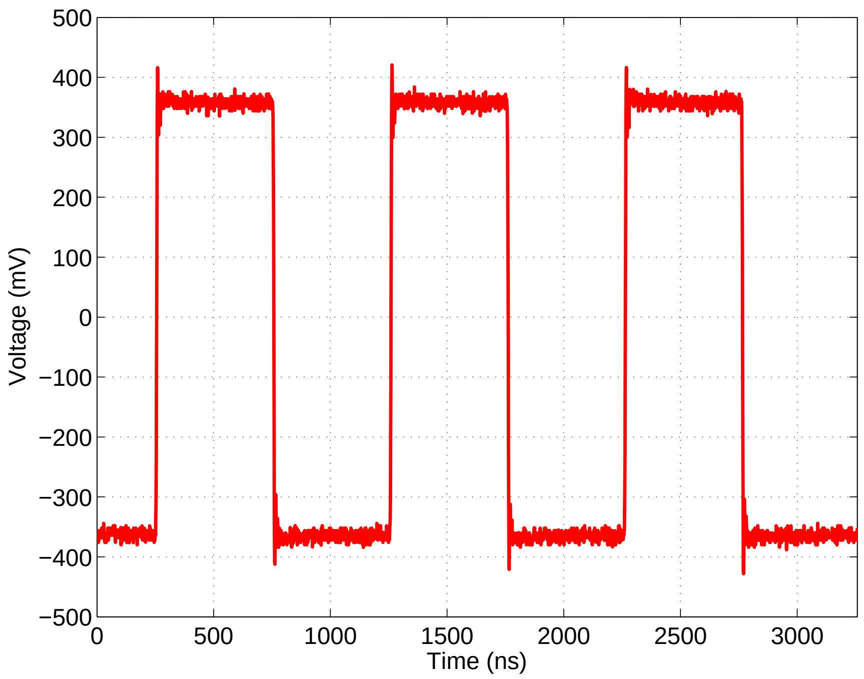

Figure 2.21: Square Waveform at a frequency of 1 MHz with a duty-cycle of 50 %. Captured with a Tektronix TDS3054 Digital Phosphor Oscilloscope.

2.3 Verification

The General Purpose Instrument signal source was designed to be used in test and measurement applications. However, before it can be used its performance must be evaluated and shown to meet the intended specifications. The performance of the signal source was evaluated in both the frequency and the time domain using an assortment of test and measurement equipment. In addition, the performance was evaluated across multiple boards.

2.3.1 Time Domain Measurements

The goal of the time domain measurements was to verify the waveform quality of the signal source when generating a variety of waveform types including: sinusoid, arbitrary, square, and triangle. The signal source time domain characterization was performed using the following equipment:

- HP/Agilent 83480A Digital Communications Analyzer (DCA)
- HP/Agilent 83485A Optical/Electrical Plug-In with a bandwidth of 20 GHz
- 20 dB Attenuator
- HP/Agilent E2050A/B LAN-to-GPIB Gateway

Time domain data is extracted from the HP/Agilent 83480A DCA by use of an HP/Agilent E2050A LAN-to-GPIB gateway. A GPIB connection is made between the HP/Agilent E2050A and the HP/Agilent 83480A DCA. The HP/Agilent 83480A DCA is then controlled via a Perl script over a LAN connection made between the HP/Agilent E2050A and the controlling computer. In addition to control, the Perl script also performed data collection and waveform extraction. The basic measurement block diagram used to collect data is shown in Figure 2.26. The HP/Agilent 83480A DCA was used to collect the following waveform data:

- RMS Voltage
- Duty Cycle
- Frequency
- Period

The above waveform data was used to evaluate the -3 dB frequency.

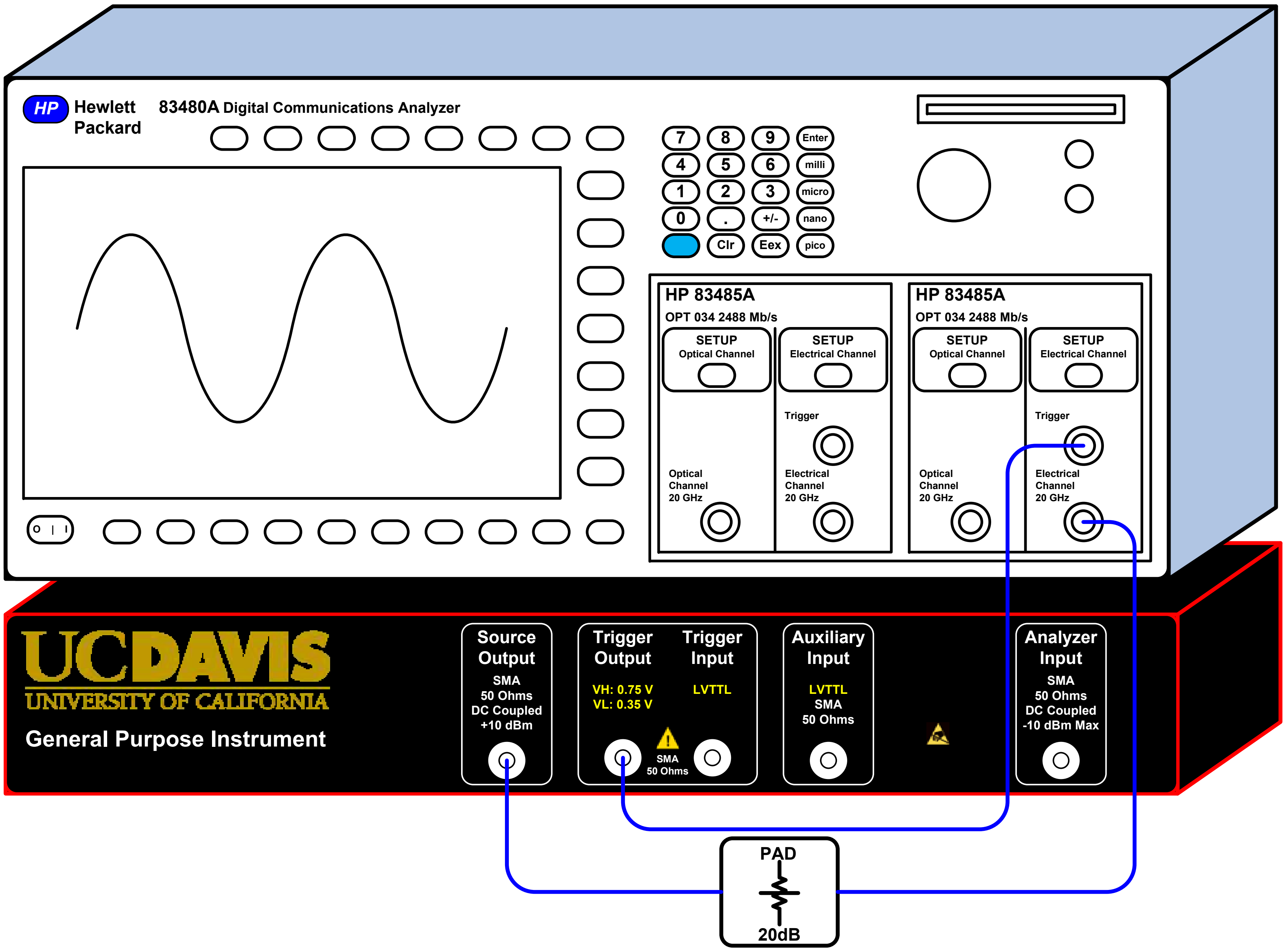

Figure 2.22: Signal Source Time Domain Test Diagram

2.3.1.1 Time Domain Test and Measurement Setup

The HP/Agilent 83480A DCA is an equivalent time sampling oscilloscope, or sampling scope, which measures the instantaneous amplitude of the input waveform at the sampling point. A DCA samples the input signal once per trigger, and adjusts the sample point by a small delay before each sample is taken [9]. The trigger used with the sampling scope must be synchronous with the input waveform in order to properly sample the data. The channel A input of the 83485A Plug-In was driven by the signal source output of the General Purpose Instrument through a 20 dB fixed attenuator. DCA plug-in modules are very sensitive and cannot tolerate signals larger than ± 3 V; the fixed attenuator helps to prevent accidental amplitudes that exceed the recommended limits.

Trigger Sensitivity The trigger input of the DCA plug-in module was driven by the trigger output of the General Purpose Instrument in pattern trigger mode. The optimum trigger point of the DCA is at the 50 % point. The trigger output of the General Purpose Instrument was designed to provide a high level of 750 mV and a low level of 350 mV. The optimum trigger point was calculated using Equation 2.3.

$$V_{trig} = \left(\frac{V_H - V_L}{2}\right) + V_L = \left(\frac{750 \text{ mV} - 350 \text{ mV}}{2}\right) + 350 \text{ mV} = 550 \text{ mV} \tag{2.3}$$

Using this trigger level, the signal source output was initially evaluated at 25 MHz, 50 MHz, and 100 MHz. The results of these initial time domain measurements indicated that the DCA was not triggering properly. Further investigation showed that the trigger output of the General Purpose Instrument was suffering from a high capacitance load caused by the electro-static discharge (ESD) protection diode attached at the output (see Figure 2.23). Refer to Section 4.3.1.4 for more information regarding the trigger output verification results.

The ESD protection diode caused a plateau in the center of the rise time of the trigger signal, as seen in Figure 4.25, which is located at the optimum trigger point of the DCA. Upon removing the ESD protection diode, the trigger signal, shown in Figure 4.26, was sufficient to trigger the DCA. Figure 2.24 shows an example of two 100 MHz sine waveform whose X- and Y- data were extracted from the DCA. The blue dashed-line waveform was captured by the DCA using the trigger with the ESD protection diode removed. The red waveform was captured by the DCA using the trigger with the ESD protection diode present. The red waveform, shown in Figure 2.24, shows signs of waveform distortion, whereas the blue waveform represents a clean sinusoid waveform.

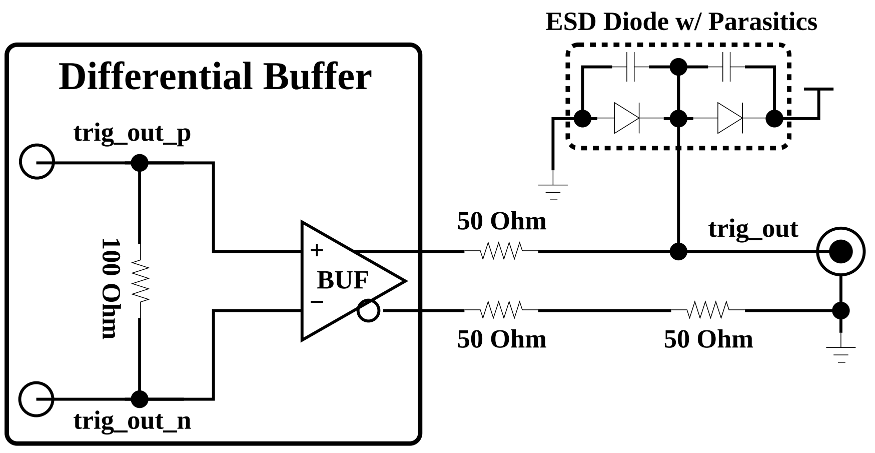

Figure 2.23: Signal source trigger output circuit showing the ESD protection diode and its parasitic capacitance, which caused the reflections on the rise time of the trigger signal shown in Figure 4.25.

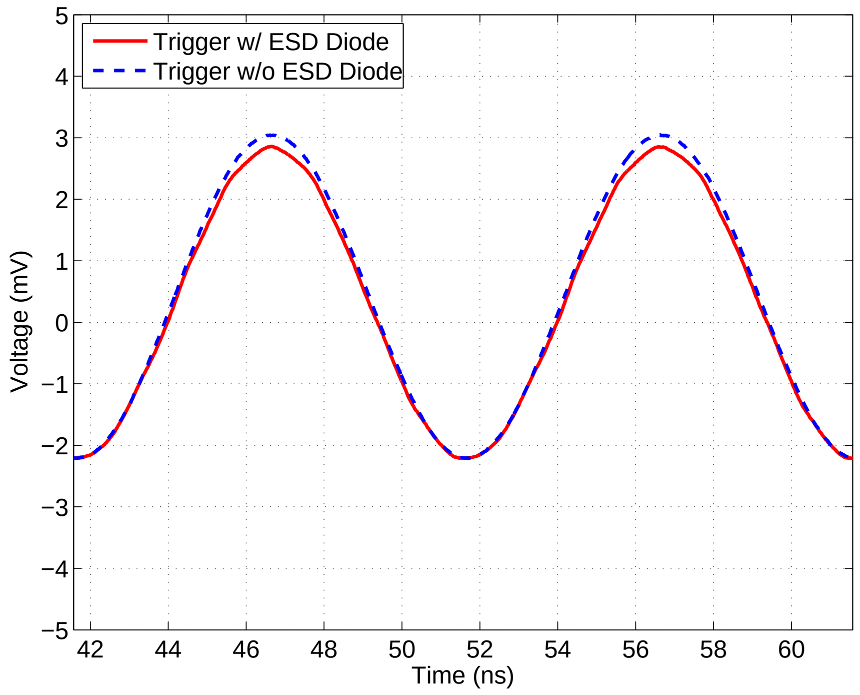

Figure 2.24: Two Sine Waveforms at a frequency of 100 MHz. The blue dashed-line waveform was triggered without the ESD protection diode on the General Purpose Instrument trigger output. The red waveform was triggered with the ESD protection diode on the General Purpose Instrument trigger output. Captured with an HP/Agilent 83480A Digital Communications Analyzer and an HP/Agilent 83485A 20 GHz Electrical/Optical Plug-In Module.

2.3.1.2 −3 dB Frequency Measurements

The −3 dB frequency can be estimated by analyzing the frequency response of the signal source output. The frequency response can be determined by sweeping the signal source from DC to the Nyquist frequency. As the signal source output is swept from DC to the Nyquist frequency, the generated signal will become attenuated. When the signal frequency approaches and passes through the stop band of the reconstruction filter, it will be further attenuated to the point that it becomes distorted. In this condition, the DCA will have difficulty distinguishing the frequency of the generated sine wave. As a result, the DCA will provide only a rough estimate of the −3 dB frequency and the frequency response. Data was collected from DC to the Nyquist frequency, but the data is only useful up to a frequency of 141 MHz. The RMS voltage level of each frequency point is recorded. Using the RMS voltage data, the frequency response can then be displayed using an x versus y plot, where the x-axis is plotted in a logarithmic scale. To plot the frequency response the RMS voltage was first converted to a power level in units of decibels relative to 1 mW (dBm). Equation 2.4 defines the relationship between power in units of Watts and the RMS voltage.

$$P = \left(\frac{V_{RMS}^2}{R_L}\right) \text{ W} \tag{2.4}$$

The power can be converted from units of Watts to units of dBm using Equation 2.5.

$$P_{dBm} = 10 \cdot \log_{10}\left(\frac{P\ [\text{W}]}{0.001\ \text{W}}\right) \text{dBm} \tag{2.5}$$

Rather than converting the measured RMS voltage data to power in units of Watts for use in Equation 2.5, I chose to convert the RMS voltage data directly to units of dBm by manipulating Equations 2.4 and 2.5. First, the power level of 1 mW was converted to the equivalent RMS voltage as shown in Equation 2.6. The DCA plug-in provides a load impedance (R_L) of 50 Ω. Equation 2.6 was determined by solving for V_{RMS} in Equation 2.4.

$$V_{1mW} = \sqrt{P \cdot R_L} = \sqrt{0.001\text{W} \cdot 50\Omega} \approx 0.224 \text{ V RMS} \tag{2.6}$$

Next Equation 2.4 was substituted for P and 1 mW was substituted with its equivalent RMS voltage into Equation 2.5. The equation was then simplified resulting in Equation 2.7.

$$P_{dBm} = 20 \cdot \log_{10}\left(\frac{V_{RMS}}{V_{1mW}}\right) = 20 \cdot \log_{10}\left(\frac{V_{RMS}}{0.224\ \text{V}}\right) \text{dBm} \tag{2.7}$$

The measurement Perl script used Equation 2.7 to convert the RMS voltage to a power level in units of dBm, and to plot the frequency response in order to determine the −3 dB frequency. Figure 2.25 shows the resulting frequency response curve created by measuring the RMS voltage of the signal at each frequency from DC to a frequency of 141 MHz. The measured −3 dB frequency was 137 MHz, which is 4 MHz less than the target cut-off frequency.

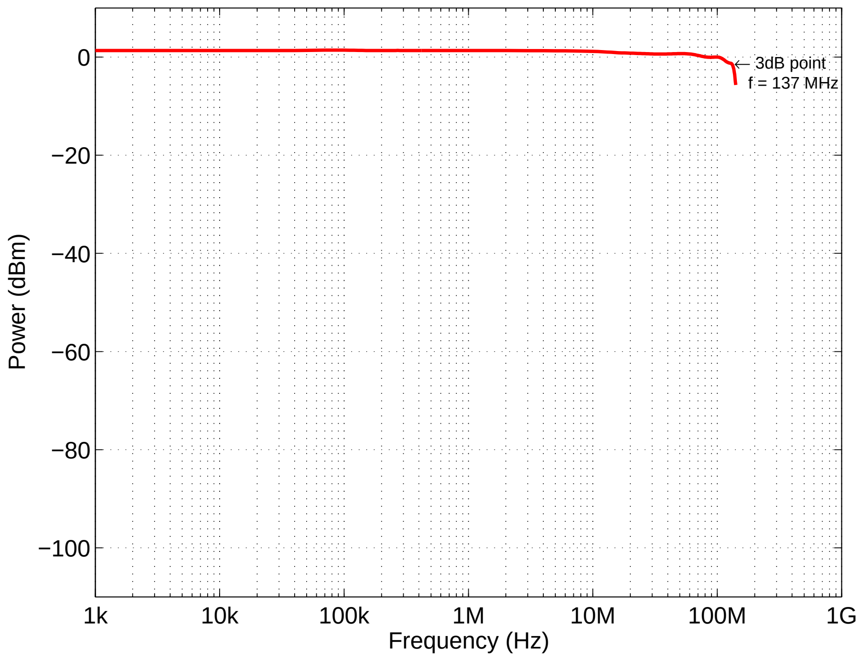

Figure 2.25: Signal Source Frequency Response. Captured with an HP/Agilent 83480A Digital Communications Analyzer and an HP/Agilent 83485A 20 GHz Plug-In module.

2.3.2 Frequency Domain Measurements

The goal of the frequency domain measurements was to verify the quality of the signal source when generating CW signals. The signal source frequency domain characterization was performed using the following equipment:

- HP/Agilent 8562E Spectrum Analyzer (30 Hz to 13.2 GHz)
- Fairview Microwave, Inc. SD3239 DC Blocking (5 kHz to 23 GHz)
- HP/Agilent E2050A LAN-to-GPIB Gateway

Frequency domain data is extracted from the HP/Agilent 8562E Spectrum Analyzer by use of an HP/Agilent E2050A LAN-to-GPIB gateway. A GPIB connection is made between the HP/Agilent E2050A and the HP/Agilent 8562E Spectrum Analyzer. The HP/Agilent 8562E Spectrum Analyzer

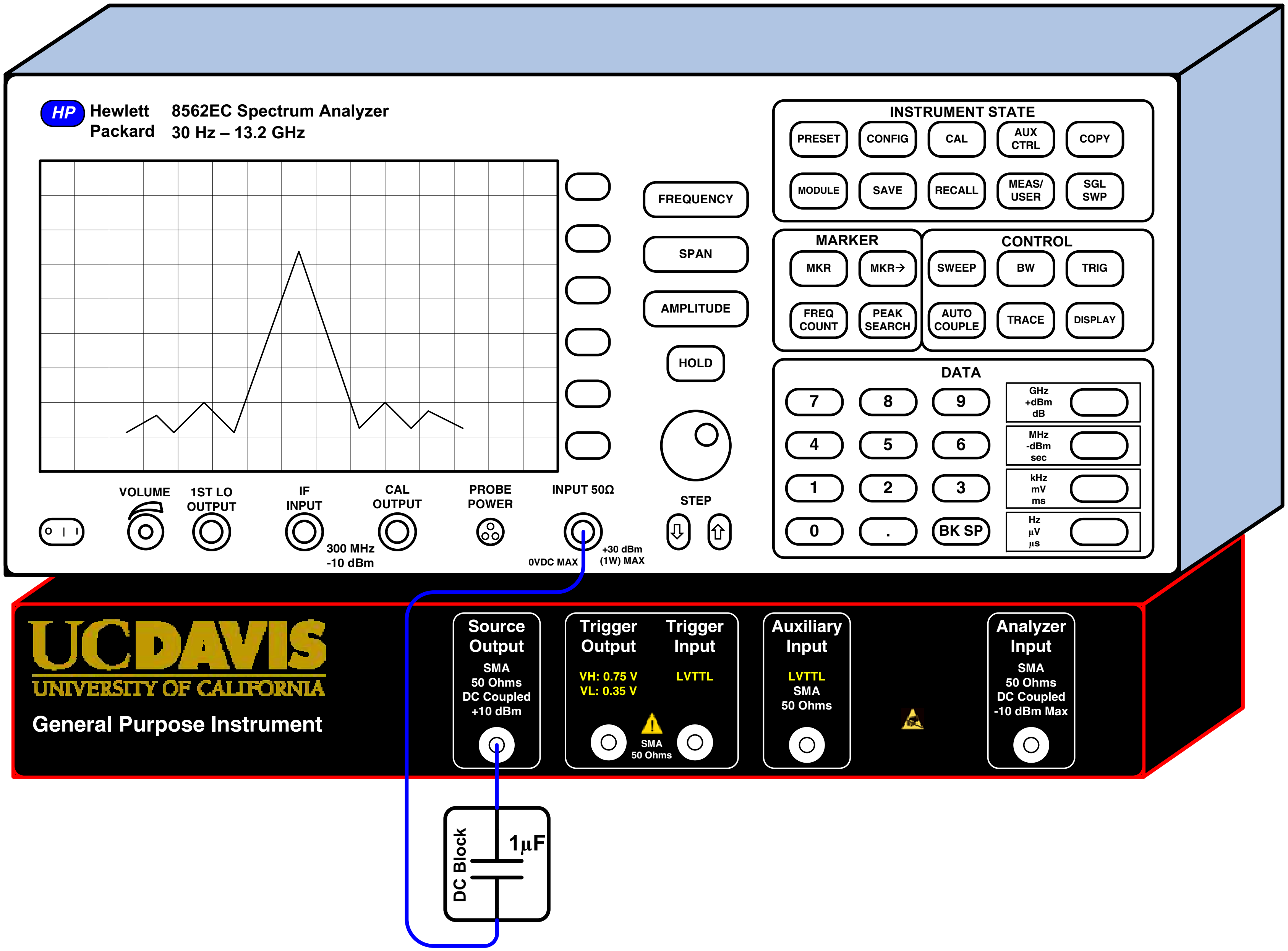

Figure 2.26: Signal Source Frequency Domain Measurement Block Diagram

is then controlled via a Perl script over a LAN connection made between the HP/Agilent E2050A and the controlling computer. In addition to control, the Perl script also performed data collection and waveform extraction. The basic measurement block diagram used to collect data is shown in Figure 2.26.

The HP/Agilent 8562E Spectrum Analyzer was used to collect data in order to calculate the following parameters:

- -3 dB frequency
- Signal-to-noise ratio (SNR)
- Spurious-free dynamic range (SFDR)
- Two-tone third-order intermodulation distortion (IMD_3)
- Third-order intercept point (TOI)

2.3.2.1 -3 dB **Frequency Measurements**

Two methods are commonly used to analyze the frequency response of a signal generator and, in turn, determine the -3 dB frequency. The simplest and least accurate method is to generate a comb signal, also known as a bed of nails, which contains signal tones from DC to the Nyquist frequency. The comb signal used to stimulate the low-pass filter is shown in Figure 2.27. The resulting signal received by the HP/Agilent 8562E Spectrum Analyzer will have the shape of a low-pass filter's frequency response. From this information the -3 dB frequency of the signal source output can be roughly estimated. Figure 3.30 shows the resulting frequency versus power level data extracted from the HP/Agilent 8562E Spectrum Analyzer with a -3 dB frequency between 137.5 MHz and 140 MHz, which is 10 MHz above the target -3 dB frequency. The increased -3 dB frequency results in more signal bandwidth.

A more accurate method to analyze the frequency response is to sweep the signal source from DC to the Nyquist frequency. The power level of each each frequency point is recorded. Using the power level information, the frequency response can then be displayed using an x versus y plot, where the x-axis is plotted in a logarithmic scale. Figure 2.29 shows the resulting frequency response curve created by measuring the power level of the fundamental signal tone at each frequency from DC to the Nyquist frequency. The measured -3 dB frequency was 138 MHz, which is 8 MHz more than the target cut-off frequency.

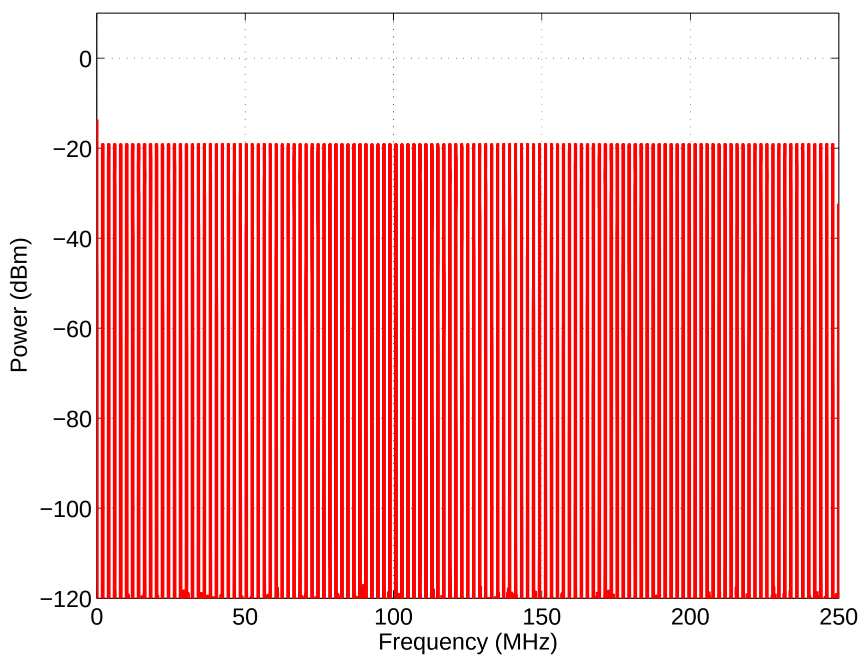

Figure 2.27: Comb input signal used to stimulate the signal source to evaluate the filter's frequency response.

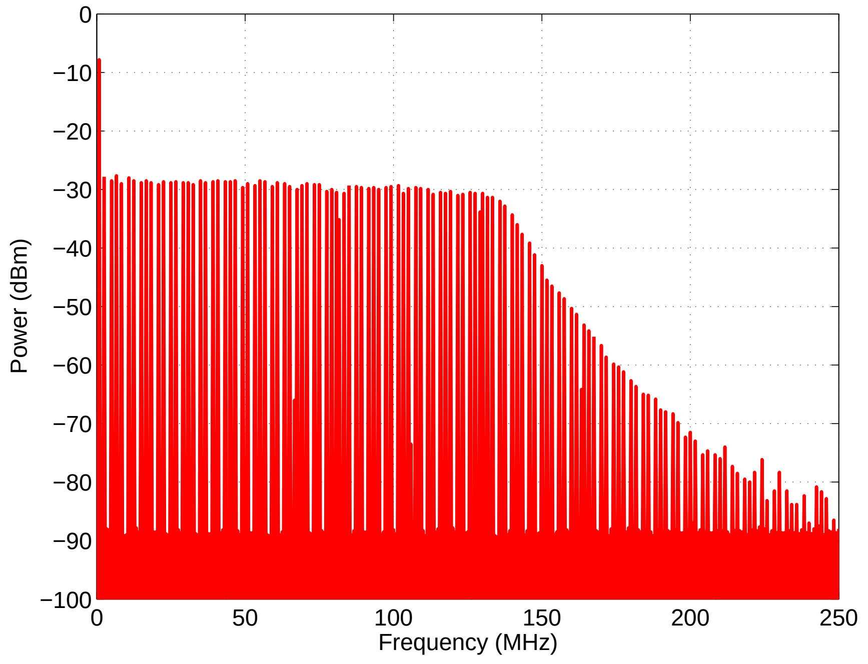

Figure 2.28: Comb Signal Frequency Response with a VBW of 3 kHz and a RBW of 3 kHz. Captured with an HP/Agilent 8562E Spectrum Analyzer.

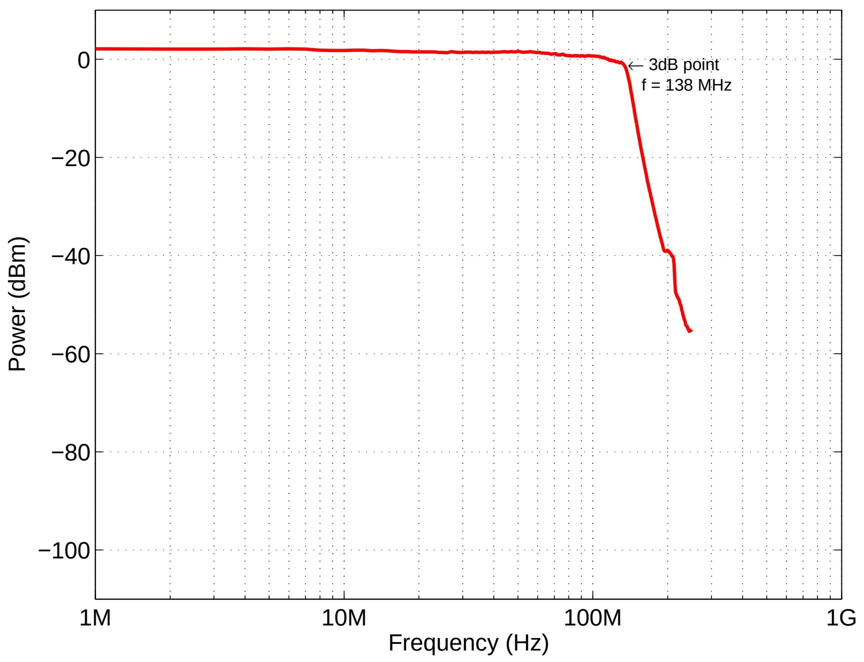

Figure 2.29: Signal Source Frequency Response. Captured with an HP/Agilent 8562E Spectrum Analyzer.

2.3.2.2 Signal-to-Noise Ratio

SNR is the ratio of the power of the fundamental signal tone (P_S) to the noise floor power (P_N), excluding the power at DC and in the first nine harmonics.

$$\mathrm{SNR} = 10 \cdot \log_{10}\left(\frac{P_S}{P_N}\right) \tag{2.8}$$

SNR is typically specified in units of dBc, or dB to carrier, when the absolute power of the fundamental signal tone is used as the reference. Figure 2.30 shows the resulting SNR performance when measured from DC to the −3 dB frequency.

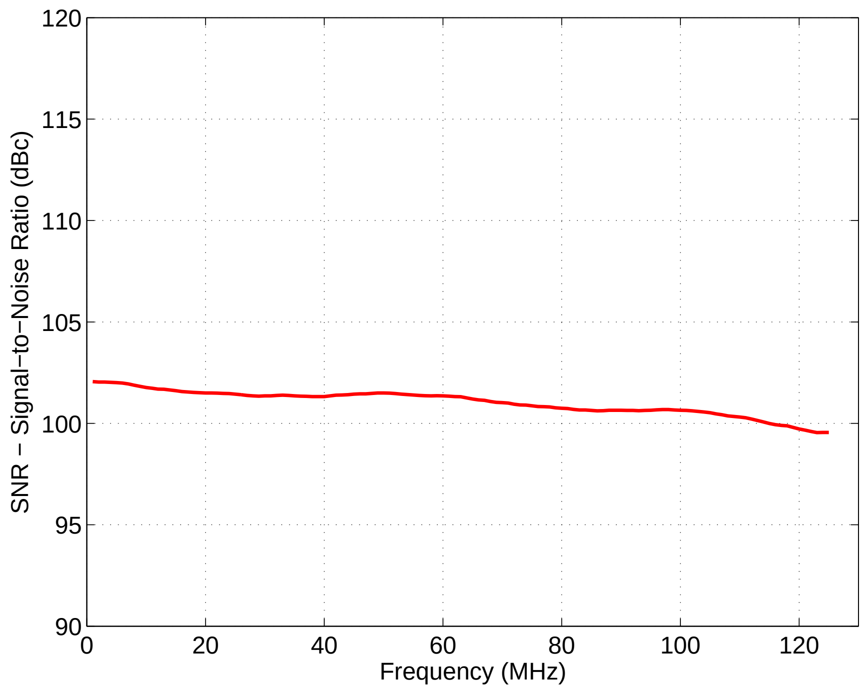

Figure 2.30: Signal-to-Noise Ratio

2.3.2.3 Spurious-Free Dynamic Range

The spurious-free dynamic range is typically measured at three power levels: 0 dBFS, -6 dBFS, and -12 dBFS. As a result, the frequency response extraction routine was performed three times, once for each power level. A measure of a signal's power level in decibels relative to full scale is referred to in units of dBFS. A power level of 0 dBFS represents the maximum possible level of a device. A power level of -6 dBFS represents a 2x reduction in signal power, as shown in Equation 2.9.

$$20 \cdot \log_{10}\left(\frac{2^{14}}{2^{15}}\right) = -6.0206 \text{ dBFS} \tag{2.9}$$

Likewise, a power level of -12 dBFS represents a 4x reduction in signal power, as shown in Equation 2.10.

$$20 \cdot \log_{10}\left(\frac{2^{13}}{2^{15}}\right) = -12.0412 \text{ dBFS} \tag{2.10}$$

The SFDR performance of the signal source was measured by modifying the frequency response extraction routine. A power level measurement was recorded for each harmonic and alias frequency as the signal source is swept from DC to the Nyquist frequency. An alias and harmonic algorithm was used to determine which harmonic frequencies fold back into the passband of the reconstruction filter. Each signal frequency was measured out to the 25^{th} harmonic frequency. The power level of the fundamental signal tone (P_o) along with the power level of the next largest spurious signal (P_n) were used to calculate the spurious-free dynamic range (P_{SFDR}). Equation 2.11 describes the relationship between P_o, P_n, and P_{SFDR}.

$$P_{SFDR} = P_o - P_n \tag{2.11}$$

The average SFDR performance of the signal source, shown in Figure 2.31, represents measurements performed on three measurement boards. The frequency and power level data was extracted from an HP/Agilent 8562E Spectrum Analyzer.

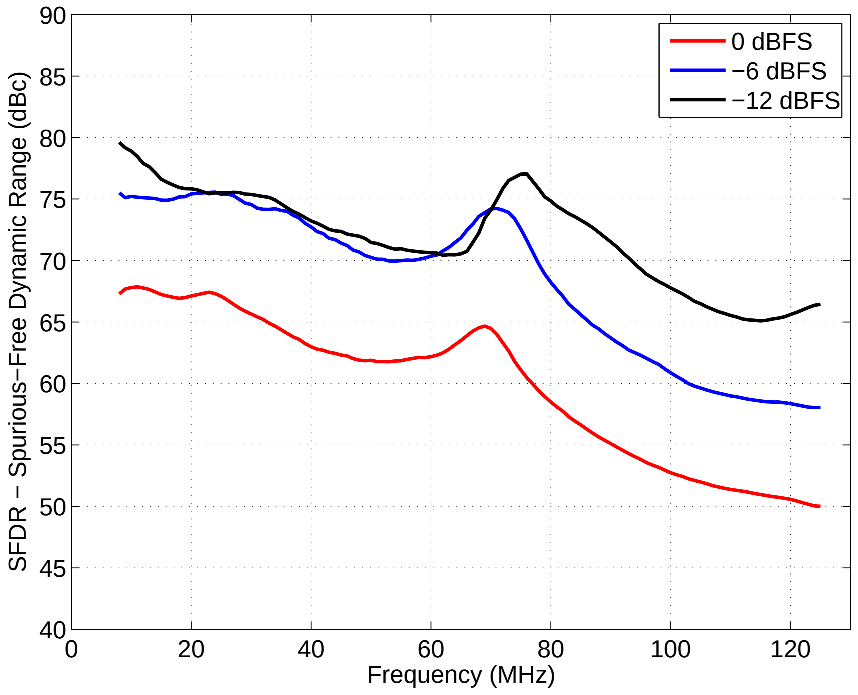

Figure 2.31: Spurious-Free Dynamic Range

2.3.2.4 Two-tone Third-order Intermodulation Distortion Measurement

The presence of multiple signals in a system is sometimes desired; for example, the generation of multi-tone signals or comb signals. However, undesired signals (e.g., noise) are typically present in a signal system, and can mix with the desired signal to generate distortion products. Understanding the effects of distortion is important when evaluating the measurement results of a DUT.

Intermodulation distortion is a type of distortion caused by the presence of two or more signal tones at the input of a non-linear device [10]. This distortion causes spurious signals to be generated, which are related to the original signal tones. The complexity of the distortion increases as the number of signal tones present in the system increases beyond two. As such, the distortion performance of signal systems are typically analyzed with two signal tones. The relationship of the two original signal tones and the generated spurious signal tones is described by Equation 2.12.

$$M \cdot f_1 \pm N \cdot f_2, \text{ where } M,\ N \ = \ 0,\ 1,\ 2,\ 3,\ \ldots \tag{2.12}$$

The order of the distortion product is represented by the sum of $M+N$. For example, the third-order intermodulation products of two signals at f_1 and f_2 would be:

$$2 \cdot f_1 + f_2$$

$$2 \cdot f_1 - f_2$$

$$f_1 + 2 \cdot f_2$$

$$f_1 - 2 \cdot f_2$$

Third-order two-tone intermodulation distortion is a metric used to describe the distortion performance of a transmitter or receiver when multiple signal tones are present in the data stream. It is measured by driving two spectrally pure sine waves through the DUT at frequencies f_1 and f_2, where the difference in frequency is small. The amplitude of each tone is generally attenuated by 6 dB to avoid clipping in the signal system. It is typically specified in dBc relative to the value of either of the two input tones [11]. Figure 2.32 shows the test setup used to measure third-order two-tone intermodulation distortion.

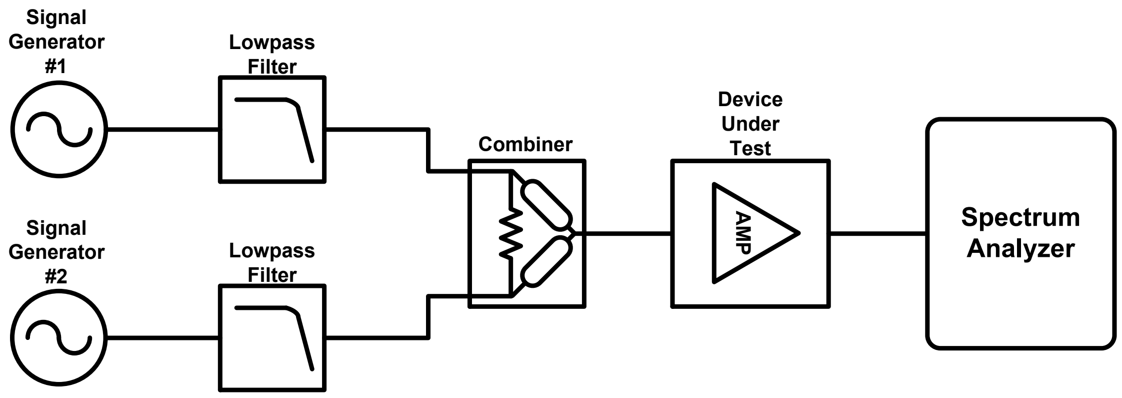

Figure 2.32: Two-tone third-order intermodulation distortion measurement block diagram

The IMD$_3$ performance of the signal source was measured using two signal tones at frequencies:

- $f_1 = 107.7$ MHz
- $f_2 = 107.9$ MHz

Figure 2.33 shows the resulting IMD$_3$ measurement performed with an HP/Agilent 8562E Spectrum Analyzer. The IMD$_3$ parameter was determined by measuring the power level of a fundamental signal tone, f_1 or f_2, (P_o) and one of the spurious tones (P_{o_3}) in units of dBm. Equation 2.13 describes the relationship between P_o and P_{o_3} when calculating IMD$_3$.

$$IMD_3 = P_o - P_{o_3} = -7.5 \text{ dBm} - (-75.66 \text{ dBm}) = 68.16 \text{ dBc} \tag{2.13}$$

An additional parameter, known as the third-order intercept (TOI) point, can be used to quantify the distortion performance of a signal source. TOI can be calculated using the power level of the fundamental tones along with the third-order two-tone intermodulation distortion, and is typically specified in dB. Equation 2.14 describes the relationship between P_o and IMD$_3$.

$$TOI = \frac{IMD_3}{2} + P_o = \frac{68.16 \text{ dBc}}{2} + (-7.5 \text{ dBm}) = 26.58 \text{ dB} \tag{2.14}$$

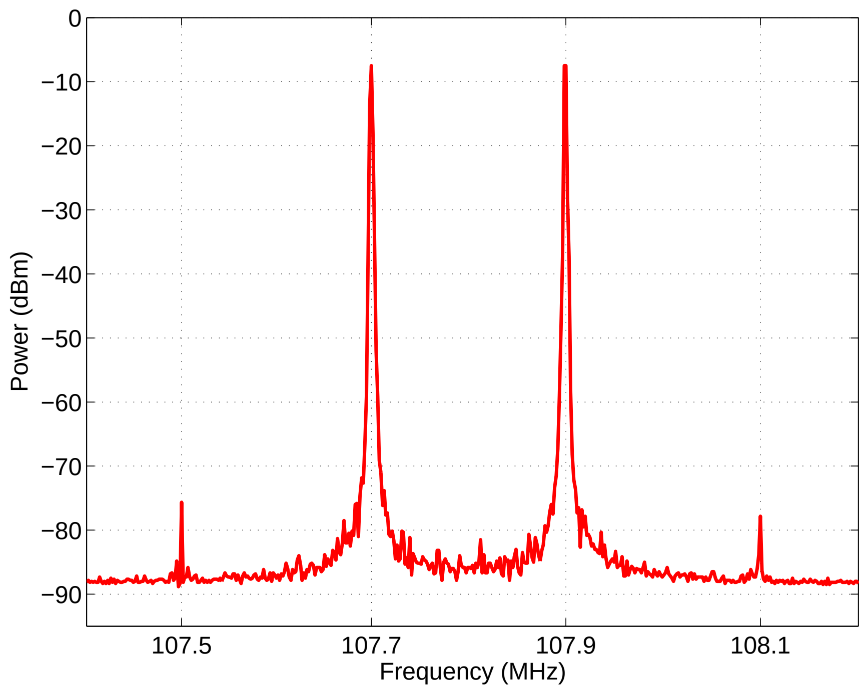

Figure 2.33: Two-tone third-order intermodulation distortion (IMD_3) plot for Signal Source with a attenuation factor of 30 dB, a VBW of 1 kHz, a RBW of 1 kHz, a SPAN of 1 MHz. Tone frequencies: f_1 = 107.7 MHz, f_2 = 107.9 MHz. Captured with an HP/Agilent 8562E Spectrum Analyzer.

2.3.2.5 Frequency Domain Test and Measurement Methodology

The frequency domain data necessary to calculate SFDR, SNR, and the frequency response of the signal source output was collected simultaneously. The data collection was performed by an automated measurement Perl script, which controlled both the spectrum analyzer and the measurement board via remote GPIB and USB interfaces. The measurement data took an average of 60 hours to collect for a single measurement board, and the signal source output of four measurement boards was characterized to determine the average performance. As a result, it was critical to outline the test procedure and determine the spectrum analyzer settings required to make the best possible measurement early on in the characterization stage.

Frequency Bands The spectrum analyzer settings and test procedure were empirically determined over the frequency range from DC to 250 MHz. Given the wide bandwidth of operation for the signal source, the measurements were divided into eight frequency bands. The eight frequency bands used by the measurement script are listed below:

- DC to 100 kHz
- 100 kHz to 500 kHz
- 500 kHz to 1 MHz
- 1 MHz to 15 MHz
- 15 MHz to 30 MHz
- 30 MHz to 60 MHz
- 60 MHz to 85 MHz
- 85 MHz to 250 MHz

The final frequency band is the widest because it encompasses the transition band and stop band of the signal source reconstruction filter.

Signal on Screen As the generated signal passes through the reconstruction filter of the signal source output, the signal amplitude will attenuate as the frequency is increased from DC to 250 MHz. Some frequency domain specifications require the generated signal to be further attenuated by fixed amounts in reference to the full scale output. In the case of SFDR, the attenuation of the generated signal is varied by 0 dBFS, −6 dBFS, and −12 dBFS. In addition, SFDR requires the power level of each alias and harmonic signal tone to be measured.

When performing frequency domain measurements with a spectrum analyzer, it is important to keep as much of the signal on the screen, and preferably in the upper half of the screen, without clipping the signal. This was achieved by adjusting the following spectrum analyzer parameters for each measurement frequency band:

- Start Frequency
- Stop Frequency
- Sweep Time
- Reference Level (RL)
- Marker Peak Excursion (MKPX)
- Marker Peak Threshold (MKPT)
- Resolution Bandwidth (RBW)
- Video Bandwidth (VBW)

For a more detailed description of these spectrum analyzer parameters, and how they affect frequency domain measurements, refer to application note 150 entitled “Spectrum Analyzer Basics” provided by Agilent Technologies [12].

In addition to varying the spectrum analyzer settings in each measurement frequency band, the reference level of the spectrum analyzer front end is further adjusted for the SFDR measurement in the presence of known attenuation. The fundamental and second harmonic frequency tones are assigned a unique reference level, and the remaining harmonic tones are assigned identical frequency tones.

In the case of the fundamental tone of each signal frequency, the displayed signal was large and required an adjustment of the reference level in the range of −5 dBm to 20 dBm, depending on the frequency of the tone. Signals less than 1 MHz required a larger adjustment of reference level. For the second harmonic frequency, the signal required an adjustment of the reference level in the range of −60 dBm to −30 dBm. For all remaining harmonic and alias frequencies, the signal required an adjustment of the reference level in the range of −70 dBm to −30 dBm.

Harmonic and Alias Frequencies The algorithm for determining the location of alias frequencies in the first Nyquist zone, as described below, was leveraged from the Maxim Integrated Products application note 3716 entitled "Folded-Frequency Calculator" [13].

1. Calculate the harmonic frequencies out to the 25^{th} harmonic based on the current signal frequency ($fin) and store the values in an array of arrays (@fharmAoA). The index of the harmonic frequencies is stored along with the harmonic frequency.

2. Determine in which Nyquist zone ($zone) the harmonic frequency is located.

3. Determine if the Nyquist zone containing the harmonic frequency is odd or even ($zonecheck).

4. Determine the location of the alias frequencies for those harmonic frequencies based on the Nyquist zone in which they reside ($fodd and $feven).

5. Store the alias frequencies ($floc) in an array of arrays (@flocAoA) for later use in the measurement Perl scripts. The Nyquist zone, harmonic frequency index, and harmonic frequency are stored along with the alias frequency location.

The Perl code, shown in Listing 2.1, describes the algorithm in detail. This sub-routine was used by the measurement Perl script responsible for characterizing the signal source in the frequency domain. For CW, or sinusoid signals, harmonic frequencies will be present both in-band and out-of-band. The signal source operates in the first Nyquist zone, therefore all out-of-band harmonic frequencies will alias or fold back into the passband of the reconstruction filter. Any out-of-band harmonic signals that fold back onto in-band signals of the same frequency will add together, resulting in an increased power level. Fortunately, the resulting alias signals will be attenuated. For example, a signal operating within the first Nyquist zone at a frequency of 100 MHz will have harmonic frequencies that fold back into the first Nyquist zone. In this case, the 4^{th}, 6^{th}, 9^{th}, 11^{th}, 14^{th}, 16^{th}, 19^{th}, 21^{st}, and 24^{th} harmonic signals are located at a frequency of exactly 100 MHz. While the larger harmonic orders still land at the same exact frequency, their signal power level is heavily attenuated by the reconstruction filter. In addition, the 2^{nd} harmonic is located at 200 MHz, which has corresponding alias frequencies at the 3^{rd}, 7^{th}, 8^{th}, 12^{th}, 13^{th}, 17^{th}, 18^{th}, 22^{nd}, and 23^{rd} harmonics. Unfortunately, a significant number of harmonic signals also alias back to DC or 0 MHz, but these are filtered out by the DC block attached to the Spectrum Analyzer front end. The receiver of a Spectrum Analyzer can be damaged by a DC signal [12]; therefore, a DC blocking capacitor is often employed to prevent any unexpected damage while measurements are performed.

Listing 2.1: Perl sub-routine for Calculating Harmonic and Alias Frequencies

```
#-------------------------------------------------------------------------------
# Calculate Harmonic Frequencies:
#-------------------------------------------------------------------------------
my (@fharmAoA);
my ($N) = 40;
for (my $i=1; $i <= $N; $i++) {
    my ($tmp) = $fin*$i;
    push(@fharmAoA,[$i,$tmp]);
}

#-------------------------------------------------------------------------------
# Calculate Locations of Alias and Harmonic Frequencies:
#-------------------------------------------------------------------------------
my (@flocAoA);
my ($fs) = 500e6; # Sample Frequency
my ($fnyq) = $fs/2; # Nyquist Frequency
my ($floc) = 0;
for (my $j=0; $j < scalar(@fharmAoA); $j++) {
    my ($fharm) = $fharmAoA[$j][1]; # Harmonic Frequency
    my ($fratio) = $fharm/$fnyq; # Ratio of Harmonic to Nyquist Frequency
    my ($zone) = floor($fratio); # Nyquist Zone
    my ($zonecheck) = $zone % 2; # Determine if Nyquist Zone is Odd (0) or Even (1)
    my ($fodd) = $fharm % $fnyq; # Alias Frequency (odd zone)
    my ($feven) = $fnyq - $fodd; # Alias Frequency (even zone)

    if ($zonecheck == 0) {
        $floc = $fodd;
    } else {
        $floc = $feven;
    }
    push(@flocAoA,[$zone,$fharmAoA[$j][0],$fharm,$floc]);
}
```

2.4 Specifications

A summary of the signal source specifications are shown in Table 2.5. These specifications were determined by characterizing the signal source outputs of four General Purpose Instruments.

Table 2.5: Signal Source Specifications

PARAMETER	**TEST CONDITIONS**	**MIN**	**TYP**	**MAX**	**UNIT**
Frequency Domain Specifications					
SFDR	f$_{DAC_{DCLK}}$ = 250 MHz, f$_{DAC_{CLK}}$ = 500 MHz, Single Tone, Sine Wave Power Level -6 dBFS, First Nyquist Zone $< \frac{F_{DAC_{DCLK}}}{2}$	58.00	68.49	76.06	dBc
SNR	f$_{DAC_{DCLK}}$ = 250 MHz, f$_{DAC_{CLK}}$ = 500 MHz, Single Tone, Sine Wave Power Level -6 dBFS	99.50	101.02	102.06	dBc
IMD$_3$	f$_{DAC_{DCLK}}$ = 250 MHz, f$_{DAC_{CLK}}$ = 500 MHz, f$_1$ = 107.7 MHz, f$_2$ = 107.9 MHz, Power level of each tone -6 dBFS		68.16		dBc
TOI	f$_{DAC_{DCLK}}$ = 250 MHz, f$_{DAC_{CLK}}$ = 500 MHz, f$_1$ = 107.7 MHz, f$_2$ = 107.9 MHz, Power level of each tone -6 dBFS		26.58		dB
Time Domain Specifications					
Period	f$_{DAC_{DCLK}}$ = 250 MHz, f$_{DAC_{CLK}}$ = 500 MHz, Sine Wave Power Level 0 dBFS	1e6		8	ns
Frequency	f$_{DAC_{DCLK}}$ = 250 MHz, f$_{DAC_{CLK}}$ = 500 MHz, Sine Waveform Power Level 0 dBFS	0.001		125	MHz
	f$_{DAC_{DCLK}}$ = 250 MHz, f$_{DAC_{CLK}}$ = 500 MHz, Square Waveform	0.001		25	MHz
	f$_{DAC_{DCLK}}$ = 250 MHz, f$_{DAC_{CLK}}$ = 500 MHz, Ramp Waveform	0.001		10	MHz
Amplitude	f$_{DAC_{DCLK}}$ = 250 MHz, f$_{DAC_{CLK}}$ = 500 MHz, Sine Wave Power Level 0 dBFS	-300		300	mV
Duty Cycle	f$_{DAC_{DCLK}}$ = 250 MHz, f$_{DAC_{CLK}}$ = 500 MHz, Sine Wave Power Level 0 dBFS		50		%

Chapter 3

Signal Analyzer

A receiver is a common component of a measurement setup and is used to characterize the performance of a DUT. Several types of test and measurement equipment act as a receiver (or signal analyzer) including:

- Spectrum Analyzer
- Network Analyzer
- FFT Analyzer
- Oscilloscope

The AsAP processor is well-suited for signal analysis in both the frequency domain and the time domain. Frequency domain analysis is performed using a fast Fourier transform (FFT) hardware accelerator [14]. The AsAP can perform an FFT ranging from 16 points to 4096 points. The General Purpose Instrument provides a high-speed analog-to-digital converter (ADC) along with a field programmable gate array to demonstrate the signal analysis capabilities of the AsAP DSP processor.

The requirements used for the signal analyzer design are covered in Section 3.1. Section 3.2 describes the design of the signal analyzer and each component in the signal path. The methods used to verify the performance of the signal analyzer are describe in Section 3.3. And finally the performance of the signal analyzer is summarized in Section 3.4.

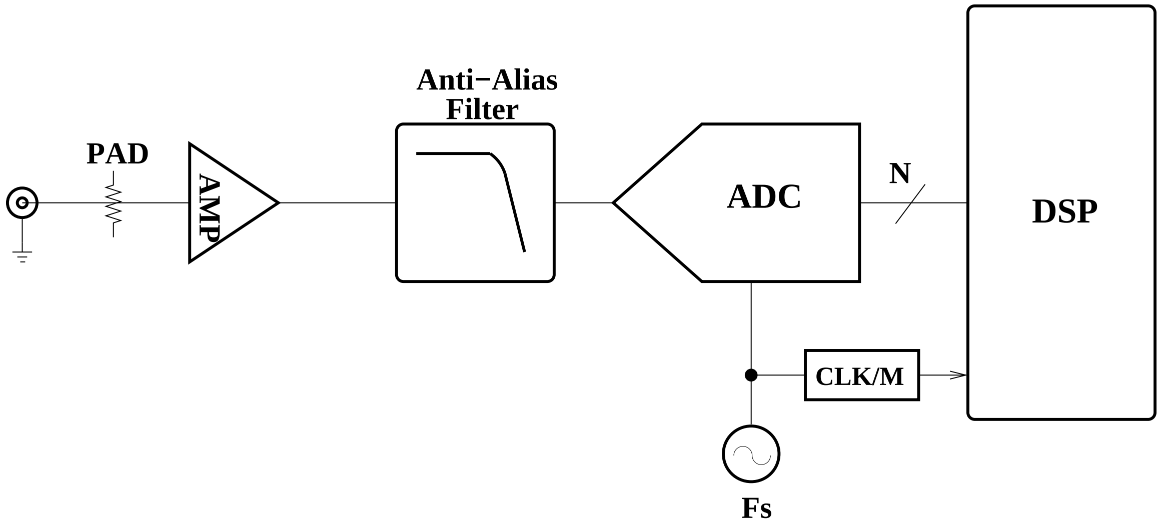

Figure 3.1: Common Signal Analyzer Block Diagram

Figure 3.1 shows the block diagram of a common receiver used in several types of test and measurement equipment.

3.1 Requirements

A signal analyzer can have a noticeable impact on the measurement results of a DUT. For example, a signal which is either too large, or has an incorrect offset level, can clip, resulting in an incomplete waveform in the time domain and additional spurious signals in the frequency domain. As such, it is important to understand the capabilities of the signal analyzer when characterizing a DUT. The following parameters are typically specified for signal analyzers by test and measurement equipment manufacturers, and were addressed by the General Purpose Instrument signal analyzer:

- Bandwidth
- Dynamic Range
- Distortion
- Accuracy

The signal analyzer of the General Purpose Instrument is intended to analyze waveforms in the frequency range of DC to 120 MHz. Given the wide range of applications in this frequency range, and the desire to digitize multi-tone waveforms, a wide bandwidth filter is required. The design complexity of the anti-alias filter will be determined by the sample rate of the ADC. The goal is to provide the maximum amount of signal bandwidth (B) without violating the Nyquist sampling theorem. As the filter's cut-off frequency approaches the Nyquist frequency ($\frac{F_s}{2}$), the cost and complexity will increase. One possible solution is to oversample the ADC by a factor of two and set the filter's cut-off frequency to $\frac{F_s}{4}$. The benefit of this type of architecture is that common and inexpensive inductors and capacitors can be used along with a filter topology that is simple to design and debug.

Some signal analysis applications may require signals that are closely spaced in the frequency domain. These applications require excellent dynamic range performance. Dynamic range is affected by several factors including noise present in the signal path, filter cut-off frequency, and amplifier performance. The measurement board of the General Purpose Instrument employs board level shields to enclose the signal analyzer circuitry. The board level shields help minimize the effects of unwanted signals, such as power supply switching frequencies and harmonic frequencies of the various clock sources on the board. The use of an oversampled system for the signal analyzer design should also improve the dynamic range by further attenuating alias frequencies of the sampled system.

3.2 Design

The General Purpose Instrument signal analyzer was designed to operate in the first Nyquist zone with a signal bandwidth of 120 MHz. Figure 3.2 shows a high-level block diagram of the signal analyzer.

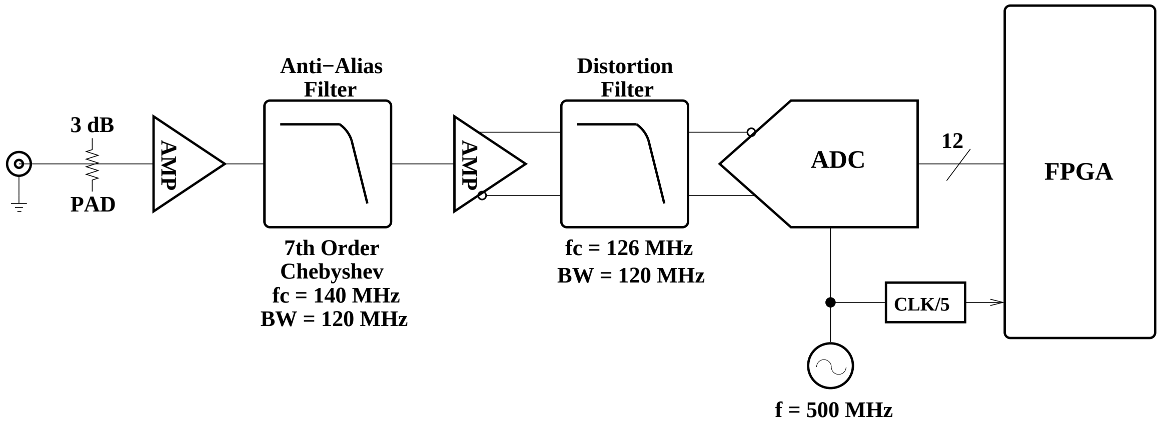

Figure 3.2: Signal Analyzer High-Level Block Diagram

The main building blocks of the signal analyzer are:

- Wideband, fixed gain operational amplifier
- Passive low-pass anti-alias filter
- Wideband, low-noise, low-distortion, differential operational amplifier
- High-speed analog-to-digital converter
- Digital signal processing, implemented in an FPGA

Each of the building blocks listed above is described in Subsections 3.2.1 to 3.2.8.

3.2.1 Signal Analyzer IF Design

The design of the General Purpose Instrument signal analyzer began with an investigation of the contribution of each component in the signal analyzer intermediate frequency (IF) chain. The goal of this investigation was to determine the power level at the input of the high-speed ADC given an input impedance of 200 Ω and a peak-to-peak voltage of 2 V_{pp}. An Excel spreadsheet was used to determine the following parameters at each component of the signal analyzer IF chain.

- CW Signal Level
- System Gain
- System Noise Figure

The gain, noise figure (NF), and input TOI contribution for each component in the signal analyzer IF chain were entered into the spreadsheet. An initial input power of −10 dBm was assumed. A summary of the input values and associated results are shown in Table 3.1. The spreadsheet uses macros to generate two plots. Figure 3.3 shows a plot of the gain, noise figure, and TOI. Figure 3.4 shows a plot of the excess noise and TOI. The values in Table 3.1 were determined by monitoring the data in Figures 3.3 and 3.4. In Figure 3.3, the goal was to adjust the parameters such that the noise figure data does not intersect with the gain data. In Figure 3.4, the goal was to adjust the parameters such that the vertical bars are approximately centered around the noise figure of the ADC (18.98980 dB) represented by the dotted horizontal line.

The values listed in Table 3.1 were used to design the signal analyzer. Based on Table 3.1, the signal power level at the high-speed ADC is +3.00 dBm. The signal analyzer IF chain has a system gain of +13 dB.

BLOCK NAME	**TLA**	**Gain (dB)**	**NF (dB)**	**Input TOI (dBm)**	**CW Signal Level (dBm)**	**System Gain (dB)**	**System NF (dB)**
Input	Input				-10.00		
-3dB PAD	VA1	-3.00	3.00		-13.00	-3.00	3.00
Amp (THS4302)	AmpIn	14.00	16.00	30.00	1.00	11.00	-11.00
-1dB PAD	Pad	-1.00	1.00		0.00	10.00	-10.00
Anti-Alias Filter	AAF	-2.40	2.40		-2.40	7.60	-7.60
-1dB PAD	Pad	-1.00	1.00		-3.40	6.60	-6.60
Amp (THS4509)	DiffAmp	8.00	17.10	38.00	4.60	14.60	10.50
Distortion Filter	FLTR	-1.60	1.60		3.00	13.00	10.51
TI ADS5463	ADC		18.98980	44.80	3.00	13.00	11.81

Table 3.1: Estimated signal analyzer IF performance

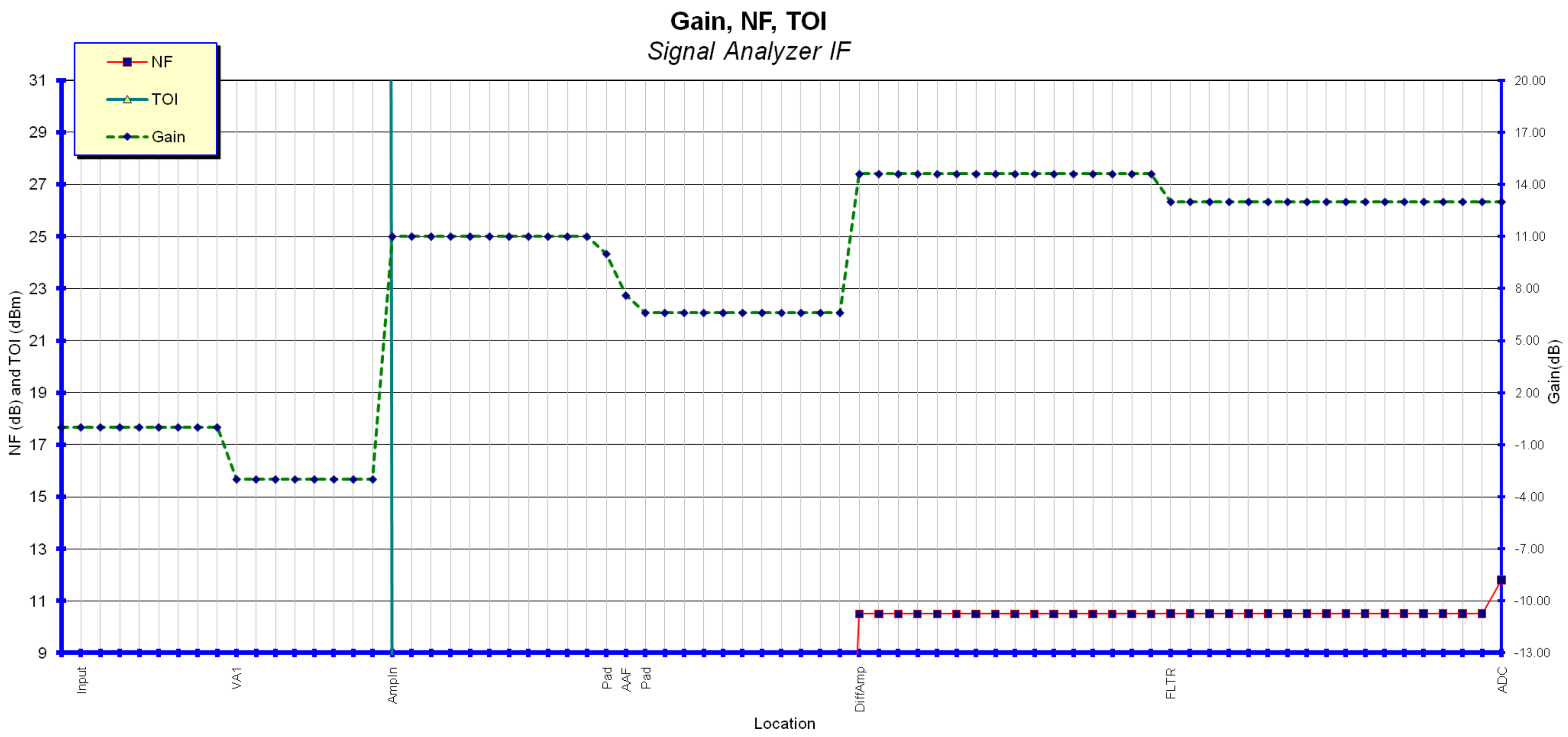

Figure 3.3: Signal Analyzer Gain, Noise Figure, and TOI Graph

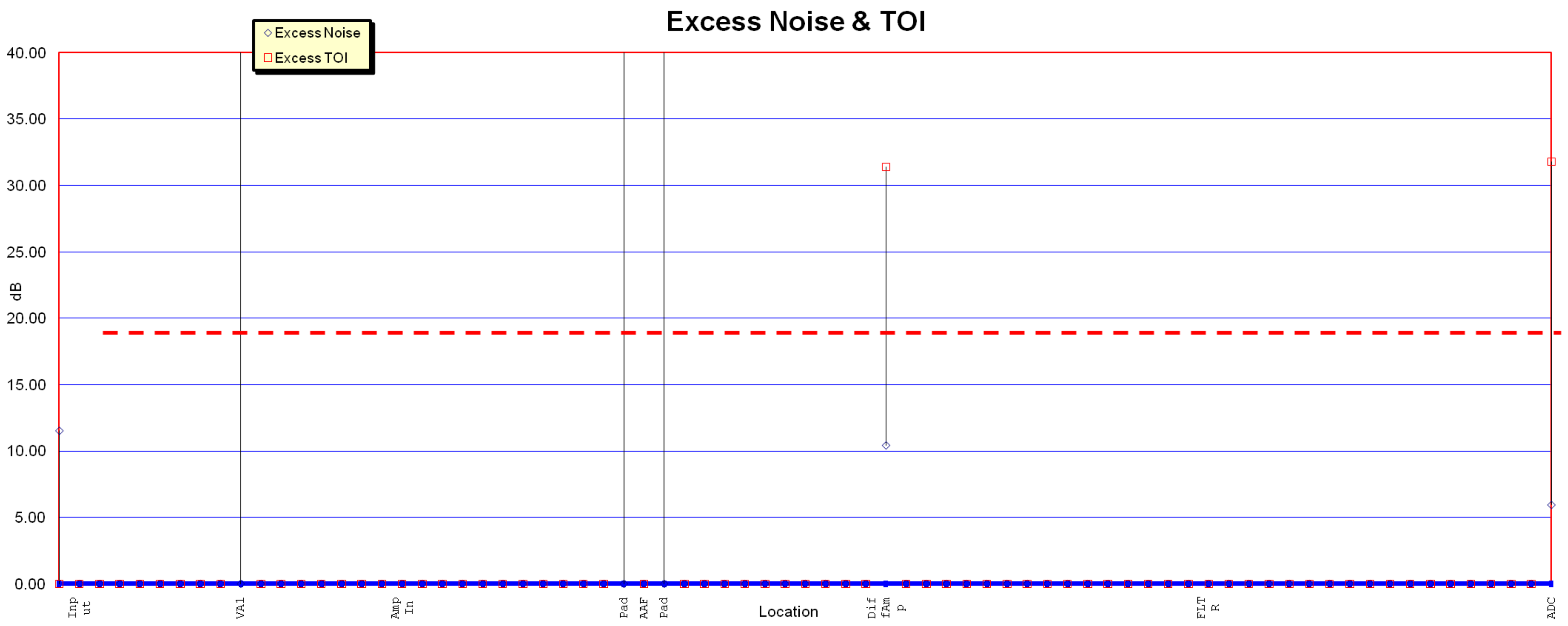

Figure 3.4: Signal Analyzer Excess Noise and TOI Graph

3.2.2 Low Noise Amplifier Design

A Texas Instruments THS4302 wideband, fixed gain amplifier with a signal bandwidth of 2.4 GHz is used to drive the input signal into the anti-alias filter [15]. The baseband signal analyzer front-end was designed to terminate the input signal into a 50 Ω impedance and amplify the signal by +11 dB. A −3 dB PAD, or attenuator, precedes the THS4302 amplifier, which was internally configured to have a gain of +14 dB. The attenuator was designed using a pi-pad configuration with discrete resistors. Figure 3.5 shows the THS4302 fixed-gain amplifier and attenuator circuits of the baseband signal analyzer.

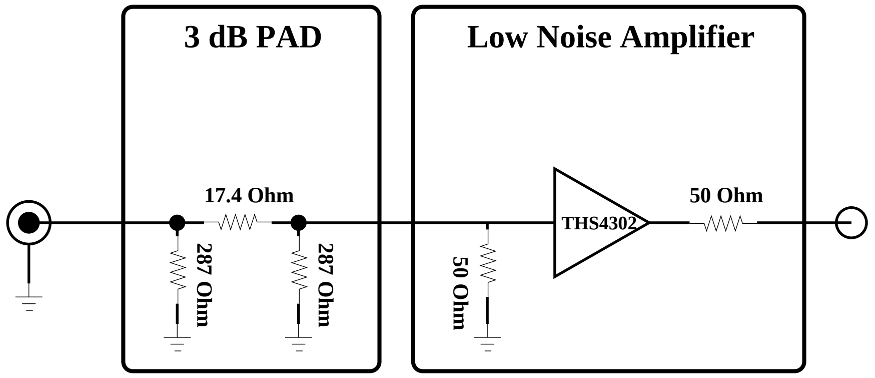

Figure 3.5: Amplifier Schematic

3.2.3 Anti-Alias Filter Design

In general, wide-band passive filters are difficult to implement with sufficient ripple and stop-band performance [7]. The role of a anti-alias filter is to attenuate harmonic and alias frequencies to a sufficient power level before they fold back into the pass band of the filter. In a typical sampled system operating in the first Nyquist zone, the cut-off frequency of the anti-alias filter would be set to a frequency slightly less than $\frac{F_s}{2}$. As the filter's cut-off frequency approaches the Nyquist frequency the steepness of the transition band increases. Figure 3.6(a) highlights the shape of the anti-alias filter when sampling at twice the Nyquist frequency.

The baseband signal analyzer takes advantage of oversampling to greatly simplify the implementation of the anti-alias filter. Figure 3.6(b) shows the effects of oversampling on the anti-alias filter. The high-speed ADC was oversampled by a factor of two, which allowed a −3 dB frequency well within the first Nyquist zone. As a result, the signal analyzer can cleanly pass sinusoid waveforms up to 120 MHz with relative ease.

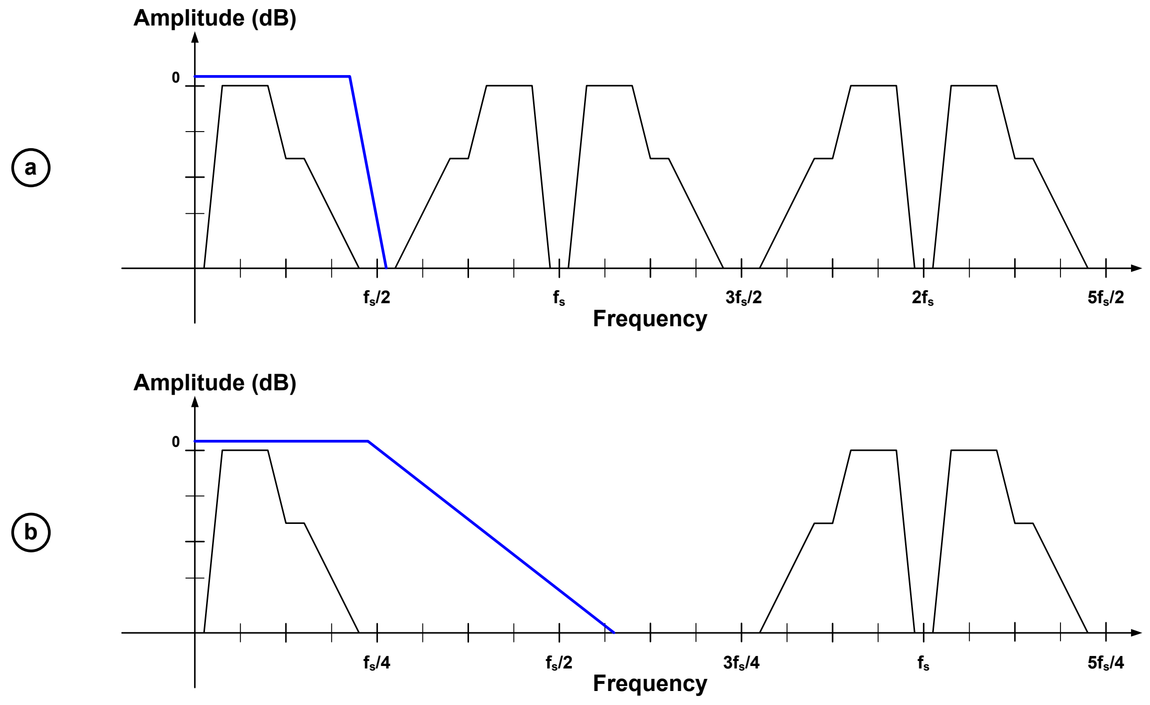

Figure 3.6: (a) Example of Nyquist sampling and the requirements for the anti-alias filter. (b) Example of anti-alias filter constraint relaxation as a result of oversampling by 2.

The anti-alias filter was designed using Agilent Technologies' Genesys Filter Synthesis software. The first step in designing the filter involved setting the desired specifications including:

- -3 dB frequency: 130 MHz
- Passband ripple: 0.5 dB
- Stopband attenuation: 60 dB
- Input impedance: 50 Ω
- Output impedance: 50 Ω

The second step was to select a filter type, filter shape, and filter topology. The baseband signal analyzer uses a 7^{th}-Order Chebyshev low-pass differential filter topology. Upon defining the filter specifications, the Genesys Filter Synthesis software generated the filter schematic and simulated frequency response shown in Figures 3.7 and 3.8, respectively. The simulated frequency response was evaluated to ensure the desired specifications would be met by the filter design.

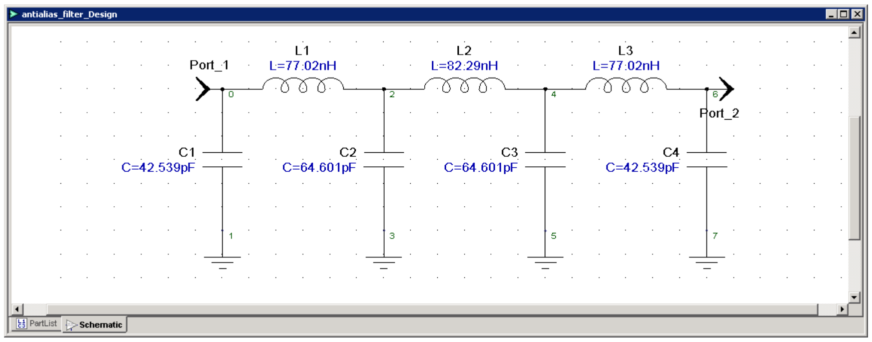

Figure 3.7: Genesys Filter Synthesis 7th-Order Chebyshev low-pass filter schematic

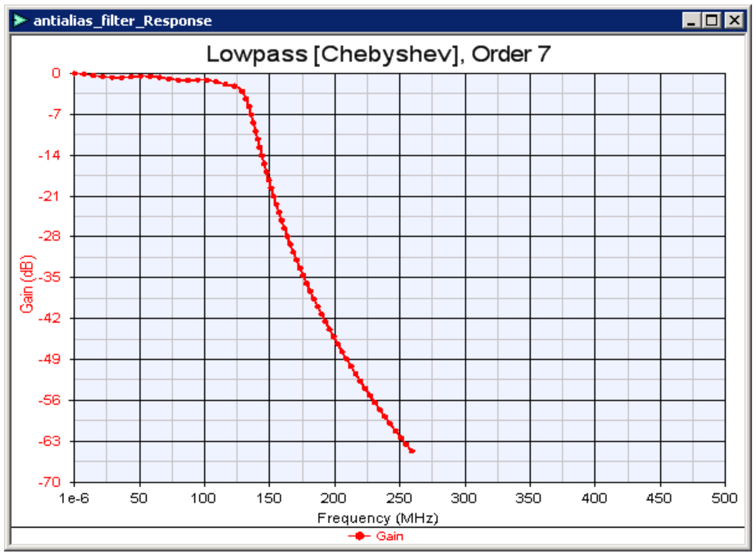

Figure 3.8: Genesys Filter Synthesis simulated frequency response for 7th-Order Chebyshev low-pass filter

The inductor and capacitor values shown in the Figure 3.7 were generated by the Genesys Filter Synthesis software, and were used as a starting point to select practical component values. When choosing component values for the anti-alias filter, it was necessary to use multiple components to achieve the correct values. In the case of inductance, the inductors were placed in series; in the case of capacitance, the capacitors were placed in parallel.

Reference Designator	Calculated	Practical	Series/Parallel
L1	77.02 nH	71 nH	22 nH+27 nH+22 nH
L2	82.29 nH	76 nH	27 nH+22 nH+27 nH
L3	77.02 nH	71 nH	22 nH+27 nH+22 nH
C1	42.539 pF	39.6 pF	3.6 pF+36 pF
C2	64.601 pF	59.9 pF	3.9 pF+56 pF
C3	64.601 pF	59.9 pF	3.9 pF+56 pF
C4	42.539 pF	39.6 pF	3.6 pF+36 pF

Table 3.2: A comparison of calculated and practical component values for the anti-alias filter

The schematic shown in Figure 3.9 represents the single-ended low-pass filter that was simulated using LTSpice.

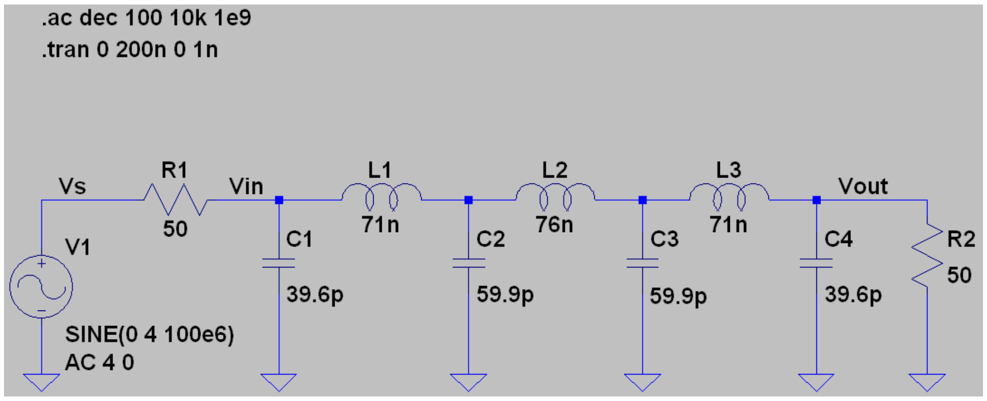

Figure 3.9: LTSpice 7th-Order Chebyshev differential low-pass filter schematic

The simulated frequency response of the Chebyshev filter, shown in Figure 3.10, has a cut-off frequency of 141 MHz, which is different from that specified in the Genesys Filter Synthesis software. However, the increased cut-off frequency provides for more signal bandwidth in the anti-alias filter.

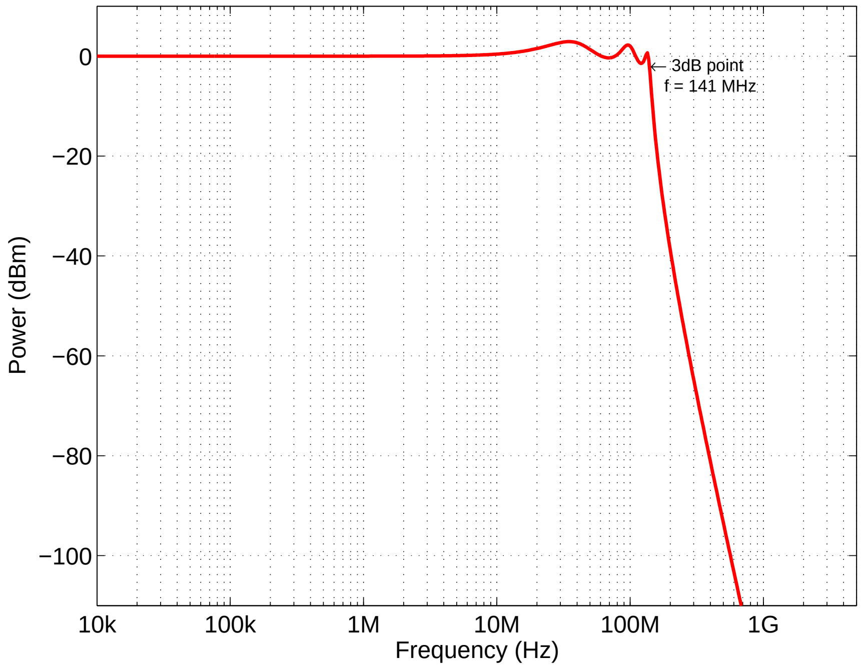

Figure 3.10: LTSpice 7th-Order Chebyshev low-pass filter simulated frequency response. The measured performance of the 7th-Order Chebyshev low-pass filter is shown in Figure 3.26.

Table 3.3 outlines the estimated performance of the 7th-Order Chebyshev low-pass anti-alias filter.

PARAMETER	**VALUE**
-3 dB Frequency	141 MHz
Bandwidth	120 MHz
Passband Ripple	0.5 dB
Signal Attenuation	2.4 dB
Stopband Attenuation	60 dB

Table 3.3: Estimated anti-alias filter specifications

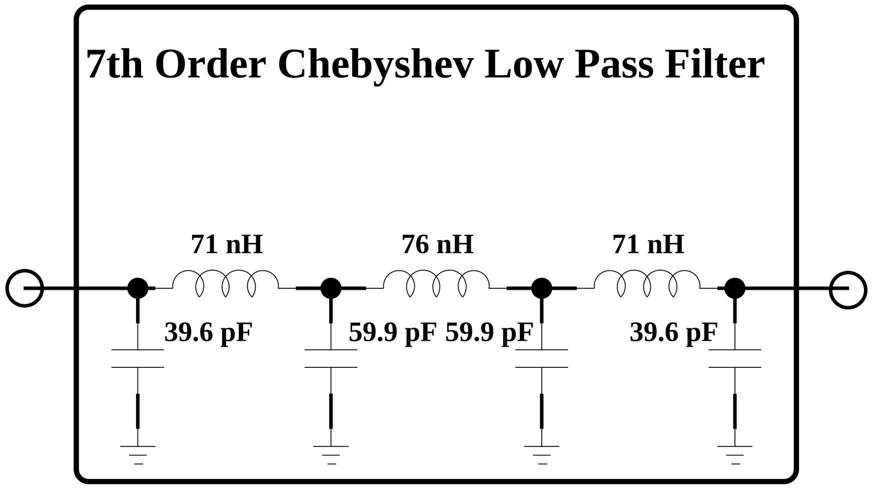

Figure 3.11: Detailed schematic of the signal analyzer anti-alias filter

The final implementation of the anti-alias filter used in the signal analyzer is shown in Figure 3.11. The bill of materials for the anti-alias filter is shown in Table 3.4.

Reference Designator	**Manufacturer**	**Part Number**	**Description**
C16, C17	Murata Electronics	GRM1885C1H3R6CZ01D	3.6 pF, 50 V Ceramic Capacitor
C143, C144	Murata Electronics	GRM1885C1H3R9CZ01D	3.9 pF, 50 V Ceramic Capacitor
C21, C22	Murata Electronics	GRM1885C1H360JA01D	36 pF, 50 V Ceramic Capacitor
C26, C27	AVX Corporation	06035A560JAT2A	56 pF, 50 V Ceramic Capacitor
L121, L122, L123, L124, L125	Coilcraft	0603CS-22NXJL	22 nH, 700 mA Ceramic Chip Inductor
L71, L72, L73, L74	Coilcraft	0603CS-27NXJL	27 nH, 600 mA Ceramic Chip Inductor

Table 3.4: Anti-alias filter bill of materials

3.2.4 Differential Amplifier Design

A Texas Instruments THS4509 wideband, low-noise, low-distortion fully-differential amplifier with a signal bandwidth of 1.9 GHz is used to drive the high-speed ADC input [16]. The amplifier is used to convert the anti-alias filter output signal (AIN) from single-ended to differential. The THS4509 is configured to amplify the anti-alias filter output signal by a gain of +8 dB. Figure 3.12 shows the THS4509 fully-differential amplifier and the high-speed ADC input filter circuits of the baseband signal analyzer.

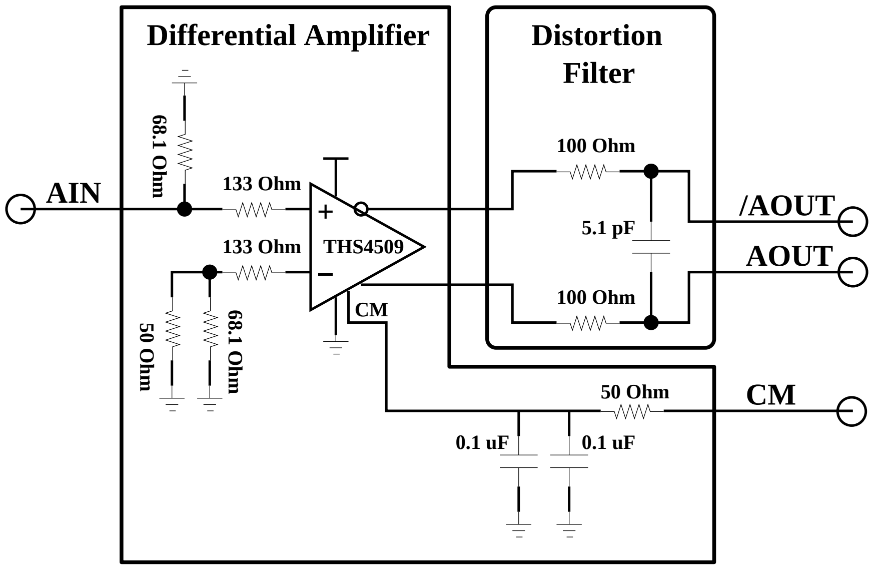

Figure 3.12: Differential Amplifier Schematic

The common mode (CM) input is driven by the high-speed ADC and is used to set the common mode of the differential amplifier output to +2.4 V. The high-speed ADC input signal will swing at most 2.2 V_{pp} about the +2.4 V common mode voltage. The output resistors and 5.1 pF capacitor create a low-pass filter, also known as a distortion or interface filter. The role of the distortion filter is to provide adequate rejection of harmonic aliasing and noise folding in the second Nyquist zone, which starts at 380 MHz ($F_s - F_{max}$). The distortion filter was designed using a single-pole, RC filter topology, with a −3 dB frequency of 123 MHz. Figure 3.13 shows the simulated frequency response of the distortion filter.

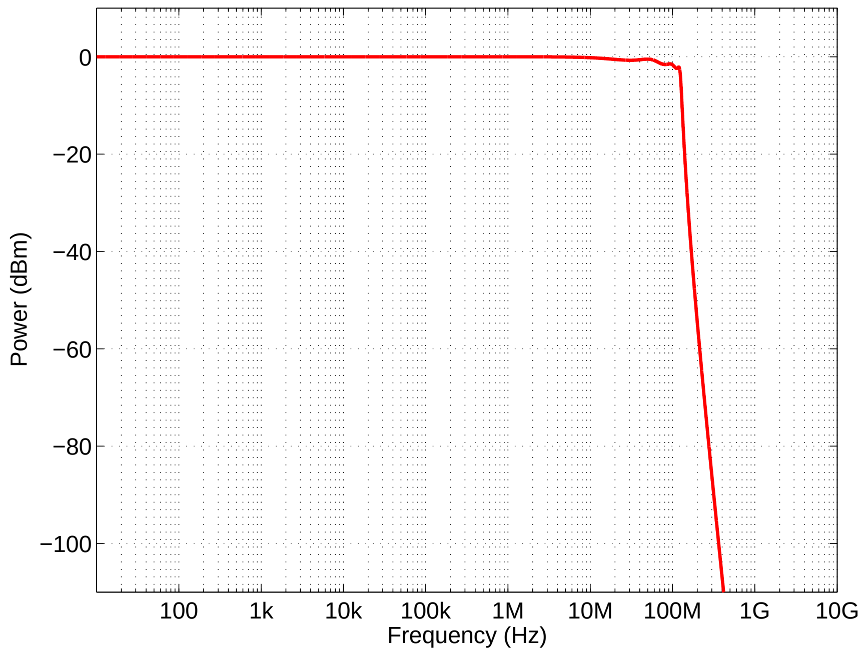

Figure 3.13: Tina-TI differential low-pass distortion filter simulated frequency response. The measured performance of the differential low-pass distortion filter is shown in Figure 3.33.

3.2.5 High-Speed Analog-to-Digital Converter

A Texas Instruments ADS5463 12-bit, 500 MS/s analog-to-digital converter is used to digitize the analog waveforms for the baseband signal analyzer. The ADC is sampled at 500 MS/s, the same rate as the DAC, which allowed the signal source and signal analyzer DSP designs to coexist in the same FPGA. This capability is especially important when the signal source is used to stimulate a DUT and measure its performance with the signal analyzer.

The high-speed ADC digital interface is made up of a single out-of-range signal and 12-bits of double data rate data in bipolar offset binary (BOB) format. The digital interface of the ADC operates at 250 MHz, which corresponds to a data rate of 500 Mb/s for each data bit. The Xilinx Virtex-5 SX50T FPGA core logic is only capable of operating at frequencies up to 450 MHz, so a straightforward 12-bit data path could not be implemented. Instead, the 12-bit ADC data was first sign-extended to 16-bits and then extended to 128-bits in order to use the built-in input serdes (ISERDES). The ISERDES facilitate higher external data rates, while keeping the internal data bus at a more manageable rate.

In the case of the signal analyzer, the internal data rate is operating at 250 Mb/s divided by 4, which is equivalent to 62.5 Mb/s. The ISERDES are used in an 1:8 DDR configuration, which requires a high-speed clock of 250 MHz and a low-speed clock of 62.5 MHz. The internal data is running at a single data rate, and the external data is running at a double data rate.

The differential analog input of the high-speed ADC was inverted as a result of the differential operational amplifier package pinout. The goal of the signal analyzer PCB layout was to keep the signal path traces on the top layer therefore, avoiding the use of vias. As a result, the digitized data must also be inverted in the FPGA to achieve the correct representation of the signal data.

The DSP sub-system requires the high-speed ADC data to be represented in bipolar two's complement (BTC) format. The high-speed ADC data is converted to BTC at the same time as the inversion is performed. Typically, when converting from BOB to BTC coded formats, the most-significant bit (MSB) of the data word would be inverted. However, since the high-speed ADC data must be inverted, the least significant bits (LSB) of the data word are inverted instead of the MSB data bit. Table 3.5 highlights the relationship between the bipolar offset binary and bipolar two's complement coded formats relative to full scale (FS) high-speed ADC output data. When converting between coded formats it is important to know that the digital zero *0000* corresponds to the bipolar zero (BPZ).

Table 3.5: Bipolar Offset Binary to Bipolar Two's Complement Conversion

MNEMONIC	**DIGITAL CODE**	
	BOB	**BTC**
-FS	0000	1000
	0001	1001
	0010	1010
	0011	1011
$\frac{1}{2}$-FS	0100	1100
	0101	1101
	0110	1110
BPZ - 1V$_{LSB}$	0111	1111
BPZ	1000	0000
BPZ + 1V$_{LSB}$	1001	0001
	1010	0010
	1011	0011
$\frac{1}{2}$+FS	1100	0100
	1101	0101
	1110	0110
+FS	1111	0111

3.2.6 Digital Signal Processing

The digital signal processing sub-system of the signal analyzer is made up of a high-performance field programmable gate array and two AsAP digital signal processors. The DSP sub-system is responsible for digitizing and capturing analog signals from the high-speed ADC. The signal analyzer can digitize and transfer data to several locations including:

- 2-Mbit Block RAM
- 32-Mbit QDR-II SRAM
- AsAP DSPs
- FPGA logic

The current implementation of the signal analyzer design supports only the storing of signals to QDR-II SRAM and Block RAM. Figure 3.14 shows the major components of the DSP sub-system, and Figure 3.15 shows a detailed view of the FPGA data path.

Several key design areas were addressed during the development of the DSP sub-system including:

- High-speed ADC interface
- Waveform DC offset
- Waveform capture
- Multiple clock domains

Each of the design areas listed above is described in Subsections 3.2.6.1 to 3.2.6.2.

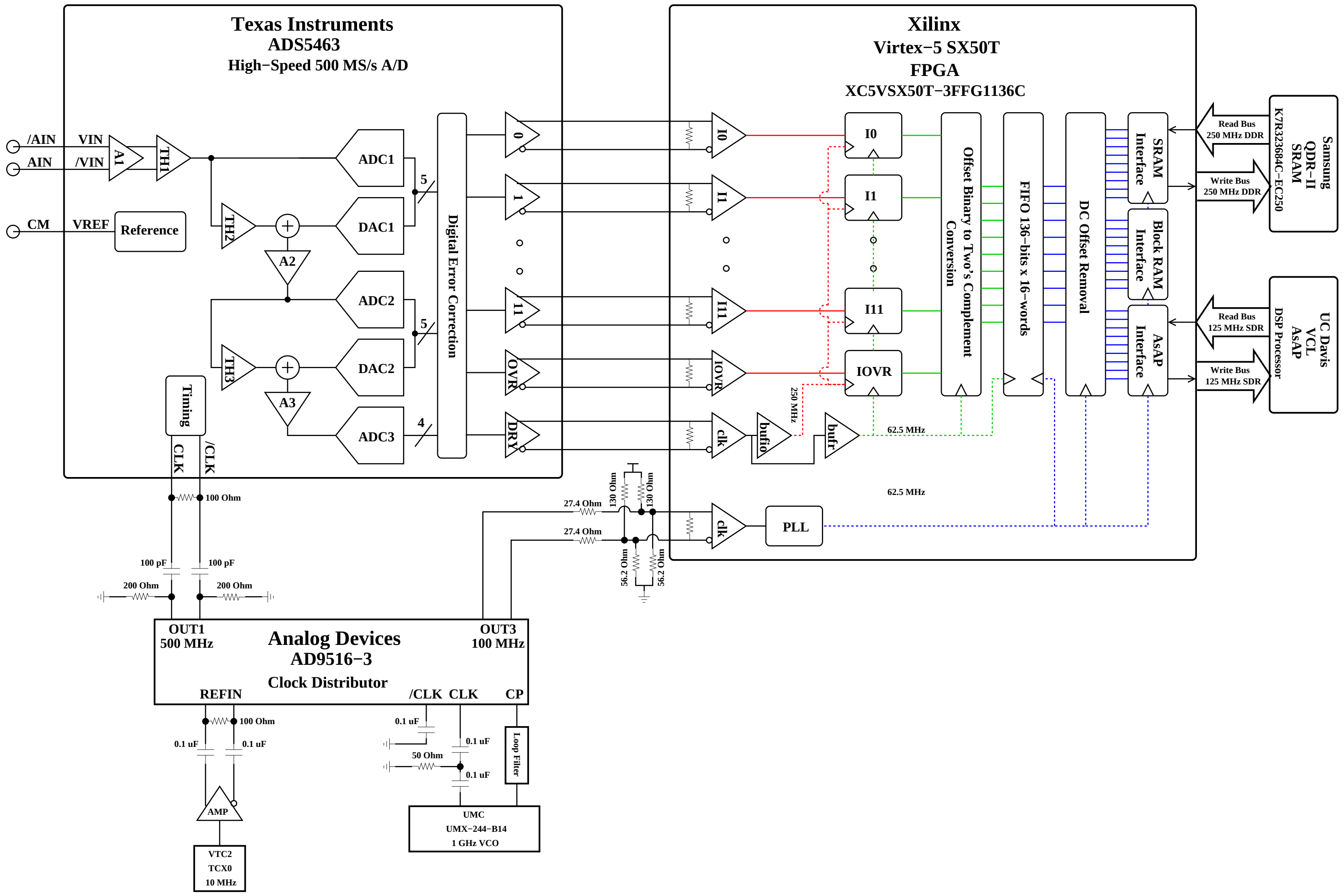

Figure 3.14: Baseband signal analyzer DSP block diagram

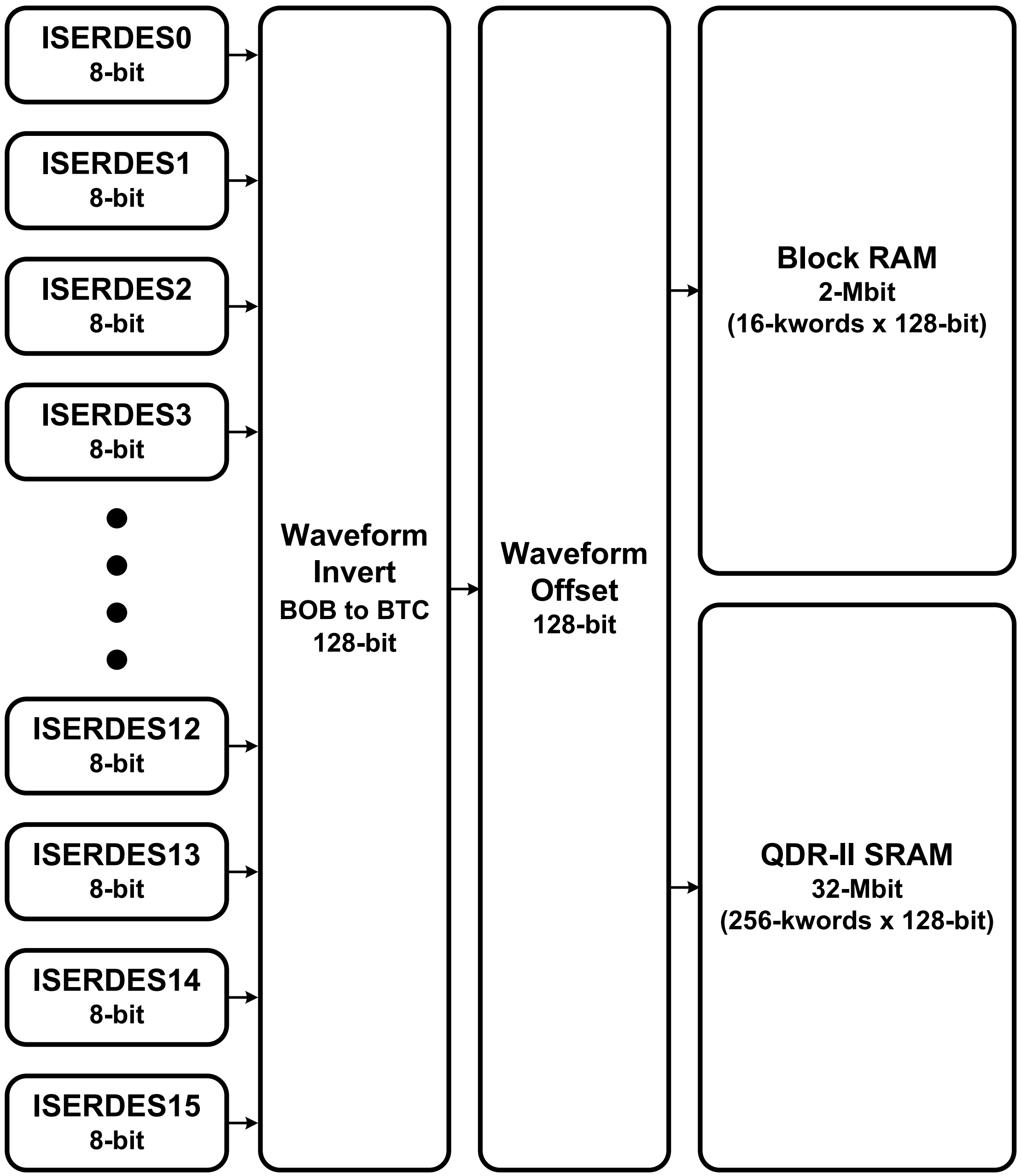

Figure 3.15: Baseband signal analyzer data path FPGA block diagram

3.2.6.1 Waveform Capture

The signal analyzer was designed to capture digitized waveforms from the high-speed ADC. The amount of storage required for a sinusoid waveform of frequency F_c can be calculated using Equation 3.1, where F_s is the sample frequency of the ADC and R is the resolution of the ADC.

$$NumBits = \frac{F_s}{F_c} \cdot R \tag{3.1}$$

The waveform capture sub-system uses a 128-bit data bus to transport data from the ADC sub-system to each data consumer. The signal analyzer will capture a minimum of sixteen 16-bit waveform samples. The internal data bus width is essentially made up of eight 16-bit data samples, which lends itself to polyphase or parallel DSP operations. The data bus operates at a clock rate of 62.5 MHz. The digitized data can be routed to one of two sources:

- 2 Mbits Block RAM
- 32 Mbits QDR-II SRAM

Block RAM The block RAM memory is capable of storing waveforms up to 128-kSamples, and is arranged as 16-kwords x 128-bits.

QDR-II SRAM The QDR-II SRAM memory is capable of storing waveforms up to 2-MSamples, and is arranged as 256-kwords x 128-bits.

3.2.6.2 Waveform DC Offset

The baseband signal analyzer allows the user to adjust the DC offset voltage of the digitized data. This feature is implemented using a DSP48 slice in the Xilinx Virtex-5 FPGA, and takes advantage of the properties of the two's complement numbering scheme. A DSP48 slice can perform a combination of 25-bit x 18-bit multiplies and 48-bit additions.

Offset Depending on the DUT, the input signal of the signal analyzer may require an offset voltage other than ground, or 0 Volts. A waveform signal can be offset by adding an offset value to the digitized waveform data. The baseband signal analyzer input typically swings about ground. An offset voltage is introduced to the waveform data by adding a two's complement 12-bit value, which represents every integer in the range -2^{12} to $\left(+2^{12}-1\right)$. In order to adjust the offset voltage such that the waveform signal swings about $\frac{V_{ampl}}{2}$, the waveform data can be summed with a value of $\left(2^{12}-1\right)$ or 0x07FF. The waveform signal is offset in the baseband signal analyzer by employing a bank of eight DSP48 slices configured as 17-bit adders, which provide a 16-bit result and a 1-bit overflow flag. The offset parameter is sign-extended to 16-bits before the add operation is performed. The same offset value is used for all eight DSP48 slices.

3.2.7 Clock Generation and Distribution

A key element of the baseband signal analyzer is the clock generation and distribution scheme. The successful analysis of arbitrary waveforms depends on synchronous high performance clocks. Several schemes were used to generate and distribute synchronous clocks throughout the DSP sub-system. Figure 3.16 describes the clock relationship between the various DSP components of the baseband signal analyzer.

3.2.7.1 High-Speed Clock Generation

An Analog Devices AD9516-3 14-output clock generator is responsible for generating all clocks for the baseband signal analyzer. A Universal Microwave Corp UMX Series 1 GHz voltage controlled oscillator is used to drive the AD9516-3 external RF clock input. Using the external 1 GHz VCO, the output frequency range of the AD9516-3 is 15.625 MHz to 1 GHz. The AD9516-3 clock generator is responsible for generating two clocks:

- 100 MHz data path FPGA clock
- 500 MHz high-speed ADC sample clock

Data Path FPGA Clock Generation The data path FPGA uses an internal phase-locked loop primitive to generate several clocks from the 100 MHz clock input. The PLL primitive generates the following clocks:

- 250 MHz high-speed clock
- 62.5 MHz internal core low-speed clock

The 62.5 MHz internal core low-speed clock is used to clock the waveform capture logic. The 250 MHz high-speed clock is used to clock the 32-Mbit QDR-II SRAM memory device and controller.

High-speed DAC Sample Clock Generation The high-speed ADC is sampled by the 500 MHz clock generated by the AD9516-3 clock generator. The high-speed ADC uses both edges of the clock for the data conversion process. In addition, the sample clock is used to generate the data ready signal.

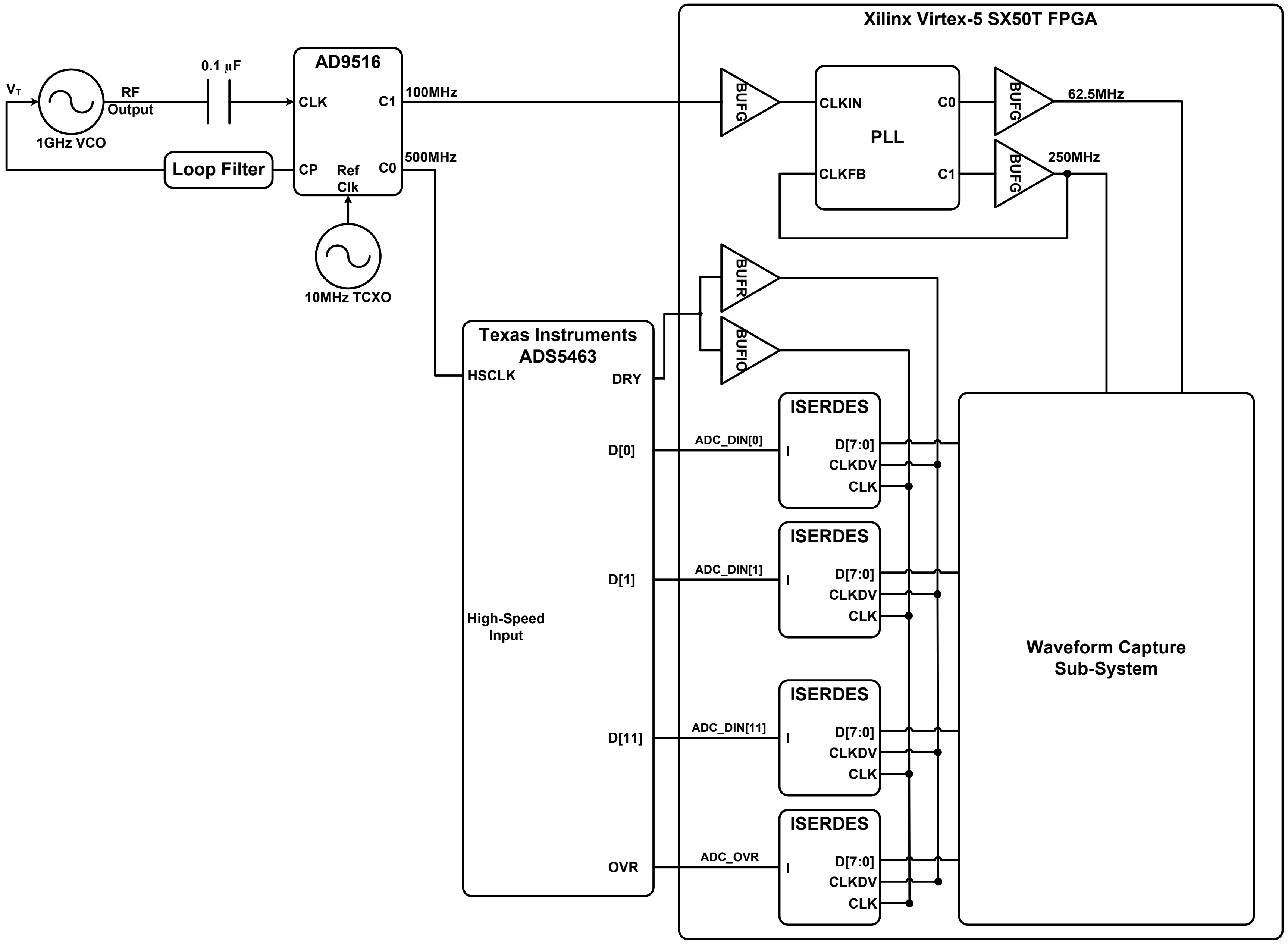

Figure 3.16: Signal Analyzer clock distribution and generation block diagram

3.2.7.2 High-Speed ADC Data Ready

The high-speed ADC generates a data ready (DRY) signal that operates at half the sample clock frequency; as such, it can be used as a half-rate DDR clock. The data ready signal is source-synchronous to the 12-bit data and over-range indicator outputs. The data path FPGA receives the data ready signal on a clock capable pin, and drives it into special clock buffers, BUFIO and BUFR, designed to clock the ISERDES devices of a Virtex-5 SX50T FPGA. The BUFIO clock buffer drives a dedicated clock net within the I/O column, which contains the ISERDES devices. The BUFR is a regional clock buffer capable of driving a dedicated clock net within a clock region. Unlike the BUFIO clock buffer, the BUFR clock buffer can drive both I/O logic and regular logic resources. In addition, the BUFR clock buffer is capable of dividing its input clock by an integer value between one and eight [17]. The data path FPGA uses the ISERDES devices in a 1:8 DDR mode, which requires a half-rate clock to drive its high-speed clock input and a divide-by-4 clock to drive its divided clock input. Figure 3.16 highlights the clock buffer configuration used by the data path FPGA.

3.2.8 Digitized Waveform Examples

A detailed block diagram of the baseband signal analyzer is shown in Figure 3.18. The signal analyzer was designed to digitize both CW and arbitrary waveforms, examples of which are listed below:

- A sawtooth signal (Figure 3.17).
- A CW signal operating at 100 MHz in the frequency domain (Figure 3.19).
- A CW signal operating at 100 MHz in the time domain (Figure 3.20).
- A three tone sine waveform operating at 80 MHz, 90 MHz, and 100 MHz in the frequency domain (Figure 3.21).
- A three tone sine waveform operating at 80 MHz, 90 MHz, and 100 MHz in the time domain (Figure 3.22).
- A ramp signal operating at 5 MHz in the time domain (Figure 3.24).
- A ramp signal operating at 5 MHz in the frequency domain (Figure 3.23).

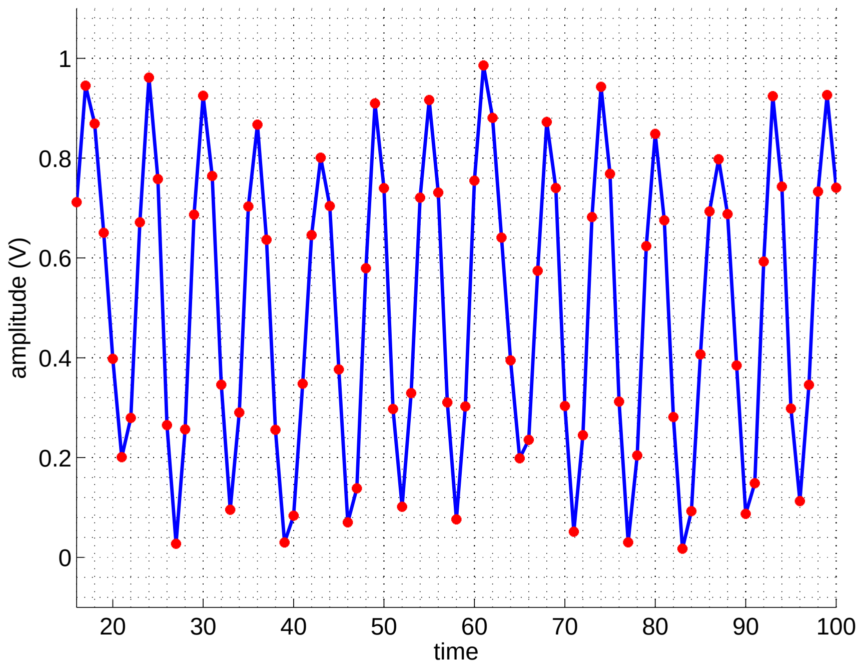

Figure 3.17: Sawtooth Waveform. Stimulated with a General Purpose Instrument Signal Source.

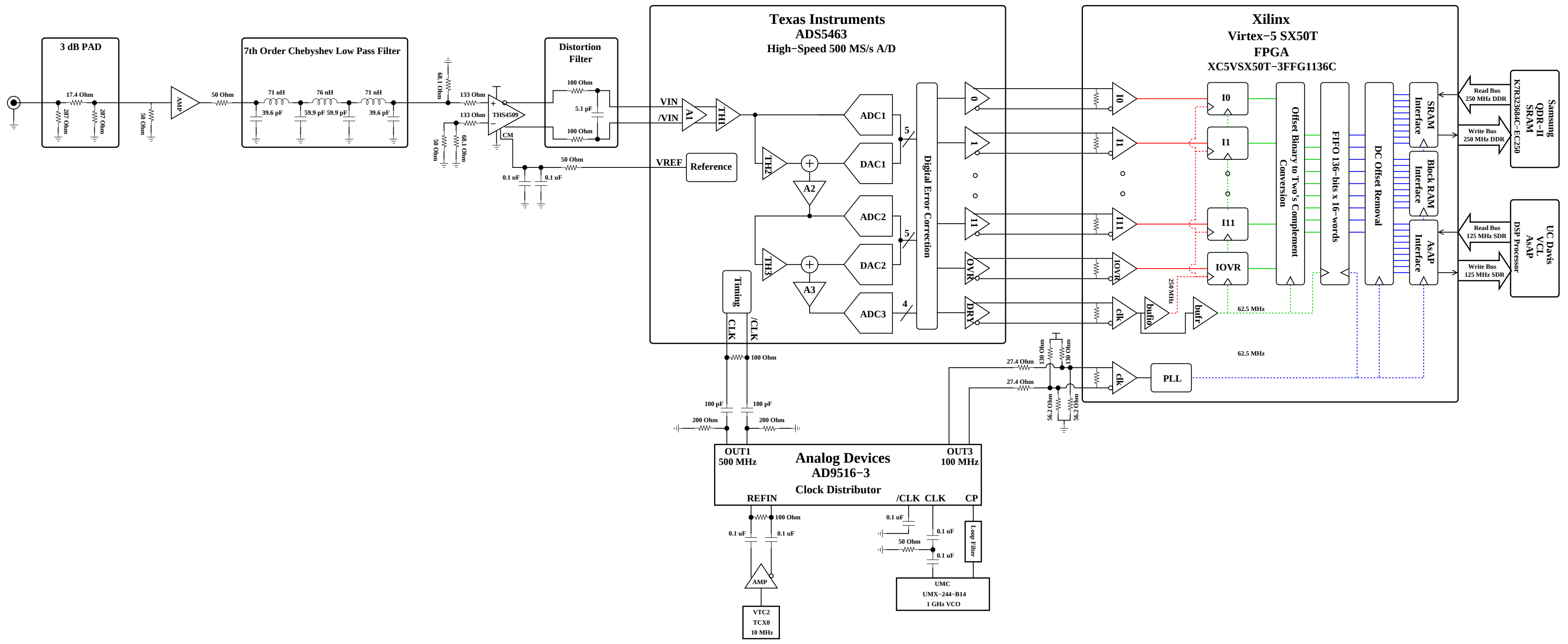

Figure 3.18: Detailed block diagram of baseband signal analyzer

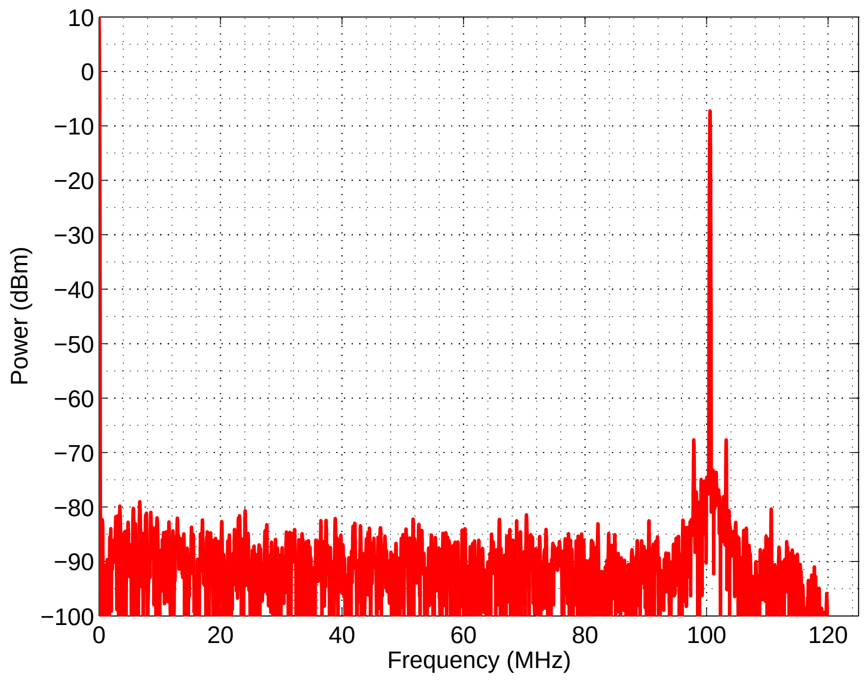

Figure 3.19: Single Tone Sine Wave at a frequency of 100 MHz with a power level of −5 dBm. Stimulated with an Anritsu MG3692A RF/Microwave Signal Generator.

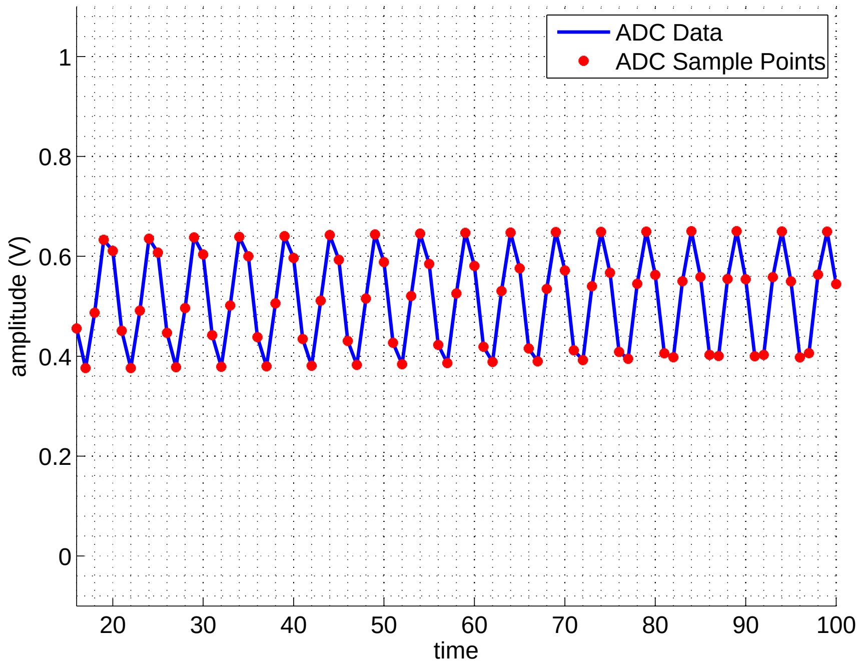

Figure 3.20: Sine Waveform at a frequency of 100 MHz with a power level of −5 dBm. Stimulated with an Anritsu MG3692A RF/Microwave Signal Generator.

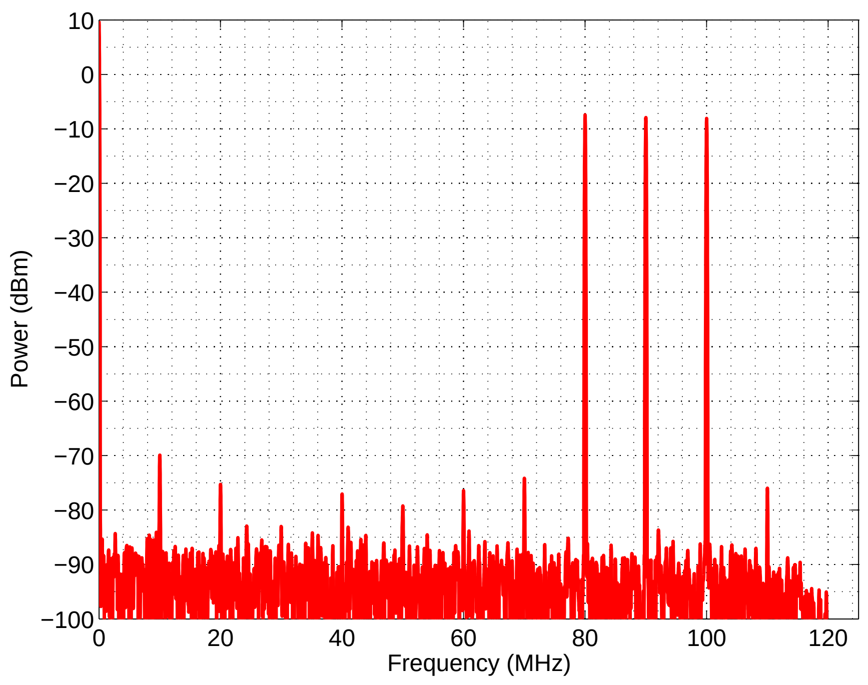

Figure 3.21: Three Tone Sine Wave at frequencies of 80 MHz, 90 MHz, and 100 MHz. Stimulated with a General Purpose Instrument Signal Source.

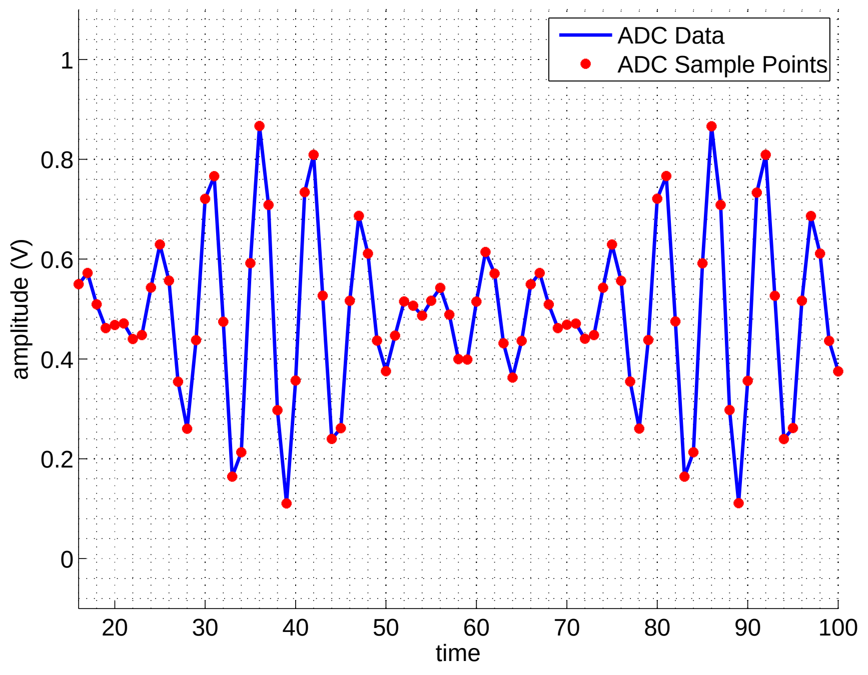

Figure 3.22: Three Tone Sine Wave at frequencies of 80 MHz, 90 MHz, and 100 MHz. Stimulated with a General Purpose Instrument Signal Source.

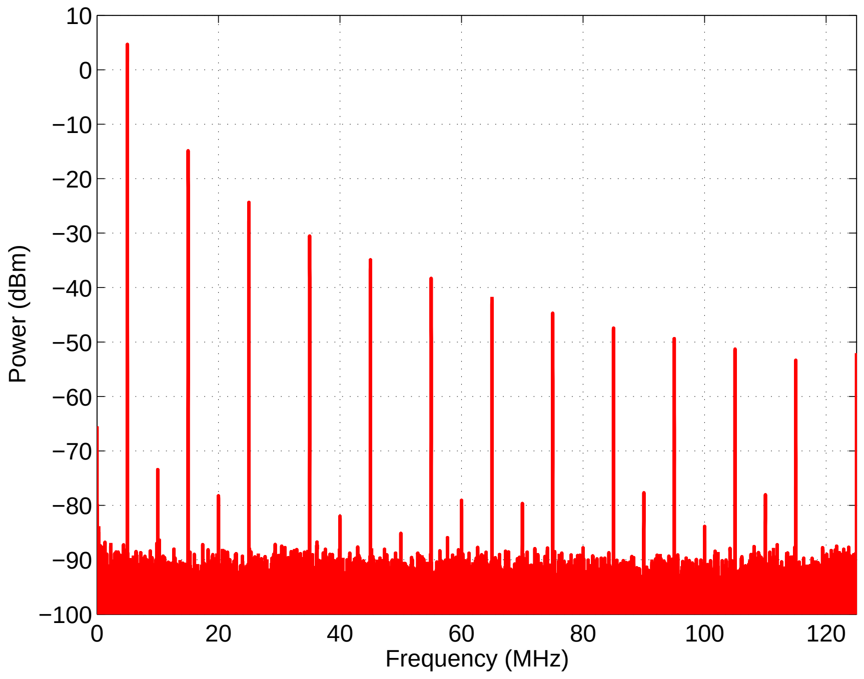

Figure 3.23: Ramp Waveform at a frequency of 5 MHz. Stimulated with a General Purpose Instrument Signal Source.

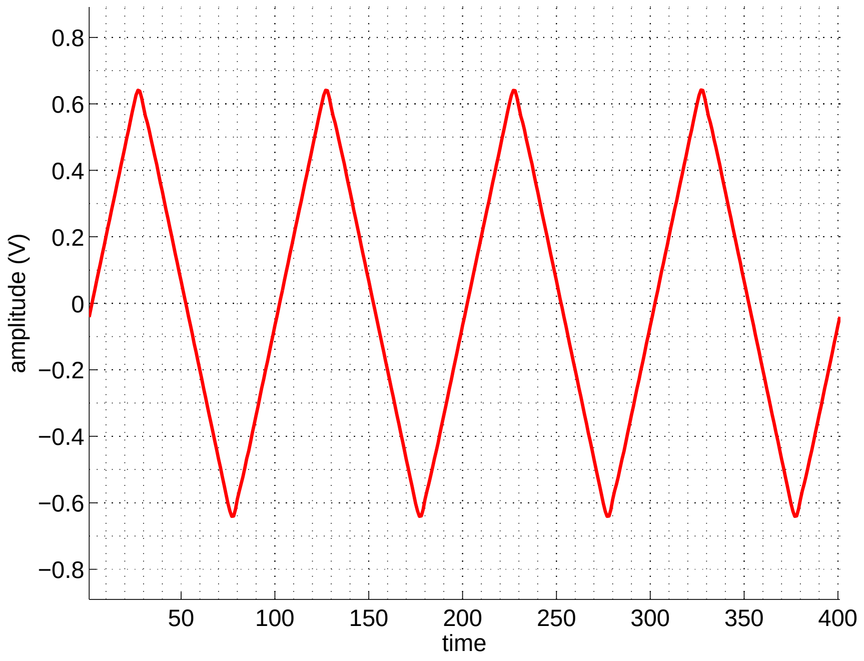

Figure 3.24: Ramp Waveform at a frequency of 5 MHz. Stimulated with a General Purpose Instrument Signal Source.

3.3 Verification

The General Purpose Instrument signal analyzer was designed to be used in test and measurement applications. However, before it could be utilized its performance had to be evaluated and shown to meet the intended specifications. The performance of the signal analyzer was evaluated in both the frequency and the time domain using an assortment of test and measurement equipment. In addition, the performance was evaluated across multiple boards.

3.3.1 Frequency Domain Measurements

The goal of the frequency domain measurements was to verify the quality of the signal analyzer when receiving both CW and arbitrary waveform signals. Receiver tests are typically performed using signal generators and arbitrary waveform generates to stimulate the receiver signal path and DSP sub-systems. However, test paths were provided in the baseband signal analyzer for characterizing the performance of individual signal path elements with a network analyzer. The signal analyzer frequency domain measurements were split up into two areas:

- Anti-alias filter characterization
- Receiver characterization

3.3.1.1 Anti-Alias Filter Frequency Measurements

The signal analyzer anti-alias filter frequency domain characterization consisted of analyzing its frequency response and identifying the -3 dB frequency. The frequency domain characterization was performed using the following equipment:

- Agilent E8358A Performance Network Analyzer (300 kHz to 9 GHz)
- HP/Agilent E2050A LAN-to-GPIB Gateway

Frequency domain data is extracted from the Agilent E8358A Performance Network Analyzer by use of an HP/Agilent E2050A LAN-to-GPIB gateway. A GPIB connection is made between the HP/Agilent E2050A and the Agilent E8358A Performance Network Analyzer. The Agilent E8358A Performance Network Analyzer is then controlled via a Perl script over a LAN connection made between the HP/Agilent E2050A and the controlling computer. In addition to control, the Perl script also performed data collection and waveform extraction. The basic measurement block diagram used to collect data is shown in Figure 3.25.

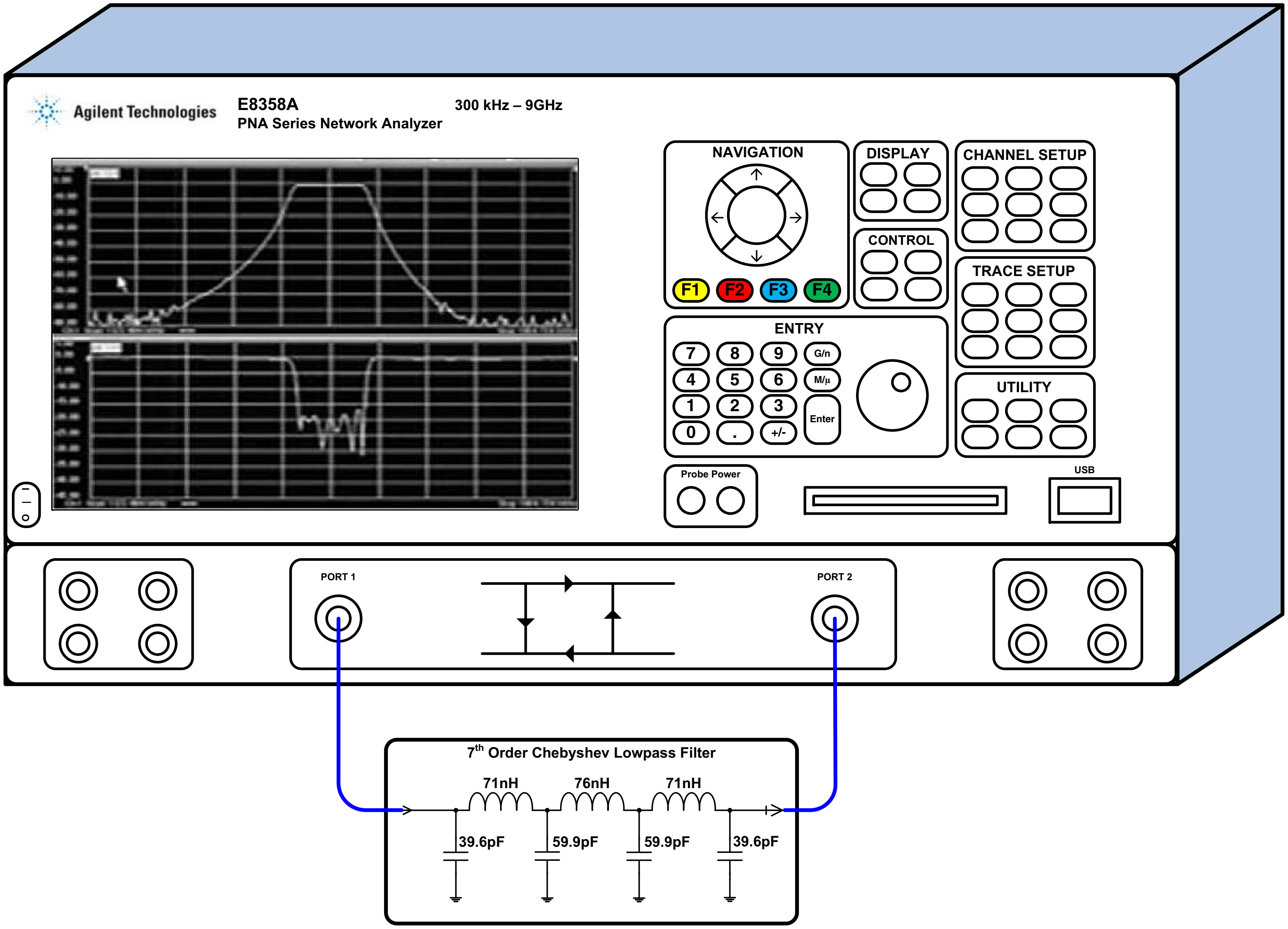

Figure 3.25: Anti-Alias Filter Frequency Domain Measurement Block Diagram

−3 dB Frequency Measurements A network analyzer is used to analyze the frequency response of an anti-alias filter and to determine the −3 dB frequency. The Agilent E8358A Performance Network Analyzer sweeps its CW signal source from DC to 1 GHz. The S-parameters of the two-port network are recorded at each frequency point in real and imaginary format, including S11, S21, S12, and S22. The frequency response of the anti-alias filter is represented by the S21 S-parameter data. The real and imaginary data was converted to a power level in decibels using Equation 3.2.

$$P_{dBm} = 20 \cdot \log_{10} \left(|s21\_real + i \cdot s21\_imag| \right) \tag{3.2}$$

Using the power level information, the frequency response is displayed using an x versus y plot, where the x-axis is plotted in a logarithmic scale. Figure 3.26 shows the resulting frequency response curve created by measuring the power level of the fundamental signal tone at each frequency from DC to the Nyquist frequency. The measured −3 dB frequency was 135 MHz, which is 5 MHz more than the target cut-off frequency.

The noise floor of the S21 measurement is limited by the noise level of the Agilent E8358A Performance Network Analyzer.

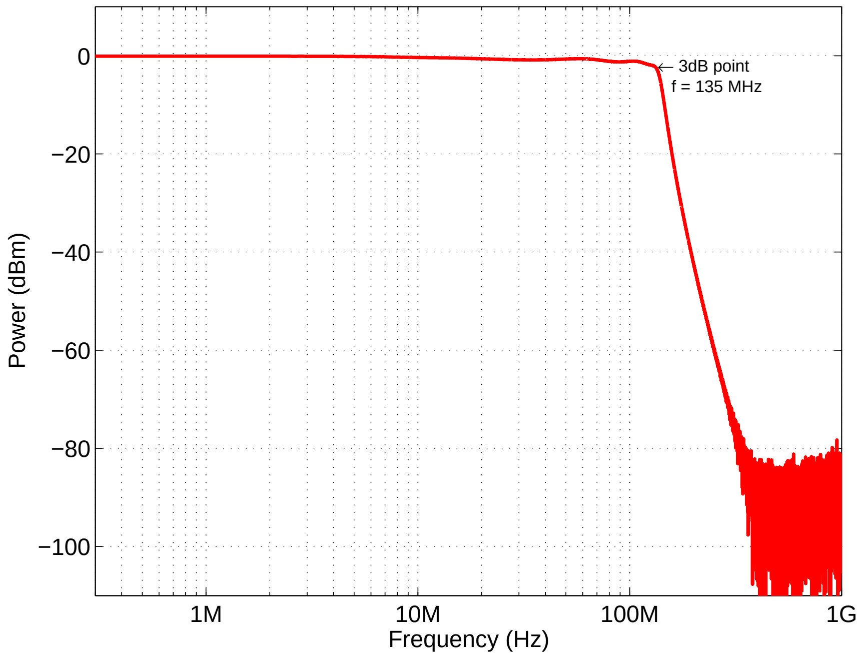

Figure 3.26: Signal Analyzer Anti-Alias Frequency Response. Captured with an Agilent E8358A Performance Network Analyzer.

3.3.1.2 Receiver Characterization

Receiver characterization encompasses the entire signal analyzer IF chain. The signal analyzer input is stimulated with a variety of signals and the digitized waveform data is then analyzed using Matlab. The digitized data is captured directly from the high-speed ADC via the Control and Data Path FPGAs on the measurement board. The receiver characterization for the signal analyzer consisted of evaluating the DC performance, the two-tone third-order intermodulation distortion performance, and the frequency response to determine the -3 dB frequency.

DC Measurements A desired feature of a receiver is sensitivity, thus it is helpful to first understand the performance of a receiver under DC conditions. The basic measurement block diagram used to collect data is shown in Figure 3.27. A 50 Ω load was attached to the signal analyzer input to provide a DC signal. The measurement was performed with the lid of the board level shield removed to provide a worst case analysis. The noise floor can be affected by many factors, including:

- Power supply noise
- Electromagnetic interference (EMI)
- ADC dynamic range and resolution

Figures 3.28 and 3.29 show the signal analyzer performance when terminated with a 50 Ω load in the time and frequency domain, respectively. As shown in Figure 3.29, the noise floor of the signal analyzer is -98 dBm.

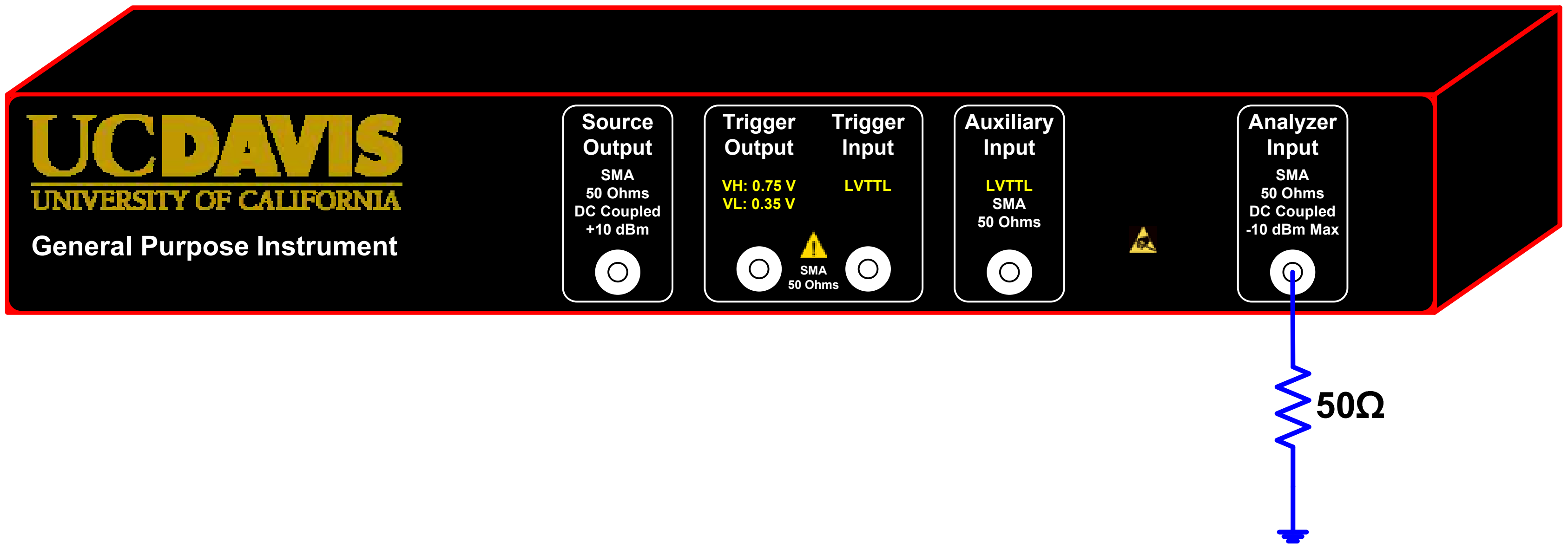

Figure 3.27: Signal Analyzer DC Measurement Block Diagram

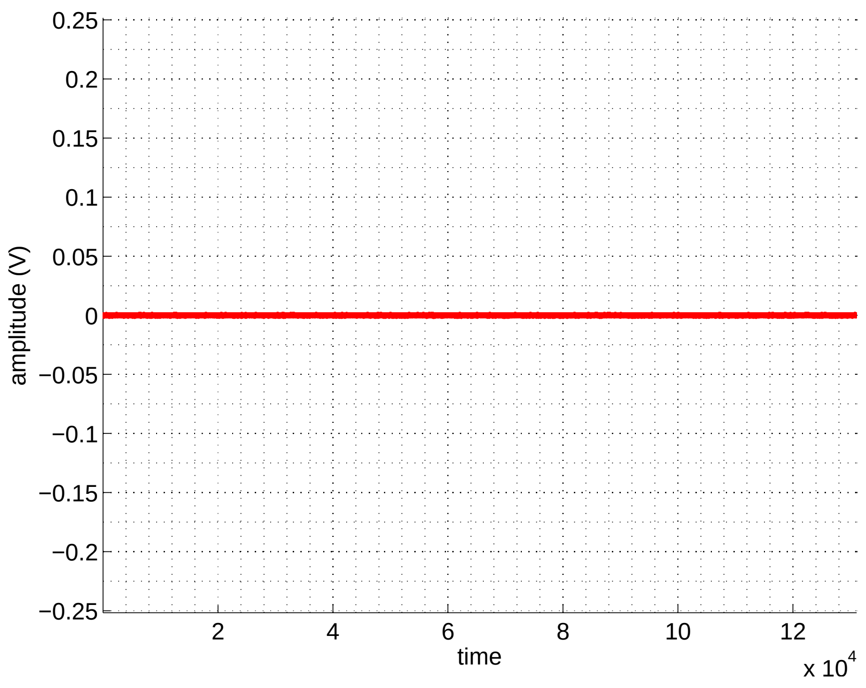

Figure 3.28: DC Waveform shown in the time domain. Stimulated with a 50 Ω attached to the input of the General Purpose Instrument Signal Analyzer.

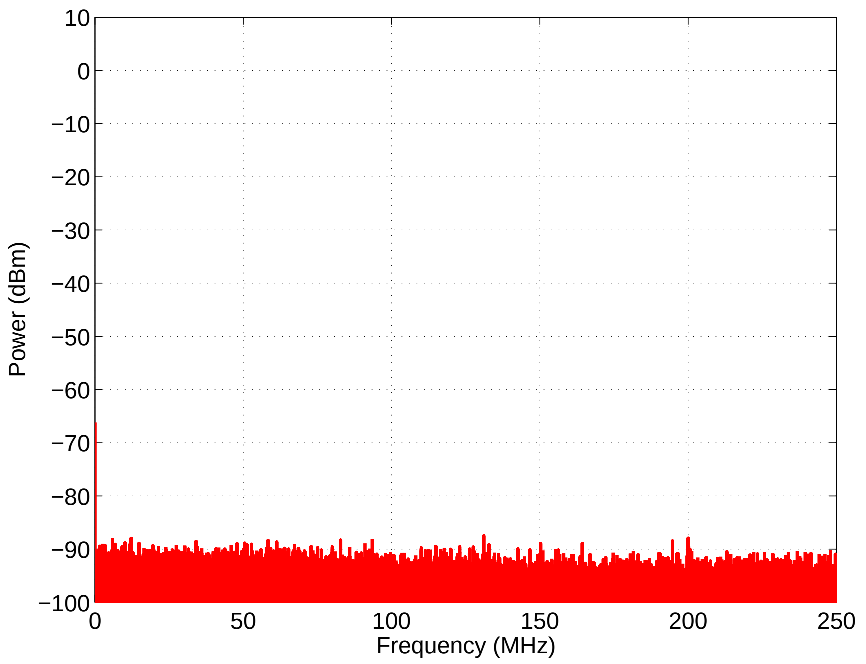

Figure 3.29: DC Waveform shown in the frequency domain. Stimulated with a 50 Ω attached to the input of the General Purpose Instrument Signal Analyzer.

−3 dB Frequency Measurements Two methods are commonly used to analyze the frequency response of a signal analyzer and determine the −3 dB frequency. The simplest and least accurate method is to generate a comb signal, also known as a bed of nails, which contains signal tones from DC to the Nyquist frequency. The comb signal used to stimulate the low-pass filter is shown in Figure 2.27. The resulting signal received by the baseband signal analyzer will have the shape of a low-pass filter's frequency response. From this information the −3 dB frequency of the signal analyzer output can be roughly estimated. Figure 3.31 shows the resulting frequency versus power level data extracted from the baseband signal analyzer with a −3 dB frequency between 133 MHz and 135 MHz, which is 5 MHz above the target −3 dB frequency. The increased −3 dB frequency results in more signal bandwidth.

A more accurate method to analyze the frequency response is to sweep the signal analyzer from DC to the Nyquist frequency. The power level of each each frequency point is recorded. Using the power level information, the frequency response can then be displayed using an x versus y plot, where the x-axis is plotted in a logarithmic scale. Figure 3.33 shows the resulting frequency response curve created by measuring the power level of the fundamental signal tone at each frequency from DC to the Nyquist frequency. The measured −3 dB frequency was 134 MHz, which is 4 MHz more than the target cut-off frequency. The basic measurement block diagram used to collect data is shown in Figure 3.32.

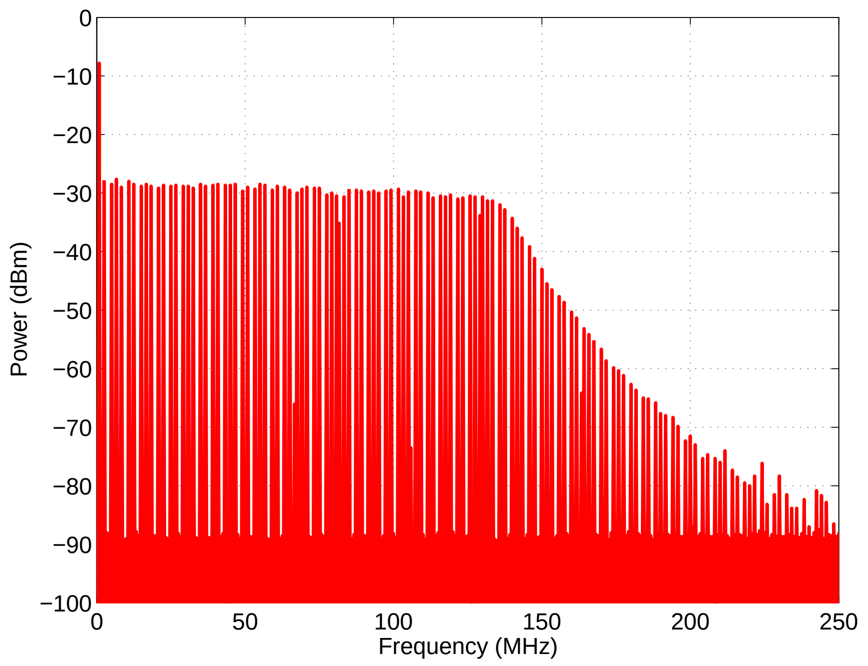

Figure 3.30: Comb input signal used to stimulate the signal analyzer to evaluate the IF frequency response. Generated by the General Purpose Instrument Signal Source.

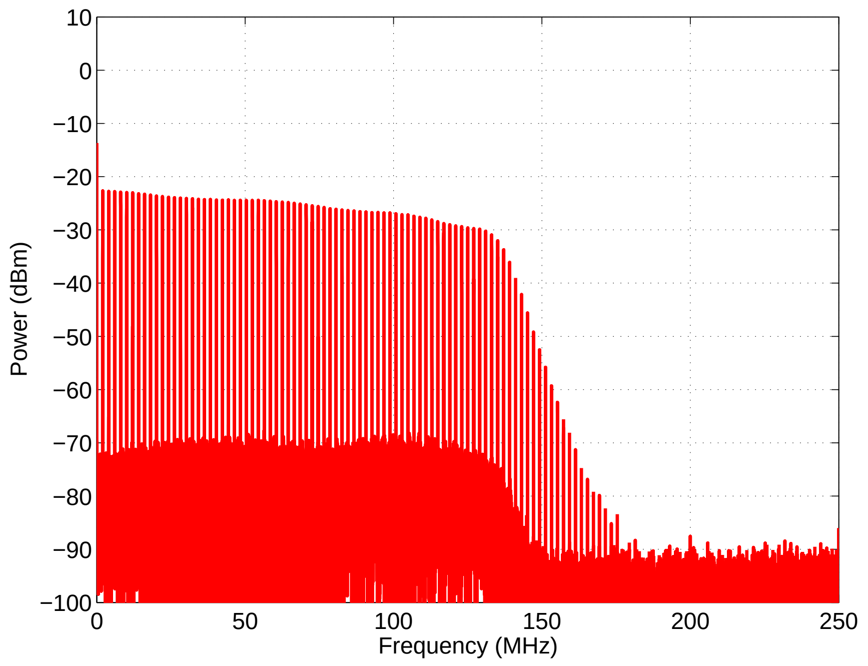

Figure 3.31: Comb Signal Frequency Response. Captured with the General Purpose Instrument Signal Analyzer.

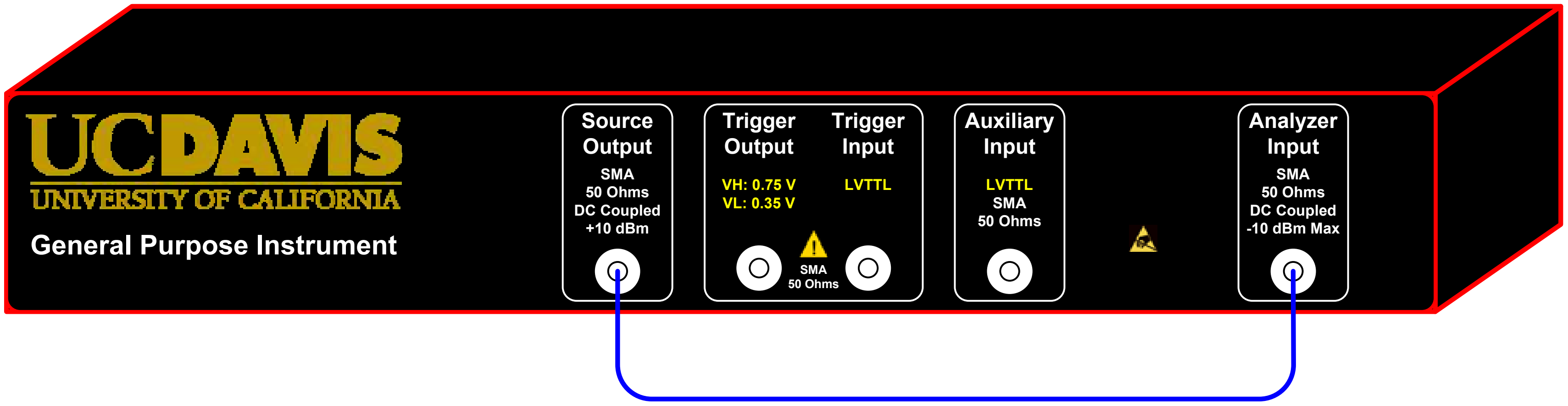

Figure 3.32: Signal Analyzer Frequency Domain Measurement Block Diagram

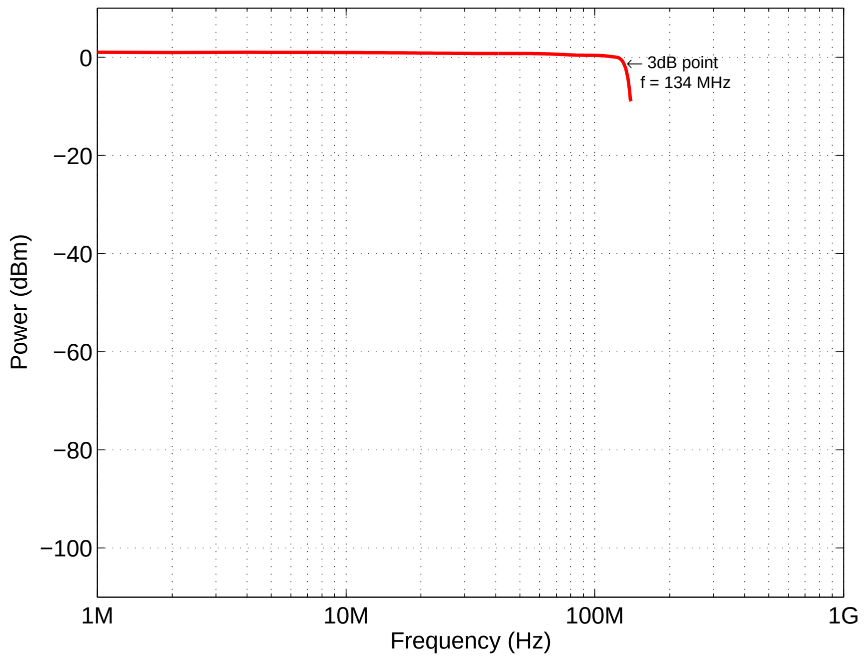

Figure 3.33: Signal Analyzer Frequency Response. Captured with the General Purpose Instrument Signal Analyzer.

Two-tone Third-order Intermodulation Distortion Measurement The presence of multiple signals in a system is sometimes desired; for example, the generation of multi-tone signals or comb signals. However, undesired signals (e.g., noise) are typically present in a signal system and can mix with the desired signal to generate distortion products. Understanding the effects of distortion is important when evaluating the measurement results of a DUT.

Intermodulation distortion is a type of distortion caused by the presence of two or more signal tones at the input of a non-linear device [10]. This distortion causes spurious signals to be generated, which are related to the original signal tones. The complexity of the distortion increases as the number of signal tones present in the system increases beyond two. As such, the distortion performance of signal analyzer systems are typically analyzed with two signal tones. The relationship of the two original signal tones and the generated spurious signal tones is described by Equation 3.3.

$$M \cdot f_1 \pm N \cdot f_2, \text{ where } M,\ N\ =\ 0,\ 1,\ 2,\ 3,\ \ldots \tag{3.3}$$

The order of the distortion product is represented by the sum of $M+N$. For example, the third-order intermodulation products of two signals at f_1 and f_2 would be:

$$2 \cdot f_1 + f_2$$
$$2 \cdot f_1 - f_2$$
$$f_1 + 2 \cdot f_2$$
$$f_1 - 2 \cdot f_2$$

Third-order two-tone intermodulation distortion is a metric used to describe the distortion performance of a transmitter or receiver when multiple signal tones are present in the data stream. It is measured by driving two spectrally pure sine waves through the DUT at frequencies f_1 and f_2, where the difference in frequency is small. The amplitude of each tone is generally attenuated by 6 dB to avoid clipping in the signal system. It is typically specified in dBc relative to the value of either of the two input tones [11]. Figure 3.32 shows the test setup used to measure third-order two-tone intermodulation distortion.

The IMD_3 performance of the signal analyzer was measured using two signal tones at frequencies:

- f_1 = 107.7 MHz
- f_2 = 107.9 MHz

Figure 3.34 shows the resulting IMD_3 measurement performed with the baseband signal analyzer. The IMD_3 parameter was determined by measuring the power level of a fundamental signal tone, f_1 or f_2, (P_o) and one of the spurious tones (P_{o_3}) in units of dBm. Equation 3.4 describes the relationship between P_o and P_{o_3} when calculating IMD_3.

$$IMD_3 = P_o - P_{o_3} = -17.9 \text{ dBm} - (-82.27 \text{ dBm}) = 64.37 \text{ dBc} \tag{3.4}$$

An additional parameter, known as the third-order intercept (TOI) point, can be used to quantify the distortion performance of a signal analyzer. TOI can be calculated using the power level of the fundamental tones along with the third-order two-tone intermodulation distortion, and is typically specified in dB. Equation 3.5 describes the relationship between P_o and IMD_3.

$$TOI = \frac{IMD_3}{2} + P_o = \frac{64.37 \text{ dBc}}{2} + (-17.9 \text{ dBm}) = 14.285 \text{ dB} \tag{3.5}$$

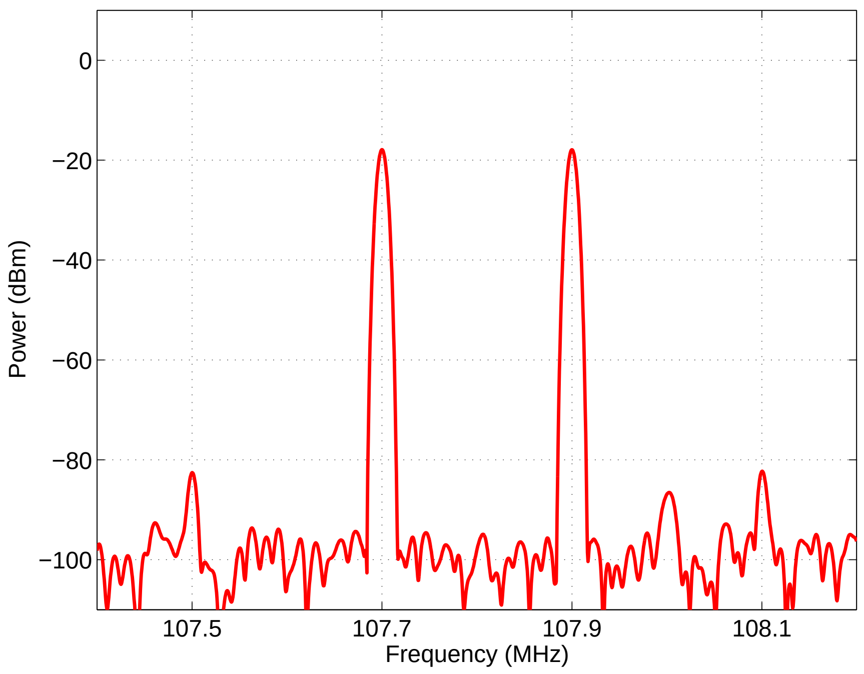

Figure 3.34: Two-tone third-order intermodulation distortion (IMD$_3$) plot for Signal Analyzer. Tone frequencies: f_1 = 107.7 MHz, f_2 = 107.9 MHz. Captured with the General Purpose Instrument Signal Analyzer.

3.4 Specifications

A summary of the signal source specifications are shown in Table 3.6. These specifications were determined by characterizing the signal analyzer outputs of four General Purpose Instruments.

Table 3.6: Signal Analyzer Specifications

PARAMETER	**TEST CONDITIONS**	**MIN**	**TYP**	**MAX**	**UNIT**
Frequency Domain Specifications					
IMD_3	$f_{ADC_{DCLK}} = 250$ MHz, $f_{ADC_{CLK}} = 500$ MHz, $f_1 = 80$ MHz, $f_2 = 100$ MHz, Power level of each tone -6 dBFS		64.37		dBc
TOI	$f_{ADC_{DCLK}} = 250$ MHz, $f_{ADC_{CLK}} = 500$ MHz, $f_1 = 107.7$ MHz, $f_2 = 107.9$ MHz, Power level of each tone -6 dBFS		14.285		dB
Time Domain Specifications					
Period	$f_{ADC_{DCLK}} = 250$ MHz, $f_{ADC_{CLK}} = 500$ MHz, Sine Wave Power Level 0 dBFS	1e6		8	ns
Frequency	$f_{ADC_{DCLK}} = 250$ MHz, $f_{ADC_{CLK}} = 500$ MHz, Sine Waveform Power Level 0 dBFS	0.001		125	MHz
	$f_{ADC_{DCLK}} = 250$ MHz, $f_{ADC_{CLK}} = 500$ MHz, Square Waveform	0.001		25	MHz
	$f_{ADC_{DCLK}} = 250$ MHz, $f_{ADC_{CLK}} = 500$ MHz, Ramp Waveform	0.001		10	MHz
Amplitude	$f_{ADC_{DCLK}} = 250$ MHz, $f_{ADC_{CLK}} = 500$ MHz, Sine Wave Power Level 0 dBFS	-1.1		1.1	V

Chapter 4

Measurement Board

The asynchronous array of simple processors is a single-chip, multi-core, computational platform that is well-suited for DSP, embedded, and multimedia applications [4]. A measurement board was designed to aid in developing software applications targeting the AsAP processor. The following is an example of the types of applications supported by the measurement board:

- Software defined radio
- Signal source
- Arbitrary waveform generator
- Spectrum analyzer
- Network analyzer
- FFT analyzer
- Oscilloscope

To demonstrate the capabilities of the AsAP processor, a high-speed ADC and DAC, along with several types of memory, were included on the measurement board. Two AsAP chips were provided to fully exercise the ADC and DAC circuits. A companion FPGA facilitates communication between the AsAP and the memory devices.

4.1 Requirements

The combination of digital and analog circuits creates a difficult environment to maintain the high level of performance required by the measurement board. To ensure optimal performance, the following areas of printed circuit board (PCB) design were addressed:

- Power Distribution System
- Radio Frequency and Electromagnetic Interference (RF/EMI)
- Component Placement
- Layout Strategy
- Signal Integrity

4.1.1 Power Distribution System

Many factors of printed circuit board design can affect the performance of the power distribution system (PDS) including the construction and layer stackup of the PCB, power supply filter design, and IC power supply decoupling schemes. The PDS is responsible for distributing clean power, power supply decoupling, and providing a low-impedance return path for current [18, 19].

The layer stackup was designed to minimize the coupling of power supply noise onto transmission lines of adjacent PCB layers. In addition, power supply filters were designed to filter out the switching frequency of the power supply regulators and converters. Adequate bypassing was provided for both the analog and digital devices, and X2Y capacitors were used in order to minimize mounting inductance on the PCB.

4.1.2 Component Placement

The measurement board was designed to fit in a 1U tall, rack-wide instrument chassis. Component placement is critical for the successful routing of the measurement board. The front and rear panel I/O connections on the measurement board are located at fixed positions and were used as starting points for the component placement. The pre-layout placement was performed using a preliminary board outline. As the layout of the board evolved, modifications were made to the component placement. In addition to the location of the components, the physical size of the components must also be evaluated based on their relative position in the chassis. For example, components that are too tall for the bottom side must be relocated to the top side, and vertical connectors placed too close to each other must be separated to accommodate insertion and removal of cable assemblies.

4.1.3 Layout Strategy

Manufacturers of high-speed devices with parallel data busses typically require matching the trace length of each signal in the bus for proper operation. However, due to component placement density and the number of data busses that needed to connect to the Data Path FPGA, it is not always possible to meet this requirement. The measurement board takes advantage of special FPGA delay elements in an attempt to equalize the trace lengths of each connecting data bus. For example, the measurement board uses high-speed, synchronous 36 Mbit QDR-II SRAM for waveform capture and playback. The synchronous SRAM operates at 250 MHz double data rate and has separate 36-bit read and write busses. The Xilinx SRAM memory controller requires the lengths of the read data, write data, and control signals to be matched. The density of signals connected between the SRAM and the FPGA is such that it is difficult to successfully match the length of all the traces. The use of FPGA delay elements greatly simplified the routing of the SRAM signals.

The use of an FPGA offers flexibility in the pin assignment of signals. Pin planning was performed for the FPGA signals with the aid of the preliminary component placement. Lack of pin planning can result in an unrouteable design. For example, assigning the signals of a data bus without proper planning may result in signals being crossed, which would require the use of many vias to effectively unravel the signals and result in poor signal performance.

4.1.4 Radio Frequency and Electromagnetic Interference

The target bandwidth for the signal source and analyzer is 120 MHz. Both front-ends were designed for operation in the first Nyquist zone (DC–120 MHz). Any signals on the measurement board operating at or below 120 MHz can potentially mix with the signal being generated or analyzed. The harmonics and sub-harmonics of any signals operating faster than 120 MHz are also potential sources of noise. In an attempt to minimize signal corruption, the signals of the front-end designs were routed on the top layer and enclosed in a board-level surface mount shield.

High-speed devices are susceptible to simultaneous switching noise (SSN), which is caused by more than one I/O toggling at the same time. The interconnection of the high-speed devices on the measurement board which share a common power supply can cause a significant voltage drop when I/O are switching at the same time. There are many ways to combat the effects of SSN including minimizing the number of I/Os that may switch simultaneously in FPGA I/O banks, providing adequate power supply decoupling, employing differential signal topologies, and minimizing the inductance of decoupling capacitors [20]. All high-speed devices were decoupled using a combination of tantalum, ceramic, and X2Y capacitors. The use of decoupling capacitors ensures that each high-speed device has the appropriate amount of transient current during device operation [21].

The close proximity of unique power supply regulators that have a common input voltage can add noise to the input supply. This is especially true when unshielded inductors are used in power supply filters. In an attempt to minimize EMI, shielded inductors were employed in the power supply filters of DC/DC converters.

4.1.5 Signal Integrity

The electrical performance of the high-speed signals on the measurement board is very important. Impedance discontinuities of high-speed traces can cause several problems including reflections and ringing. High-speed circuits were simulated using a variety of simulators such as Agilent's advanced design system (ADS), Linear Technology's LTSpice, and DesignSoft's TINA to ensure the best signal integrity. Both single-ended and differential transmission line structures were used to achieve the target impedance of each high-speed trace. Single-ended transmission lines typically have a target impedance of 50 Ω, whereas differential transmission lines have an impedance of 100 Ω. In addition to impedance, high-speed signal paths were also designed to minimize reflections by using either source or end terminations, or a combination of the two.

4.2 Design

The measurement board, shown in Figure 4.1, was designed using Mentor Graphics DxDesigner Schematic Entry and PADS Layout software over a period of six months. It contains 2,262 components and cost approximately $4,000.00 to fabricate and assemble. The overall dimensions of the measurement board are 15.51 inches X 12.05 inches. Given the size, component density, and routing density it is considered to be a complex printed circuit board.

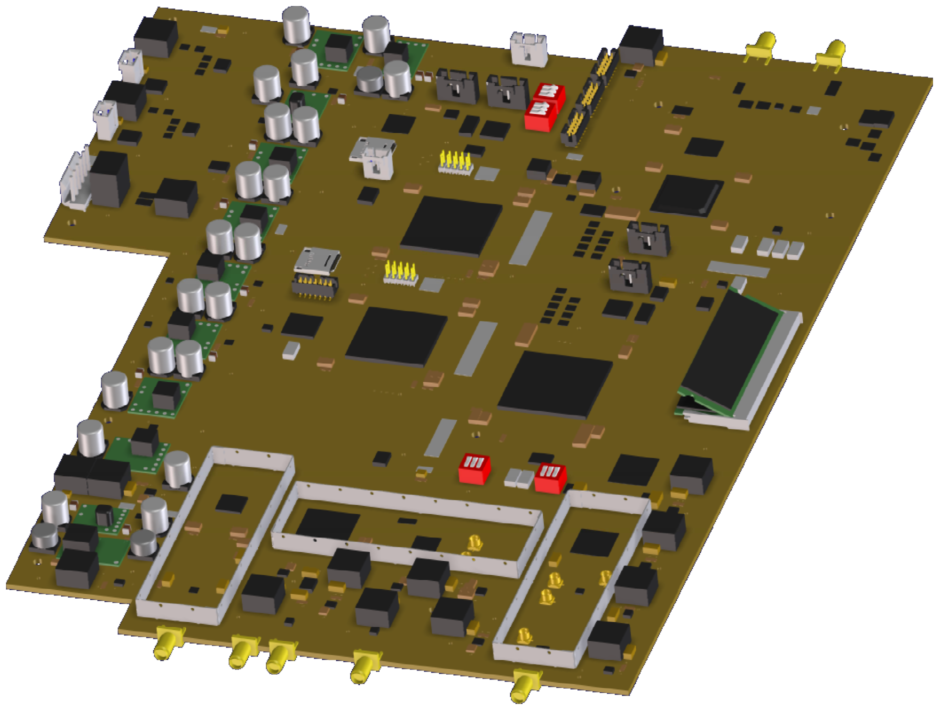

Figure 4.1: Measurement Board ISO View.

4.2.1 Printed Circuit Board Construction

The measurement printed circuit board is fabricated using a 12-layer mixed dielectric core construction and assembled using a lead-free process. The two dielectric materials used are:

- Rogers Corporation RO4003C laminate [22]
- Isola Group FR408 laminate/prepreg [23]

Rogers RO4003C laminate is a low loss material designed for high frequency circuits and has a dielectric constant (ϵ_r) of 3.38. Isola FR408 laminate has similar properties to conventional FR-4 and a dielectric constant of 3.9. Both laminate materials are compatible with a lead-free process, which means they can withstand the higher temperatures required to melt lead-free solder. Typically, lead-free solder has a melting point 34 °C to 37 °C higher than lead-based solder [24]. The stackup used for the measurement board is shown in Figure 4.2. Its dimensions are outlined in Table 4.1. The mixed dielectric construction allows for a cost effective solution with high performance signals routed on the expensive Rogers RO4003C laminate and slower signals routed on the less expensive Isola FR408 laminate.

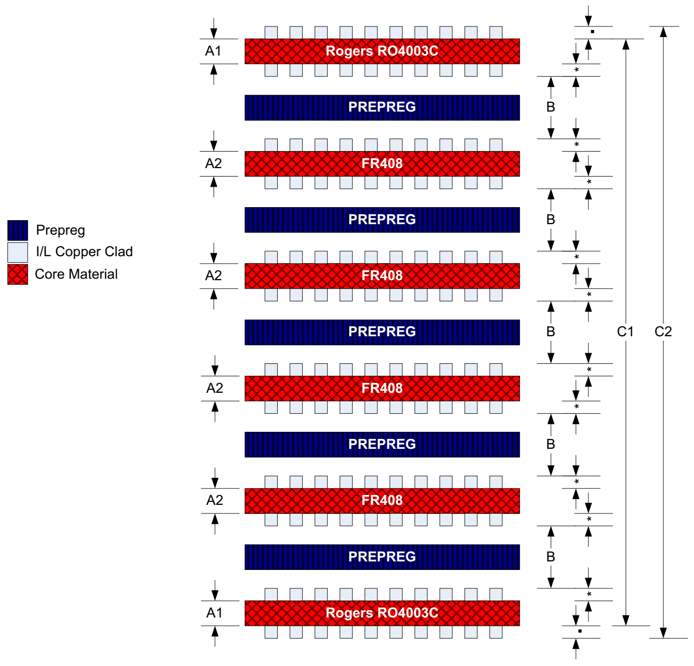

Figure 4.2: 12-Layer FR408/Rogers RO4003C Core Construction.

Table 4.1: 12-Layer FR408/Rogers RO4003C Core Construction

12-Layer FR408/Rogers RO4003C Core Construction					
Construction Type	**A1 (inches)**	**A2 (inches)**	**B (inches)**	**C1 (inches)**	**C2 (inches)**
FR408/Rogers Mix	.008 ± .001	.008 ± .001	.0093 ± .0007	.092-.111	.0934-.1124

* - Copper thickness on inner/outer layers, unless otherwise specified: $\frac{1}{2}$ oz. Cu (0.0007”).

A1 - Rogers RO4003C Core thickness

A2 - FR408 Core thickness

B - Pressed thickness of prepreg - 1x FR408 Prepreg 1080 and 1x FR408 Prepreg 7628

C1 - Overall finished board thickness substrate-to-substrate

C2 - Overall finished board thickness plated metal to plated metal. In addition, some surface coatings (i.e., HAL) can add up to 0.002” of solder per side.
Soldermask (also not specified) typically adds about 0.001” per side but can add up to 0.004” per side in extreme examples.

4.2.1.1 Printed Circuit Board Stackup

The measurement board layer stackup is divided into two sections: analog and digital. These two sections have different requirements based on:

- Number of power supply voltages
- Current requirements of power supplies
- Number of signal nets
- Density of component placement
- Available board area

The board layer stackup assignments are shown in Table 4.2. A symmetric layer stackup was employed to help prevent warping of the printed circuit board [25].

Table 4.2: 12-Layer FR408/Rogers RO4003C Stackup

Layer Name	**Signal or Plane**	**Analog Stackup**	**Digital Stackup**
TOP	Signal	Routing	Routing, Ground, and BGA Breakout
SIDE2	Plane	Ground	Ground
SIDE3	Signal	Routing	Routing - Offset Stripline
SIDE4	Signal	Routing	Routing - Offset Stripline
SIDE5	Plane	Ground	Ground
SIDE6	Signal	Power	Power - +3.3V, +1.8V, and +5V
SIDE7	Signal	Power	Power - +2.5V, +1.2V, and +1V
SIDE8	Plane	Ground	Ground
SIDE9	Signal	Power	Routing - Offset Stripline
SIDE10	Signal	Power	Routing - Offset Stripline
SIDE11	Plane	Ground	Ground
BOTTOM	Signal	Routing	Routing, Ground, and BGA Breakout

High-speed analog stackup The majority of the high-speed analog signals are routed on the top layer in order to minimize effects of layer transitions. The measurement board has 3 main high-speed analog circuits:

- High-speed Analog-to-Digital Converter
- High-speed Digital-to-Analog Converter
- High-speed clock generation

Each circuit has its own set of power supplies, the benefits of which include isolation of high-speed analog circuits, distributed heat generation, and heat dissipation.

High-speed digital stackup The high-speed digital signals are very dense and require many signal routing layers. The measurement board has 7 main high-speed digital circuits:

- High-speed Analog-to-Digital Converter
- High-speed Digital-to-Analog Converter
- AsAP #1
- AsAP #2
- QDR-II SRAM
- DDR2 SDRAM (SODIMM)
- Data Path FPGA

Unlike the high-speed analog circuits, the majority of the high-speed digital circuits share a common digital power supply.

4.2.1.2 Transmission Line Structures

The measurement board contains several high-speed mixed-signal circuits that require controlled impedance transmission lines. The four transmission line structures employed on the measurement board are shown below.

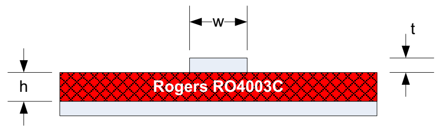

Figure 4.3: Single-Ended Microstrip

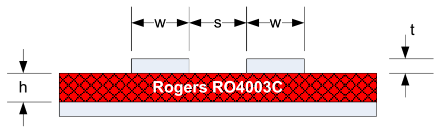

Figure 4.4: Differential Microstrip

Transmission Line	**w** **(Inch)**	**h** **(Inch)**	**s** **(Inch)**	**t** **(oz)[Inch]**	**Z_o** **(Ohms)**	ϵ_r
Single-Ended (Solder Mask)	0.015	0.008 ± 0.001	N/A	$\frac{1}{2}$[0.0007]	52 Ω	3.38
Single-Ended (No Solder Mask)	0.016	0.008 ± 0.001	N/A	$\frac{1}{2}$[0.0007]	52 Ω	3.38
Differential (Solder Mask)	0.010	0.008 ± 0.001	0.006	$\frac{1}{2}$[0.0007]	103 Ω	3.38

Table 4.3: Microstrip Transmission Line Information

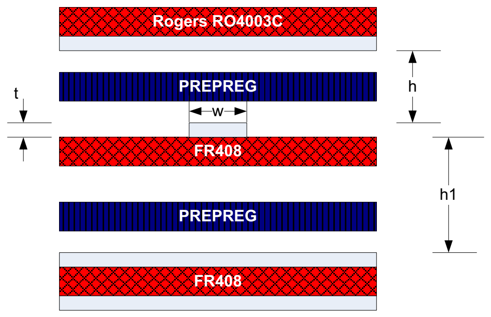

Figure 4.5: Single-Ended Stripline

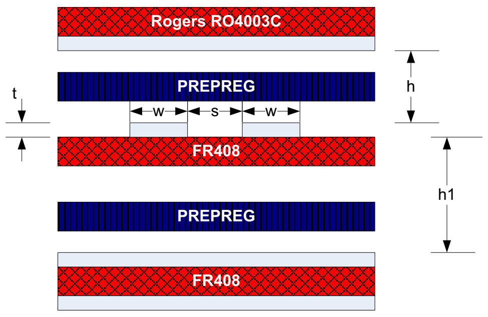

Figure 4.6: Differential Stripline

Transmission Line	**h1** **(Inch)**	**h** **(Inch)**	**w** **(Inch)**	**s** **(Inch)**	**t** **(oz)[Inch]**	**Z_o** **(Ohms)**	ϵ_r
Single-Ended	0.0149 ± 0.0017	0.0062 ± 0.0007	0.007	N/A	$\frac{1}{2}$[0.0007]	49 Ω	3.90
Differential	0.0149 ± 0.0017	0.0062 ± 0.0007	0.004	0.004	$\frac{1}{2}$[0.0007]	107 Ω	3.90

Table 4.4: Stripline Transmission Line Information

4.2.2 Component Placement

Component placement can have a huge impact on the routeability of a printed circuit board. Once the PCB construction and stackup designs have been completed, a board outline can be chosen and the component placement can begin. The measurement board was designed to fit in a 1U tall, 19" rack-wide, 16" deep instrument chassis. The chosen board outline is shown in Figure 4.7.

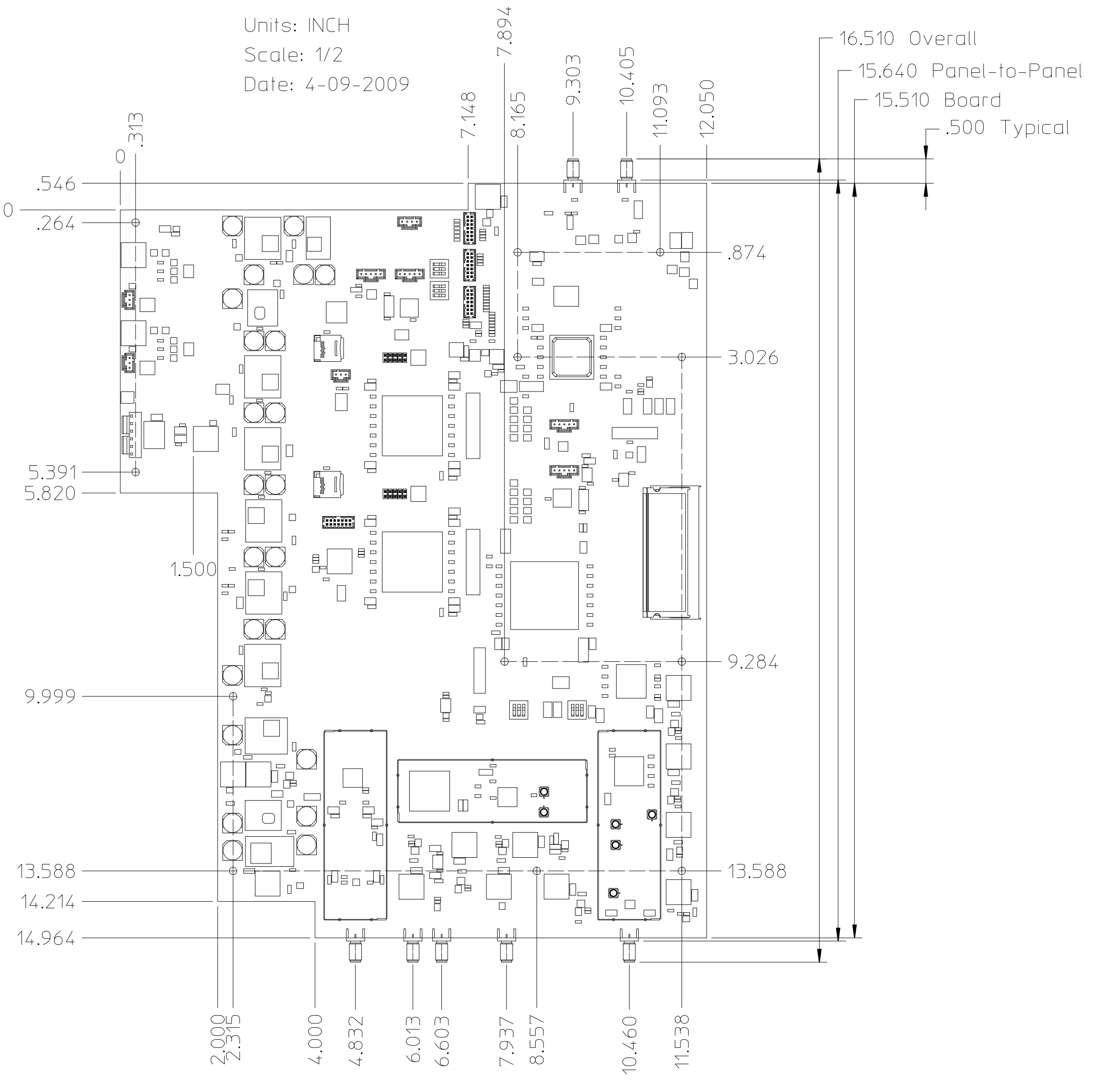

Figure 4.7: Printed Circuit Board Outline

4.2.2.1 Printed Circuit Board Pre-Placement

Using the initial board outline, a preliminary component placement was performed in Microsoft Office Visio 2007. The printed circuit board outline was imported as a drawing exchange format (DXF) file. Components were drawn and scaled proportionally to the board outline in order to represent the physical size of the devices.

4.2.2.2 FPGA Pin Assignments

Both FPGAs are central components of the measurement board. They are used to control peripheral devices, store and retrieve data from memory, and interface with digital signal processors. As a result, it was very important to plan the pin out of the FPGA devices before placing components. Pin planning was performed with the aid of Xilinx' PlanAhead software package. This software provides both a package and device view, and allowed the signals to be placed on appropriate pins while keeping in mind the relative placement of other signals, the I/O standards for the various signals, and whether signals needed to be placed in a bank compatible with differential signals. Several of the digital device interfaces required the signals to be placed on adjacent I/O blocks of the device in pin order; the PlanAhead software was instrumental to the success of the pin planning for these devices.

4.2.2.3 Decoupling Capacitor Placement Planning

Early planning of the decoupling capacitor placement around the perimeter of the FPGAs was critical. By placing the decoupling capacitors before starting the layout, routing channels were created for the signals to reach their destination, the mounting inductance of the decoupling capacitors was minimized, and the power supply planes or area fills for each FPGA supply were defined. Figures 4.8 to 4.11 show the preliminary placement of the top and bottom side decoupling capacitors for each FPGA device. The decoupling capacitor placement of the bottom-side of the PCB mirror the top-side with the addition of ceramic capacitors in an 0402 package placed close to the power pins in the ball-grid array.

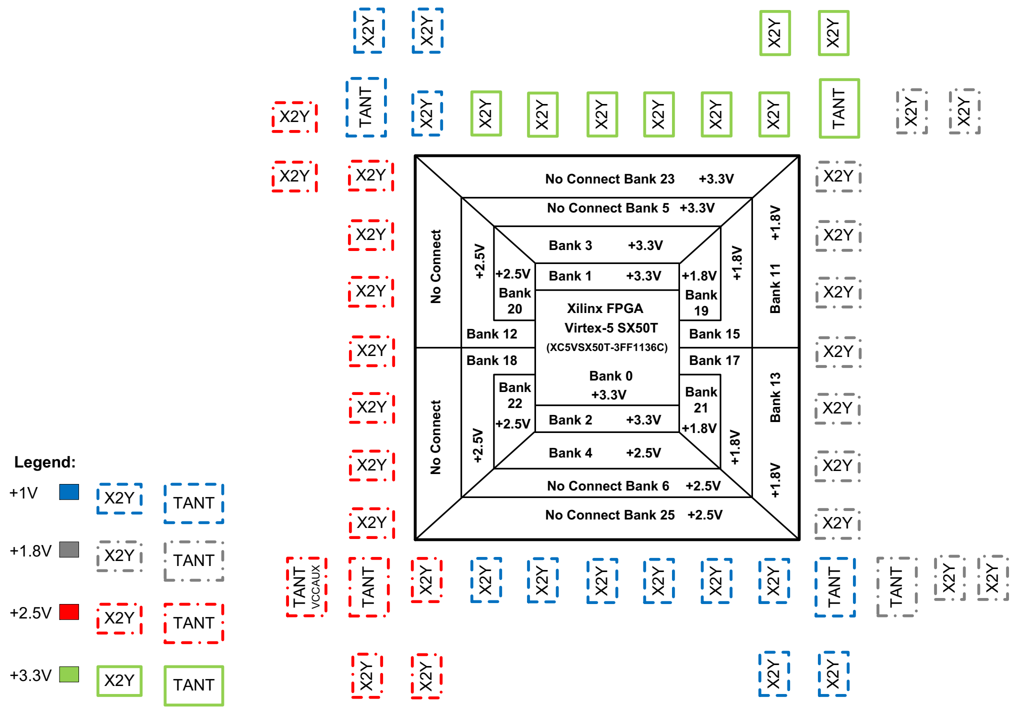

Figure 4.8: Xilinx Virtex-5 FPGA Decoupling Capacitor Top Side Placement

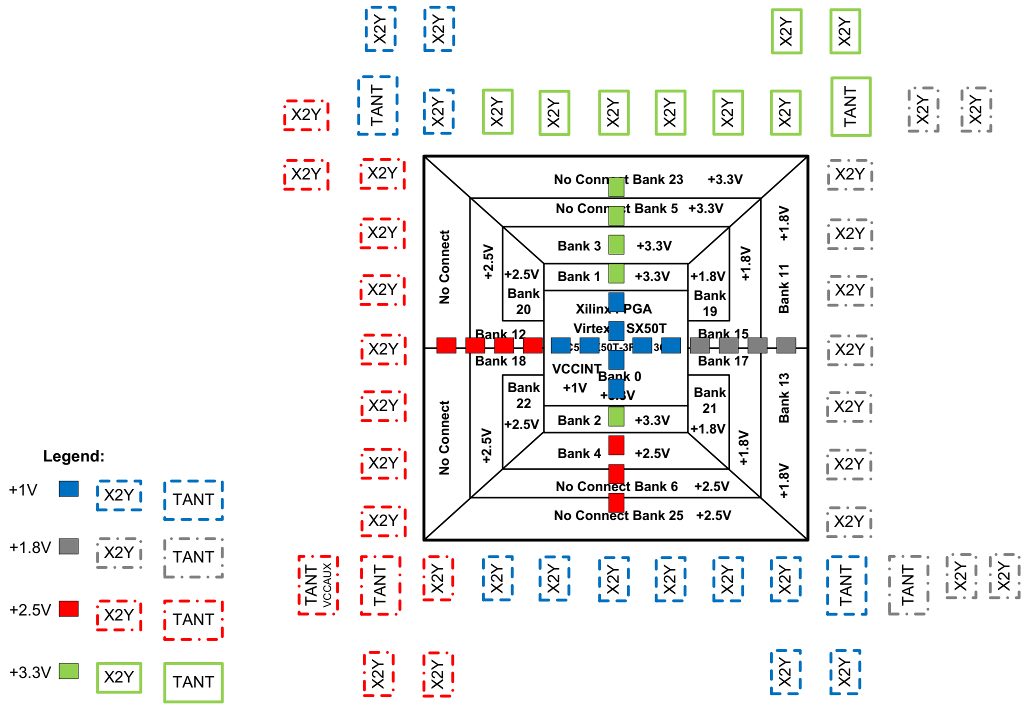

Figure 4.9: Xilinx Virtex-5 FPGA Decoupling Capacitor Bottom Side Placement

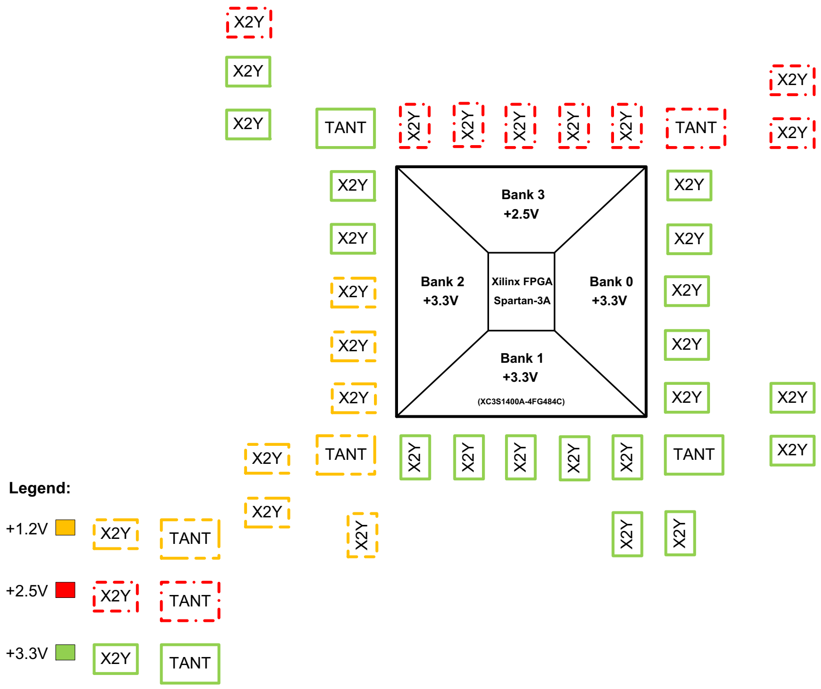

Figure 4.10: Xilinx Spartan-3A DSP FPGA Decoupling Capacitor Top Side Placement

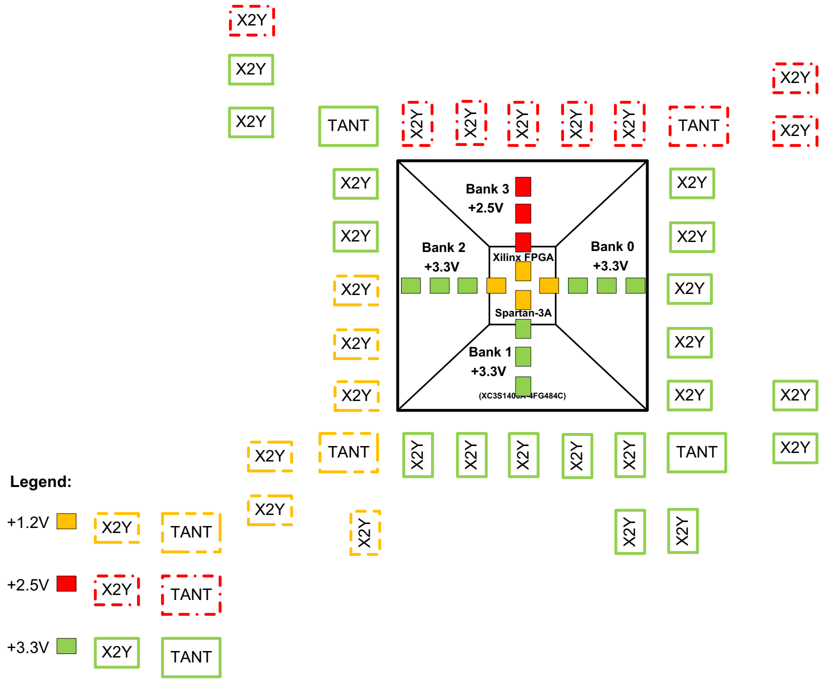

Figure 4.11: Xilinx Spartan-3A DSP FPGA Decoupling Capacitor Bottom Side Placement

4.2.2.4 High-Speed Device Considerations

The high-speed signal generator and analyzer circuits interface to the user via the front panel of the chassis. As a result, these circuits were placed as close to the end-launch SMA connectors as possible. The circuits were placed in a linear signal flow such that all traces could be routed on the top side of the board. Each circuit receives a sample clock from the clock distribution circuit, which was placed between the two groups of circuits. Given the close proximity of these devices and the signal performance requirements, it was important to minimize any possible radiation into or out of these circuits. Each circuit was enclosed in a single-cavity board level shield to minimize RF/EMI radiation from circuit to circuit and within the chassis. The board level shield was manufactured using a tin-plated, mild steel material, and has a removable top cover for debugging of the internal circuits. The components of each circuit were placed in such a way that each could use the same sized board level shield. This resulted in a cost savings for the shield design, since a non-recurring engineering charge of $350.00 was required for each unique shield design. The total cost per shield for a quantity of 25 shields was $11.04. The dimensions and mechanical outline of the board level shields is shown in Figure 4.12.

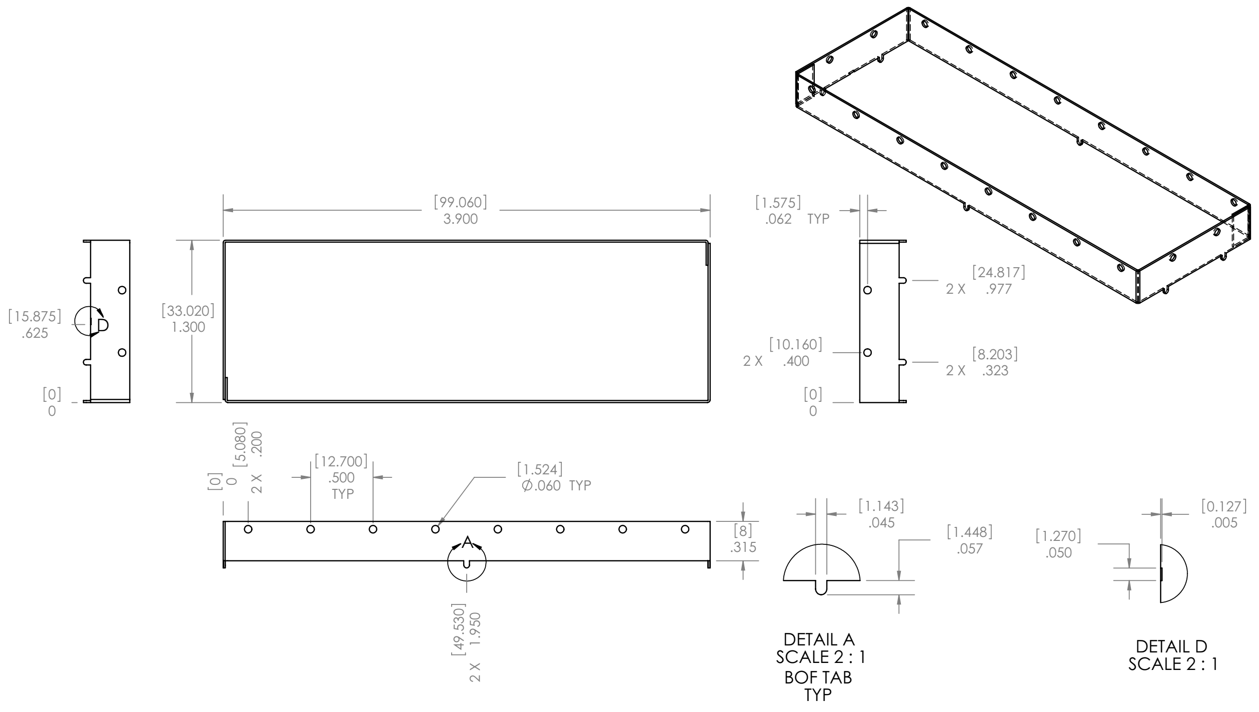

Figure 4.12: Board Level Shield Dimensions

Printed Circuit Board Placement The preliminary component placement was used by the PCB layout designer as a starting point for the actual board design. The final version of the preliminary component placement is shown in Figure 4.13

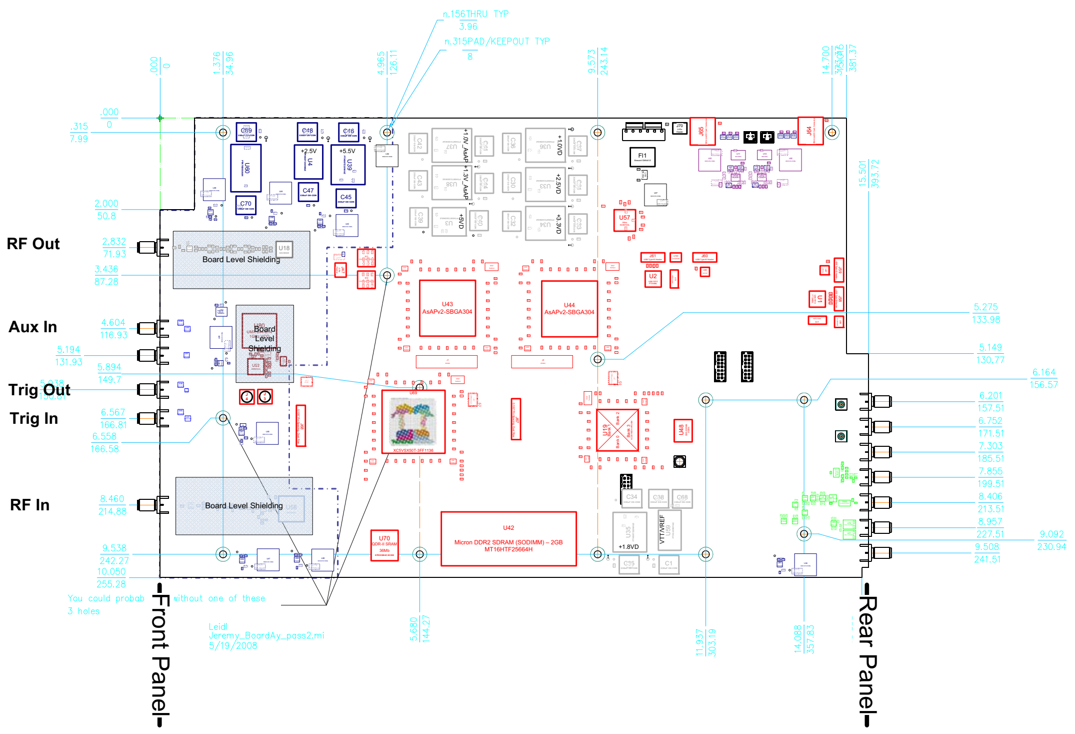

Figure 4.13: Preliminary Component Placement

The measurement board component placement was performed over a period of two weeks, and continued to evolve throughout the layout process. During the component placement phase of the design, the board was continually evaluated by a mechanical engineer to ensure the board would fit properly in the chassis. The measurement board was modeled using a 3D CAD program called CoCreate. The PCB layout designer exported the design as an intermediate data format (IDF) file, which the mechanical engineer could use to import into CoCreate. Once the board was imported, it was analyzed to ensure the following requirements were met:

- Top-side components are no taller than 1.100”
- Bottom-side components are no taller than 0.200”
- All end-launch SMA connectors align with the front and rear panel cut-outs

- All mounting holes align with the stand-off placement on the chassis base plate
- All connectors are accessible when the board is mounted in the chassis
- All daughter card mounting holes and pads are in the correct location

4.2.3 Printed Circuit Board Layout

Signal routing is the most time-consuming and complex part of printed circuit board design. As a result, careful planning is required in order to achieve the desired performance. Once the preliminary component placement has been completed, the size and shape of the power and ground planes can be chosen, net types can be defined and mapped to the appropriate signals, and signal groups requiring length matching can be identified.

4.2.3.1 Routing Groups

High-speed devices with source-synchronous read and/or write interfaces require closely matched clock and data signals. Each source-synchronous interface was identified, and length matching tolerances were provided as necessary. Interfaces for high-speed devices connecting to an FPGA with special delay primitives were routed using a shortest length method. The signal lengths of high-speed devices connecting to an FPGA without delay primitives were closely monitored. The high-speed source-synchronous devices on the measurement board include:

- DDR SDRAM
- High-speed Analog-to-Digital Converter
- High-speed Digital-to-Analog Converter
- AsAP #1 and #2
- QDR-II SRAM
- DDR2 SDRAM (SODIMM)

The specifications for the high-speed digital circuits are shown in Sections F.2 to F.6 in the Appendix.

4.2.3.2 Planning for Power and Ground Planes

Power and ground planes are an important part of the printed circuit board. These planes serve many purposes, some of which include providing a return path for current and providing power supply voltages to ICs on the board. In the case of digital printed circuit boards, many unique or filtered power supplies are used, which require the power planes to be shared by creating unique polygons for each power supply. In addition, digital boards typically use devices with a ball-grid-array (BGA) package. Designing power and ground planes for these types of packages can be difficult, especially if consideration for the size and location of the planes is not done early enough in the design cycle. To ensure the proper location and correct amount of copper was available for the many power and ground planes, the size and shape of each plane was identified before the layout was initiated. Planning the size and shape of the planes can also provide helpful insight when performing the initial component placement. Figures 4.14 and 4.15 show the planned division of two power planes for the Virtex-5 FPGA power supplies used on the measurement board.

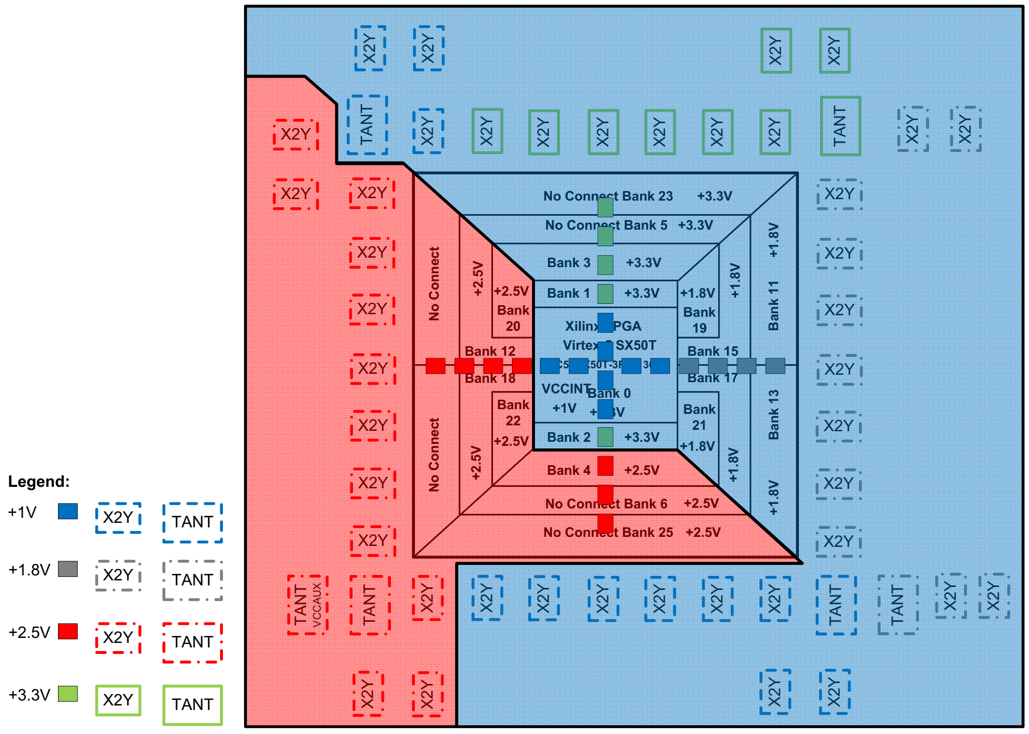

Figure 4.14: Recommended power plane shape for Virtex-5 +1.0V and +2.5V power supplies.

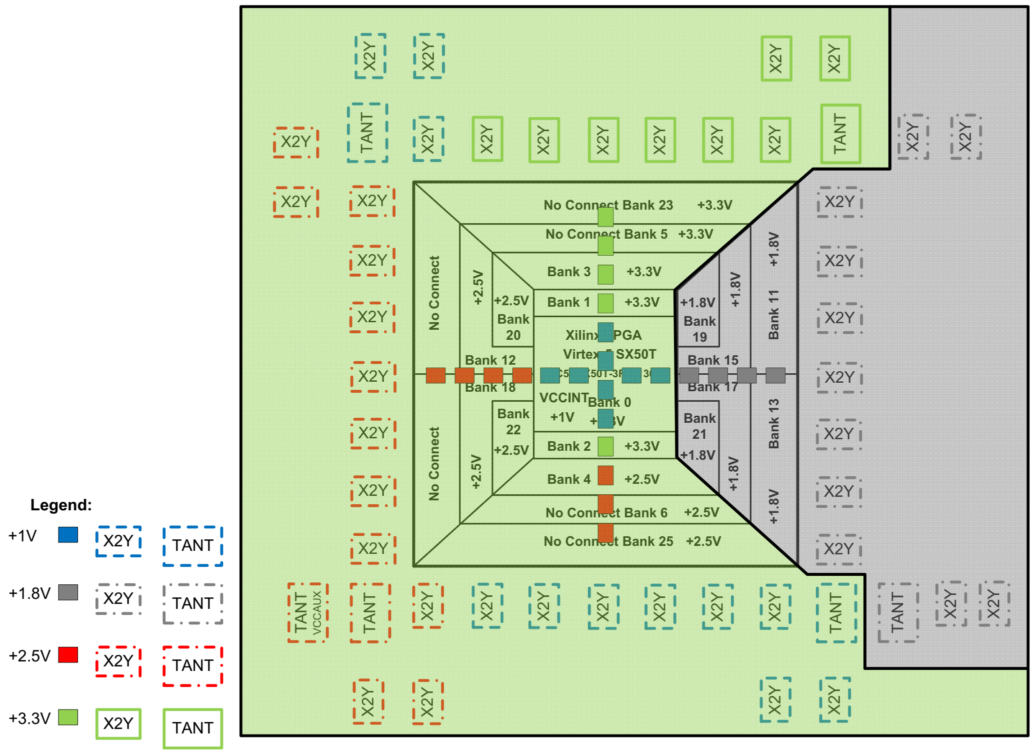

Figure 4.15: Recommended power plane shape for Virtex-5 +1.8V and +3.3V power supplies.

4.2.3.3 Net Type Assignments

High-speed printed circuit boards typically use devices that require a specific trace impedance for input and output signals. Section 4.2.1.2 describes the construction of both single-ended and differential microstrip and stripline transmission lines. Before the layout was started, each signal on the measurement board was assigned a net type based on whether it would be routed on external layers only or both internal and external layers. The PCB layout designer imported this data into the PADS Layout software, which allowed each trace to be routed with the proper width and spacing. The PADS Layout software performs design rule checking (DRC) throughout the layout process using the net type information.

In addition to transmission line net types, the power and ground net types also require definition based on the current requirements. Each power and ground signal on the measurement board was assigned a net type based on whether it was routed as a trace or as a plane of solid copper.

Table 4.5 describes the net type definitions used on the measurement board. The net type assignments for the measurement board are shown in Table G.1 of the Appendix.

Table 4.5: PC Board Net Types

Net Type Name	**Inner Outer**	**Trace Width**	**Trace Separation**	**Trace Impedance**	**Notes**
Default Net Types					
DEFAULT	I/O	6 mils	N/A	N/A	1. Default analog trace width.
Signal Net Types					
SE_FPGA	I/O	4 mils	N/A	N/A	1. Default FPGA trace width.
SE_50	I	7 mils	N/A	50 Ω	1. No area fill on adjacent layer.
	O	15 mils	N/A	50 Ω	
SE_50_O	I	NO_TRACE	N/A	N/A	1. No trace allowed on inner layer.
	O	15 mils	N/A	50 Ω	
DIFF_100	I	4 mils	4 mils	100 Ω	1. No area fill on adjacent layer.
	O	10 mils	6 mils	100 Ω	
DIFF_100_O	I	NO_TRACE	N/A	N/A	1. No trace allowed on inner layer.
	O	10 mils	6 mils	100 Ω	
Power Net Types					
GND_PLANE	I	PLANE	N/A	NA	1. Outer can be as narrow as 30 mils.
	O	50 mils	N/A	NA	
PWR_15MIL	I	15 mils	N/A	NA	1. Outer can be as narrow as 10 mils.
	O	15 mils	N/A	NA	2. Maximum Current: 200 mA.
PWR_25MIL	I	25 mils	N/A	NA	1. Outer can be as narrow as 15 mils.
	O	25 mils	N/A	NA	2. Maximum Current: 300 mA.
PWR_50MIL	I	50 mils	N/A	NA	1. Outer can be as narrow as 25 mils.
	O	50 mils	N/A	NA	2. Maximum Current: 500 mA.
PWR_100MIL	I	100 mils	N/A	NA	1. Outer can be as narrow as 40 mils.
	O	100 mils	N/A	NA	2. Maximum Current: 800 mA.

4.2.3.4 Digital Signal Routing

The central digital device on the measurement board is the Data Path FPGA, which has a 1760-pin, 42.5mm x 42.5mm BGA package; as a result the signal routing is very dense. The Data Path FPGA, which is a Xilinx Virtex-5 SX50T, contains delay primitives that allow the timing of each I/O to be adjusted. These delay primitives are known as IODELAY, and contain 64 delay taps, where each delay tap is equivalent to 78.125 ps (t_{tap}) [26]. Using the IODELAY primitives, the signals for each device can be routed to the FPGA using the shortest possible route; the signal lengths can be equalized inside of the Data Path FPGA.

Calculating IODELAY Taps for Data Path FPGA The number of IODELAY taps required to equalize the traces of a device interface can be calculated by taking into account the length of each trace and the estimated propagation delay on inner and outer layers of the measurement board. The propagation delay of traces on the measurement board are:

- Top/Bottom layer microstrip: $\sim$(130–140) $\frac{\text{ps}}{\text{in}}$
- Inner layer stripline: $\sim$(160–170) $\frac{\text{ps}}{\text{in}}$

The trace lengths for each high-speed signal connected to the Data Path FPGA are exported from the layout tool, and grouped according to each unique device. A Perl script was written to calculate the appropriate number of IODELAY taps for each signal. For each device, a signal is selected as the reference. In the case of synchronous device interfaces, the reference signal is typically the clock signal. The lengths of the traces are exported from the PADS Layout software in units of mils, where one mil is equivalent to one thousandth of an inch. The script uses an average value of 167 $\frac{\text{ps}}{\text{in}}$ for the propagation delay, which can be converted into units of $\frac{\text{ps}}{\text{mils}}$ by using Equation 4.1.

$$PropDelay = \left(\frac{167\text{ ps}}{1\text{ inch}}\right) \cdot \left(\frac{1\text{ inch}}{1000\text{ mils}}\right) = 0.167\ \frac{\text{ps}}{\text{mils}} \tag{4.1}$$

When the propagation delay is represented in the proper units, the total delay in picoseconds of the reference signal and remaining signals can be calculated using Equations 4.2 and 4.3.

$$t_{RefSig}[\text{ps}] = Length_{RefSig}[\text{mils}] \cdot PropDelay[\frac{\text{ps}}{\text{mils}}] \tag{4.2}$$

$$t_{Sig}\ [\text{ps}] = Length_{Sig}\ [\text{mils}] \cdot PropDelay\ [\frac{\text{ps}}{\text{mils}}] \tag{4.3}$$

Once the total delay for each trace has been calculated, the difference between the reference signal and each trace can be calculated using Equation 4.4 and the number of IODELAY taps for each trace can be estimated using Equation 4.5.

$$t_{Diff}\ [\text{ps}] = t_{RefSig}\ [\text{ps}] - t_{Sig}\ [\text{ps}] \tag{4.4}$$

$$Taps_{Sig} = \begin{cases} -1 \cdot \lfloor \frac{t_{Diff}\ [\text{ps}]}{t_{tap}\ [\text{ps}]} \rfloor, & \text{if } t_{Diff} < 0 \\ \lfloor \frac{t_{Diff}\ [\text{ps}]}{t_{tap}\ [\text{ps}]} \rfloor, & \text{otherwise.} \end{cases} \tag{4.5}$$

As can be seen in Equation 4.5, it is possible to calculate negative IODELAY taps. Since the delay cannot be adjusted in the negative direction, the number of estimated IODELAY taps must be normalized by adding the absolute value of the minimum IODELAY taps calculated (Taps$_{min}$). If a device interface has no negative IODELAY taps, then the number of IODELAY taps calculated using Equation 4.5 can be used directly.

$$Taps_{Norm} = \begin{cases} Taps_{Sig} + |Taps_{min}|, & \text{if } Taps_{min} < 0 \\ Taps_{Sig}, & \text{otherwise.} \end{cases} \tag{4.6}$$

Table 4.6 shows the IODELAY taps calculated for the high-speed ADC interface. The script does not distinguish between single-ended and differential signals. When the estimated number of IODELAY taps for the two complementary signals is different, the amount of added delay required for both signals should be averaged and rounded to the nearest multiple of IODELAY taps. For example, the script estimated a different number of IODELAY taps for the differential pair containing the signals: FPGA_ADC_DATA_P10 and FPGA_ADC_DATA_N10. Taking the average of both signals results in a delay -76.27642 ps, which can be rounded to -78.125 ps or -1 IODELAY tap. In this case, the number of normalized IODELAY taps for the complementary signals would be zero. The Perl script used to calculate the number of IODELAY taps for the high-speed ADC interface is shown in Appendix Chapter H.1. The IODELAY tap results for the remaining high-speed digital circuits are shown in Sections H.2 to H.6 in the Appendix.

Table 4.6: ADC Signal Delay Values

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps**$_{Sig}$	**Taps**$_{Norm}$
FPGA_ADC_DATA_RDY_P	5084.05	849.0364	1	0.0000000	0	1
FPGA_ADC_DATA_RDY_N	5084.41	849.0965	0	-0.060120	0	1
FPGA_ADC_OVR_P	4838.18	807.9761	0	41.060290	0	1
FPGA_ADC_OVR_N	4781.46	798.5038	0	50.532530	0	1
FPGA_ADC_DATA_P15	4181.28	698.2737	0	150.76259	1	2
FPGA_ADC_DATA_N15	4161.74	695.0106	0	154.02577	1	2
FPGA_ADC_DATA_P14	4911.44	820.2105	0	28.825870	0	1
FPGA_ADC_DATA_N14	4933.09	823.8260	0	25.210320	0	1
FPGA_ADC_DATA_P13	4431.04	739.9837	0	109.05267	1	2
FPGA_ADC_DATA_N13	4418.19	737.8377	0	111.19862	1	2
FPGA_ADC_DATA_P12	5105.24	852.5751	0	-3.538730	0	1
FPGA_ADC_DATA_N12	5028.53	839.7645	0	9.2718400	0	1
FPGA_ADC_DATA_P11	4849.78	809.9133	0	39.123090	0	1
FPGA_ADC_DATA_N11	4929.46	823.2198	0	25.816530	0	1
FPGA_ADC_DATA_P10	5500.59	918.5985	0	-69.56218	0	1
FPGA_ADC_DATA_N10	5581.00	932.0270	0	-82.99065	-1	0
FPGA_ADC_DATA_P9	4717.87	787.8843	0	61.152060	0	1
FPGA_ADC_DATA_N9	4737.86	791.2226	0	57.813730	0	1
FPGA_ADC_DATA_P8	5423.00	905.6410	0	-56.60465	0	1
FPGA_ADC_DATA_N8	5384.44	899.2015	0	-50.16513	0	1
FPGA_ADC_DATA_P7	4994.17	834.0264	0	15.009960	0	1
FPGA_ADC_DATA_N7	5013.08	837.1844	0	11.851990	0	1
FPGA_ADC_DATA_P6	4770.75	796.7153	0	52.321100	0	1
FPGA_ADC_DATA_N6	4731.44	790.1505	0	58.885870	0	1
FPGA_ADC_DATA_P5	4352.98	726.9476	0	122.08869	1	2
FPGA_ADC_DATA_N5	4330.28	723.1568	0	125.87959	1	2
FPGA_ADC_DATA_P4	4684.41	782.2965	0	66.739880	0	1
FPGA_ADC_DATA_N4	4700.57	784.9952	0	64.041160	0	1
FPGA_ADC_DATA_P3	4282.20	715.1274	0	133.90895	1	2
FPGA_ADC_DATA_N3	4306.90	719.2523	0	129.78405	1	2
FPGA_ADC_DATA_P2	4825.42	805.8451	0	43.191210	0	1
FPGA_ADC_DATA_N2	4894.90	817.4483	0	31.588050	0	1
FPGA_ADC_DATA_P1	4284.51	715.5132	0	133.52318	1	2
FPGA_ADC_DATA_N1	4301.71	718.3856	0	130.65078	1	2
FPGA_ADC_DATA_P0	4725.53	789.1635	0	59.872840	0	1
FPGA_ADC_DATA_N0	4712.04	786.9107	0	62.125670	0	1

4.2.4 Power Supply Design

The measurement board is powered by a Power-One 125 W, +12 V AC-DC power supply. The +12 V DC voltage is filtered and split into two paths: analog and digital. The majority of the digital power supplies are generated using a single stage of DC/DC converters. The analog power supply voltages are generated using two stages; one is generated by a DC/DC converter, and the second is generated by a low-dropout linear regulator. A block diagram of the power supply sub-system is shown in Figure 4.16.

4.2.4.1 Power Supply Synchronization

The digital power supplies are typically generated using a DC/DC converter with a switching frequency of less than 1 MHz, which can couple onto high-speed signals. As a result, the digital power supplies are distributed on two solid planes of copper; the planes are stacked between two solid ground planes, thus reducing power supply noise [25]. In addition to the layer stack of the digital power supplies, four solid ground planes are used to provide isolation for the high-speed digital signals. Furthermore, the DC/DC converters used on the measurement board contain a synchronization input that allows multiple converters to be synchronized together using a common clock running at a frequency of 312.5 kHz. The synchronization clock is generated from a 10 MHz reference clock using a chain of 5 flip-flops in divide-by-two configuration. The resulting clock is then driven into a chain of flip-flops to create a unique clock phase for each DC/DC converter. The circuit, shown in Figure 4.17, is implemented in a Xilinx CPLD. Synchronizing the DC/DC converters helps to simplify the EMI noise suppression and reduce the overall capacitance requirements.

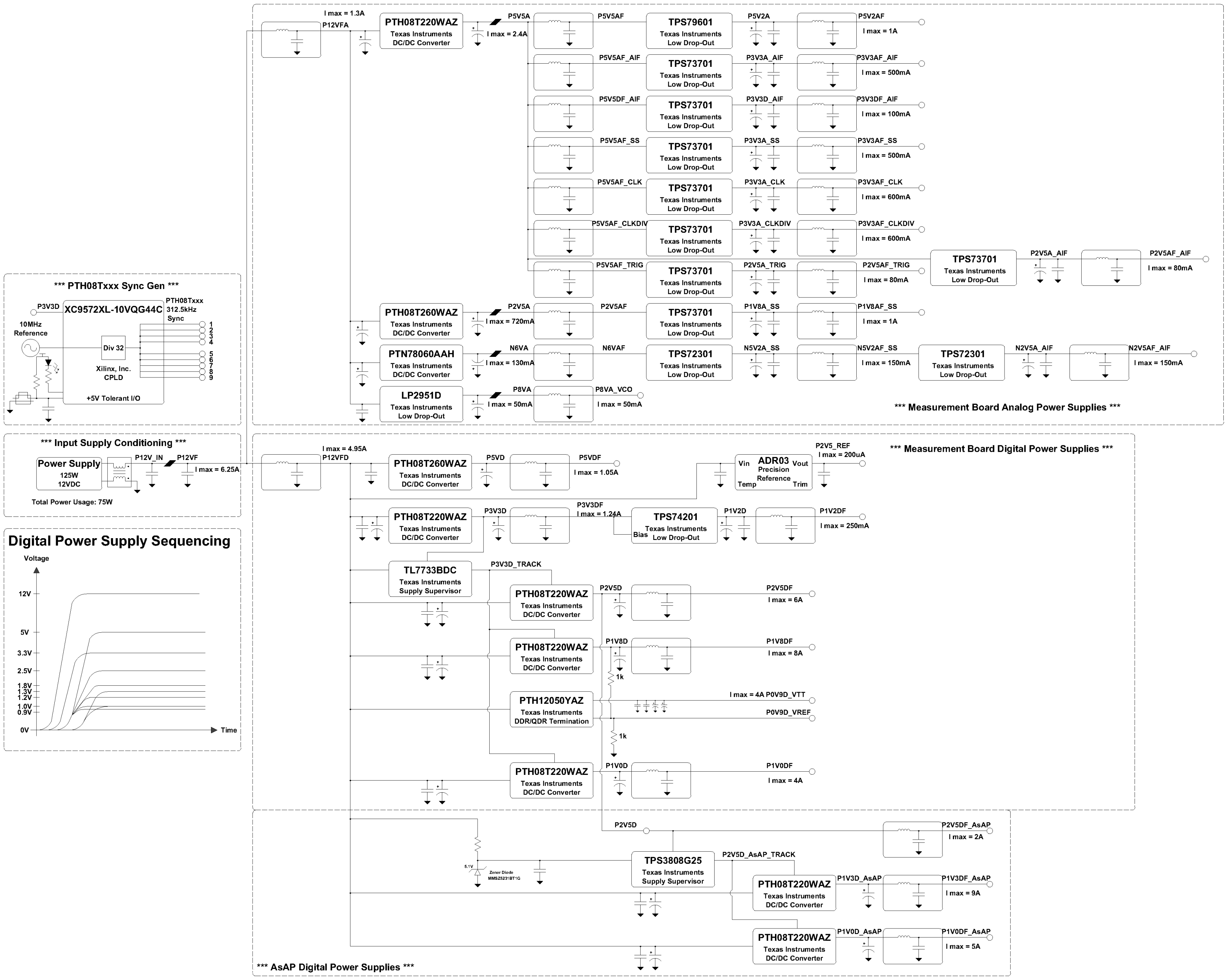

Figure 4.16: Power Supply Sub-System Block Diagram

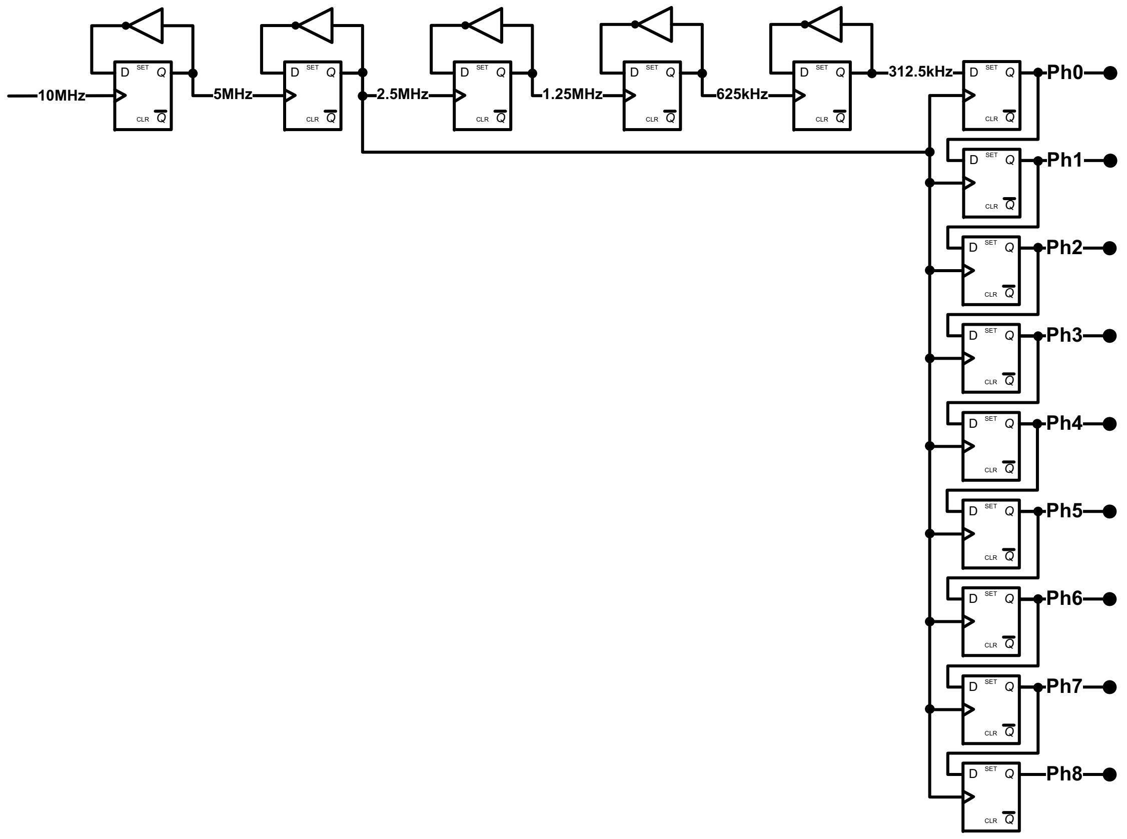

Figure 4.17: DC/DC Converter Switching Frequency Generation.

4.2.4.2 Power Supply Filtering

The DC/DC converters operate with a switching frequency of 312.5 kHz, and therefore a power supply filter is required before the voltage is driven into a low-dropout linear regulator. A low-pass filter, shown in Figure 4.18, was designed with a −3 dB frequency of 32 kHz, as can be seen in Figure 4.19. The goal of the power supply filter was to attenuate the signal to an amplitude voltage less than 20 mV.

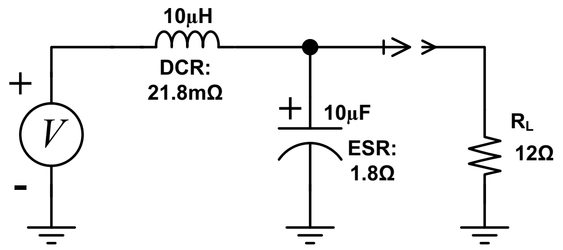

Figure 4.18: DC/DC Converter Output Filter

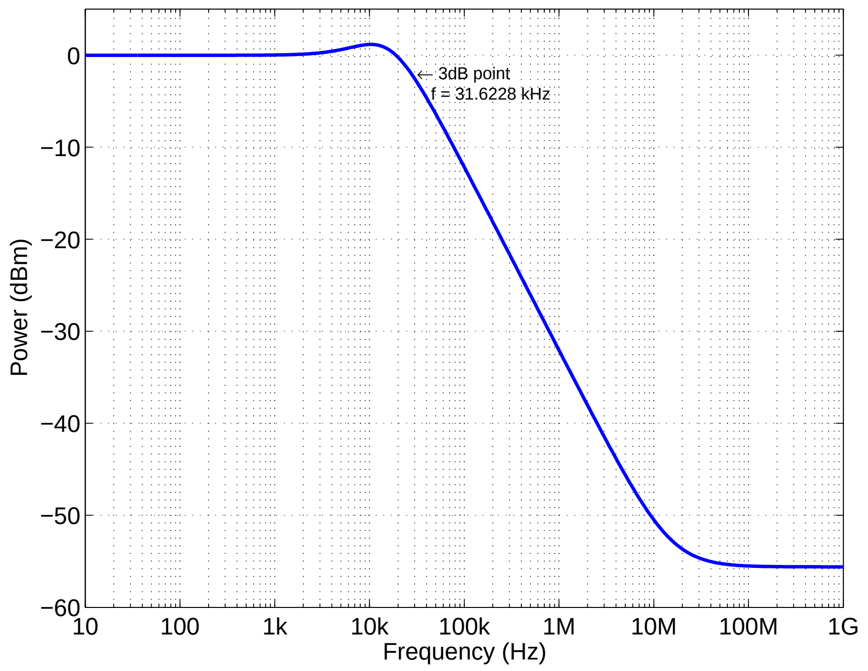

Figure 4.19: Power Supply Filter: Frequency Response

The power supply filter was simulated using LTSpice. The input of the filter was stimulated with a 312.5 kHz waveform with an amplitude of 100 mV. The time domain plot, shown in Figure 4.20, shows the output of the filter at an amplitude voltage of 15 mV, which is adequate to drive the input of a linear regulator, thus the signal is sufficiently attenuated.

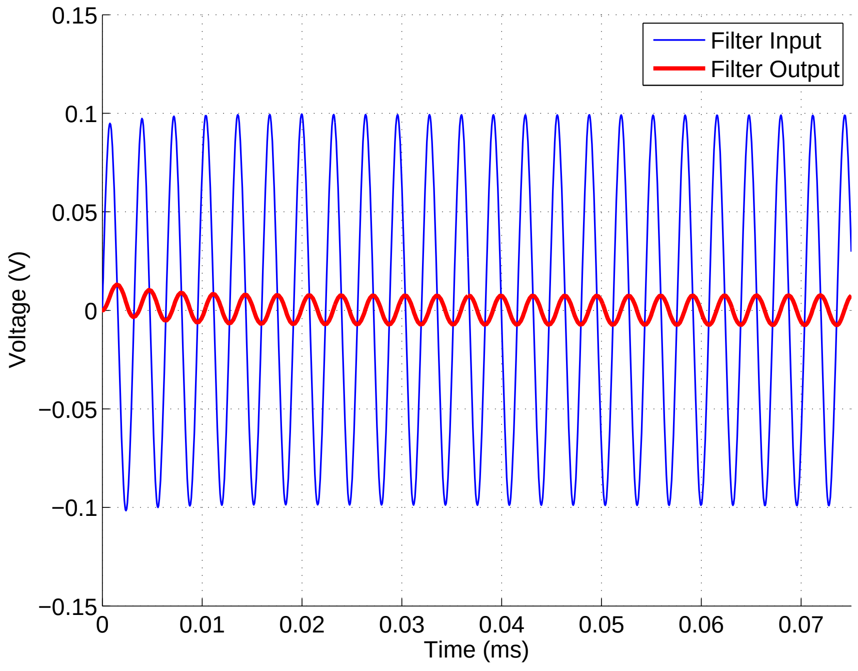

Figure 4.20: Power Supply Filter I/O: Time Domain Waveforms

4.2.4.3 Power Supply Sequencing

Power supply sequencing is a common requirement of digital circuits. Several circuits on the measurement board require their power supplies to turn on in a particular sequence. In some cases, power supply sequencing is achieved by the natural order of supply generation. For example, the sequencing for a circuit that requires +3.3 V to turn on before +1.2 V can be achieved by generating +1.2 V from +3.3 V. In other cases, power supply voltage supervisors are required to delay the turn on of supplies by a precise amount of time.

At power-up, the Control FPGA is configured by a serial peripheral interface (SPI) programmable read-only memory (PROM), which is powered off of +3.3 V. The configuration I/O bank of the Control FPGA is also powered off of +3.3 V. The Control FPGA defaults to a master SPI configuration mode; therefore as soon as it powers up, the configuration process will be initiated and it will begin to drive the configuration clock of the SPI PROM. In order for the configuration process to complete successfully, the +3.3 V supply must turn on before the core voltage (+1.2 V) of the Control FPGA, which in turn must power up before the remaining digital power supplies. If this turn-on sequence is violated, the configuration process could potentially fail, or could fail at a random time.

The measurement board uses a Texas Instruments TL7733B single supply voltage supervisor which is powered off of the +12 V supply and senses when the +3.3 V supply crosses a voltage threshold of +3.08 V. A slowly ramping signal is generated, which is driven into the tracking pin of the DC/DC converters powering the digital circuits. A timing capacitor (C$_T$) is used to set the turn on delay. The amount of delay inserted can be calculated using Equation 4.7. The timing capacitor is set such that the +3.3 V power supply will turn on approximately 50 ms before the other digital power supplies being generated by a DC/DC converter.

$$t_{delay}[\mathrm{s}] = 2.6e4 \cdot C_T[\mathrm{F}] = 2.6e4 \cdot 2.2\ \mu\mathrm{F} = 0.0572\ \mathrm{s} \tag{4.7}$$

The Control FPGA core voltage is generated from the +3.3 V power supply using a Texas Instruments TPS74201 single output low-dropout (LDO) linear regulator with programmable soft-start. Using a timing capacitor (C_{SS}), the TPS74201 linear regulator can delay the turn on of its output. The soft-start time can be calculated using Equation 4.8.

$$t_{SS}\ [\mathrm{s}] = \frac{V_{REF}\ [\mathrm{V}] \cdot C_{SS}\ [\mathrm{F}]}{I_{SS}\ [\mathrm{A}]} \tag{4.8}$$

From the TPS74201 data sheet, the soft-start current (I_{SS}) is equal to 0.73 μA and the reference voltage (V_{REF}) is equal to 0.8 V. The desired soft-start time for the +1.2 V power supply is 10 ms. Solving for the soft-start capacitor variable C_{SS} in Equation 4.8 yields Equation 4.9, which results in a capacitance of 9.125 nF.

$$C_{SS}\ [\mathrm{F}] = \frac{t_{SS}\ [\mathrm{s}] \cdot I_{SS}\ [\mathrm{A}]}{V_{REF}\ [\mathrm{V}]} = \frac{10\ \mathrm{ms} \cdot 0.73\ \mu\mathrm{A}}{0.8\ \mathrm{V}} = 9.125e-9\ \mathrm{F} \approx 10\ \mathrm{nF} \tag{4.9}$$

The soft start capacitor value was rounded to the nearest standard value of 10 nF, which results in a soft start time of approximately 10.96 ms as shown in Equation 4.10.

$$t_{SS} = \frac{0.8\ \mathrm{V} \cdot 10\ \mathrm{nF}}{0.73\ \mu\mathrm{A}} = 0.01096\ \mathrm{s} \tag{4.10}$$

The desired digital power supply sequencing of the measurement board is shown in Figure 4.21.

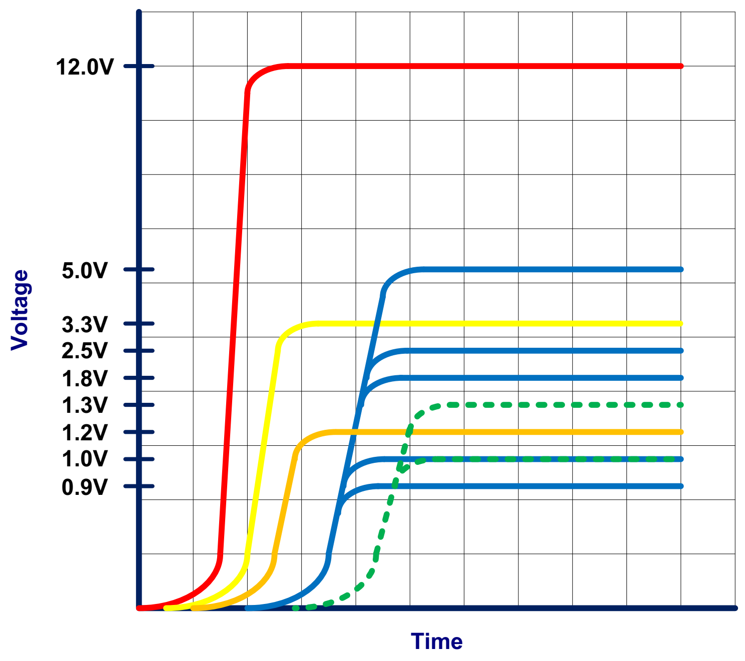

Figure 4.21: Desired Digital Power Supply Sequence

4.2.4.4 Power Consumption

Estimating power supply current consumption is an essential part of printed circuit board design. If the current requirements for each device are not accounted for, then the design could fail to operate correctly or even turn on successfully. To estimate the current requirements of the measurement board the maximum current requirements for each device were tabulated, and the requirements for each power supply were summed. Table 4.7 describes the worst case power consumption for the measurement board.

Table 4.7: Total Power Supply Current Usage

Total Power Supply Current Usage		
ITEM	**+12V**	**Total Power (W)**
Analog Power Supply Current Usage	1.322989	15.875872
Digital Power Supply Current Usage	4.933585	59.203015
TOTAL CURRENT (A)	6.256574	
TOTAL POWER (W)	75.078888	75.078888

The analog power supply current usage was calculated using Table 4.8. The digital power supply current usage was calculated using Tables 4.9 and 4.10.

Table 4.8: Analog Power Supply Current Usage

Analog Power Supply Current Usage					
ITEM	**+12V**	**+5.5V**	**+2.5V**	**-6V**	**Total Power (W)**
THS4302 Fixed-Gain Op-Amp 14dB	0.130	0.284	0	0	1.56
THS4509 Wide-Band Differential Op-Amp 8 dB	0.043	0.095	0	0	0.52
ADS5463 12-bit, 500 MS/s ADC	0.333	0.725	0	0	3.99
SN74LVC2G125 x12	0.0000260	0.0000567	0	0	0.000312
DAC5682Z 16-bit, 1 GS/s DAC	0.288	0.3	0.720	0	3.45
OPA695 Op-Amp	0.128	0.142	0	-0.125	1.53
Vectron 10 MHz TCXO Oscillator	0.003	0.006	0	0	0.033
On Semiconductor - NB6L11MMNG	0.021	0.045	0	0	0.2475
Micrel - SY58017UMG	0.019	0.042	0	0	0.231
Micrel - SY58017UMG	0.019	0.042	0	0	0.231
Micrel - SY58017UMG	0.019	0.042	0	0	0.231
Micrel - SY58011UMG	0.026	0.057	0	0	0.3135
Micrel - SY58608UMG	0.022	0.048	0	0	0.264
Texas Instruments - ONET1191PRGTT	0.013	0.029	0	0	0.1617
On Semiconductor - MC10EP89DTG	0.030	0.065	0	0	0.3564
On Semiconductor - NB4N527S	0.015	0.032	0	0	0.1749
On Semiconductor - MC10EP89DTG	0.028	0.060	0	0	0.33
Analog Devices AD9516	0.138	0.3	0	0	1.65
Micrel - SY58601UMG - Trigger Out	0.013	0.027	0	0	0.15
SN65LVDS1 (x2)	0.004	0.009	0	0	0.05
UMC: UMX-244-B14	0.033	0	0	0	0.4
TOTAL CURRENT (A)	1.322989	2.350159	0.72	-0.125	
TOTAL POWER (W)	15.875872	12.925872	1.8	0.75	15.875872

Table 4.9: Digital Power Supply Current Usage

Digital Power Supply Current Usage		
ITEM	**+12V**	**Total Power (W)**
XC5VSX50T-3FF1136C (Virtex 5 SX50T)	1.531	18.3666
DDR2 SDRAM SODIMM (MT16HTF25664H)	0.620	7.434
QDR-II SRAM (K7R323684C-EC250)	0.214	2.565
AsAP DSP IC (x2)	1.617	19.4
XC3S1400A-4FG484 (Spartan 3A)	0.088	1.052
M25P64-VMF6TP (SPI Flash 64 Mb)	0.006	0.066
TPS3823-25DBVT (Reset uChip)	0.000006875	0.000083
TPS3823-25DBVT (Config Hold-Off uChip)	0.000006875	0.000083
CSX750FB (14.7456 MHz)	0.004	0.0495
MT46V32M16BN-6 (DDR SDRAM)	0.141	1.6875
SN74LVC1G08 (Single 2-Input AND Gate)	0.000002750	0.000033
SN74LVC1G32 (Single 2-Input OR Gate)	0.000002750	0.000033
CP2102 (x2)	0.044	0.528
MicroSD Card (x2)	0.168	1.98
ADR03 +2.5V Precision Ref for V5 SM	0.01	0.12
AMC6821 (x2)	0.006	0.066
XC9572XL-10VQ44	0.011	0.132
CCLD-033-50-100.00 (x2)	0.035	0.4224
FTDI FT245BL (x2)	0.021	0.25132
User Interface Board	0.417	5.0
Temperature Sensor (TMP125) (x6)	0.00011	0.00132
SN65LVDT2DBVR (TTL-to-LVDS)	0.00275	0.033
TPS3808G25DRV (x2)	0.000012	0.000144
TL7733BCDR	0.004	0.048
TOTAL CURRENT (A)	4.933585	
TOTAL POWER (W)	59.203015	59.203015

Table 4.10: Digital Power Supply Current Usage Detail

Digital Power Supply Current Usage Detail										
ITEM	**+0.9V**	**+1V**	**+1V AsAP**	**+1.2V**	**+1.25Vref**	**+1.3V**	**+1.8V**	**+2.5V**	**+3.3V**	**+5V**
XC5VSX50T-3FF1136C	1.71	3.737	0	0	0	0	2.723	3.138	0.104	0
MT16HTF25664H DDR2 SDRAM	1.26	0	0	0	0	0	3.5	0	0	0
K7R323684C-EC250	0.45	0	0	0	0	0	1.2	0	0	0
AsAP DSP IC (x2)	0	0	4	0	0	8	0	2.0	0	0
XC3S1400A-4FG484	0	0	0	0.134	0	0	0	0.173	0.139	0
M25P64-VMF6TP	0	0	0	0	0	0	0	0	0.02	0
TPS3823-25DBVT (x2)	0	0	0	0	0	0	0	0	0.00005	0
CSX750FB (14.7456 MHz)	0	0	0	0	0	0	0	0	0.015	0
MT46V32M16BN-6 (DDR SDRAM)	0	0	0	0	0.45	0	0	0.45	0	0
SN74LVC1G08 (Single AND Gate)	0	0	0	0	0	0	0	0	0.00001	0
SN74LVC1G32 (Single OR Gate)	0	0	0	0	0	0	0	0	0.00001	0
CP2102 (x2)	0	0	0	0	0	0	0	0	0.16	0
MicroSD Card (x2)	0	0	0	0	0	0	0	0	0.6	0
ADR03 +2.5V Precision Ref	0	0	0	0	0	0	0	0	0	0
AMC6821 (x2)	0	0	0	0	0	0	0	0	0.02	0
XC9572XL-10VQ44	0	0	0	0	0	0	0	0	0.04	0
CCLD-033-50-100.00 (x2)	0	0	0	0	0	0	0	0	0.128	0
FTDI FT245BL (x2)	0	0	0	0	0	0	0	0	0.0004	0.05
User Interface Board	0	0	0	0	0	0	0	0	0	1.0
Temperature Sensor (TMP125) (x6)	0	0	0	0	0	0	0	0	0.0004	0
SN65LVDT2DBVR	0	0	0	0	0	0	0	0	0.01	0
TPS3808G25DRV (x2)	0	0	0	0	0	0	0	0	0	0
TL7733BCDR	0	0	0	0	0	0	0	0	0	0
TOTAL CURRENT (A)	3.42	3.737	4	0.134	0.45	8	7.423	5.7614	1.23687	1.05
TOTAL POWER (W)	3.078	3.737	4	0.161	0.5625	10.4	13.361	14.4035	4.081671	5.25

4.3 Verification

The final version of the component placement can be seen in Figure 4.22.

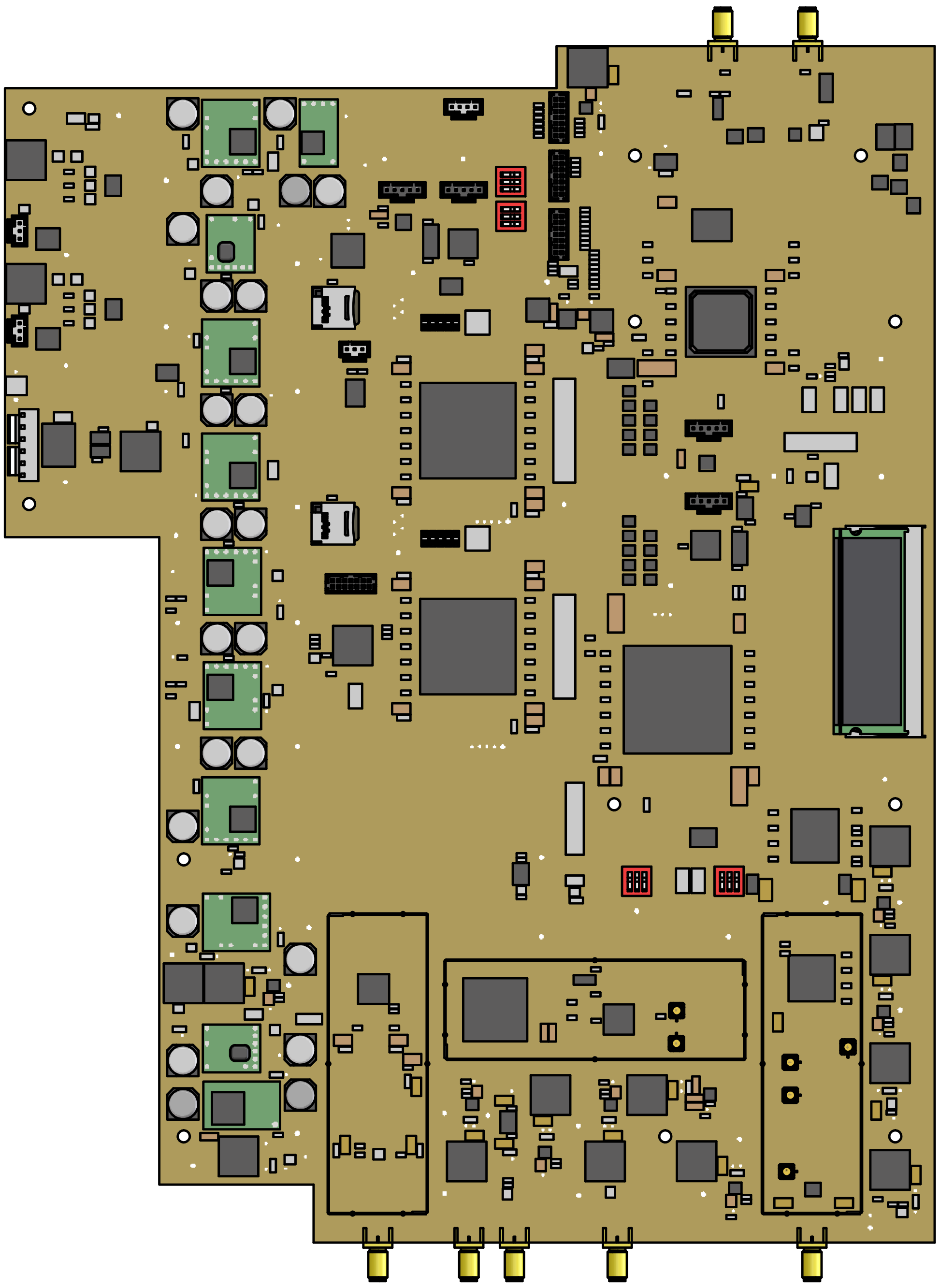

Figure 4.22: Measurement Board Top View

4.3.1 Printed Circuit Board Turn-on

Overall, the PCB turn-on phase was successful. Initially, the power supply sub-system, Control FPGA sub-system, Data Path FPGA sub-system, and high-speed sub-systems were functioning properly. During the PCB turn-on phase of the project, two bugs were found in the schematic design, which required PCB modifications for correct operation. In addition, two bugs were discovered in the PCB layout library used with the PADS Layout software and trigger output circuit. The power supply consumption, digital power supply turn-on sequence, and power supply filters were measured to verify proper operation. Sections 4.3.1.1, 4.3.1.2, 4.3.1.3, and 4.3.1.4 describe the solution for the Data Path FPGA configuration, high-speed DAC output, SODIMM layout decal, and trigger output bugs, respectively.

4.3.1.1 Data Path FPGA Configuration Modification

The measurement board was designed with several options for configuring the Data Path FPGA including slave-serial daisy chain configuration, SPI serial daisy chain configuration, and JTAG daisy chain configuration. For two of the three configuration methods available, the Control FPGA is responsible for configuring the Data Path FPGA. The slave-serial daisy chain configuration method was used to configure the Data Path FPGA, and required 4 resistors to be loaded onto the PCB. The configuration resistors loaded were:

- R569: CUST_INIT_B
- R568: CUST_CFG_DONE
- R567: CUST_CCLK
- R566: CUST_PROG_B

4.3.1.2 High-Speed Digital-to-Analog Converter Modification

The measurement board signal source output uses a Texas Instruments DAC5682Z dual-channel, 16-bit, 1GS/s digital-to-analog converter. The original intent of the signal source design was to use the dual-channel DAC in single channel mode with channel A. The DAC channel was changed during the layout phase of the measurement board design from channel A to channel B for layout reasons. Using channel B allowed the analog signal path of the signal source to be routed without the use of vias. During the turn-on phase of the signal source circuit there was no data observed on the channel B DAC output. After reading through the data sheet, it was discovered that when the DAC5682Z is used in single channel mode, channel A must be used. Measuring the unused terminated output of the DAC5682Z revealed the desired data being played back through the signal source circuit. Fortunately, the unused channel of the DAC5682Z was terminated into 50 Ω and channel B was driven into the reconstruction filter through 0 Ω resistors. The following PCB modification was made:

1. Removed resistors R88, R89, R530, and R531.
2. Soldered a wire from pad 1 of resistor R88 to pad 2 of resistor R531.
3. Soldered a wire from pad 1 of resistor R89 to pad 2 of resistor R530.

The length of the wire was cut to the shortest possible length, approximately $\frac{3}{4}$ inch, to minimize signal degradation of the DAC output. The crudeness of this modification was acceptable due to the range of frequency operation, DC–120 MHz. The effectiveness of this modification would be lessened if the frequency of operation was pushed to the maximum capability of the DAC5682Z. After the PCB modification was made, the Data Path FPGA was able to drive data into the DAC5682Z and out of the signal source circuit across the entire range of DC–120 MHz.

4.3.1.3 DDR2 SDRAM SODIMM Socket Modification

During the assembly phase of the measurement board it was discovered that the layout footprint for the DDR2 SDRAM SODIMM socket (U42) was incorrect. The SODIMM socket footprint consists of two rows of 100 pads which are offset in the x- and y-dimensions. The incorrect footprint was not properly offset in either dimension. To fix this problem, an interposer board was designed with the incorrect footprint on the bottom side and the correct footprint on the top side. The layer stackup for the 4-layer interposer board is shown in Table 4.11.

Table 4.11: 4-Layer FR408 Stackup

Layer Name	**Signal or Plane**	**Stackup**
TOP	Signal	Routing
SIDE2	Plane	Ground
SIDE3	Plane	Power
BOTTOM	Signal	Routing

The interposer board, the construction of which is shown in Figure 4.23, is fabricated using Isola FR408 laminate. Its dimensions are outlined in Table 4.1.

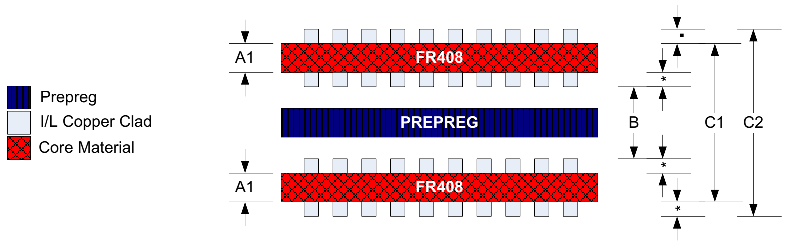

Figure 4.23: 4-Layer FR408 Core Construction

Table 4.12: 4-Layer FR4 Core Construction

4-Layer FR4 Core Construction				
Construction Type	**A1 (inches)**	**B (inches)**	**C1 (inches)**	**C2 (inches)**
FR408	.0135 ± .002	.027 ± .001	.0504-.0604	.0518-.0618

* - Copper thickness unless otherwise specified on inner layers: $\frac{1}{2}$ oz. Cu (0.0007”).
* - Copper thickness unless otherwise specified on outer layers: $\frac{1}{2}$ oz. Cu (0.0007” before plating).
A1 - FR408 Core thickness
B - Pressed thickness of prepreg
C1 - Overall finished board thickness substrate-to-substrate
C2 - Overall finished board thickness plated metal to plated metal. In addition, some surface coatings (i.e., HAL) can add up to 0.002” of solder per side.
Soldermask (also not specified) typically adds about 0.001” per side but can add up to 0.004” per side in extreme examples.

The assembly top diagram of the interposer board is shown in Figure 4.24. The interposer board is assembled onto the measurement board using a process similar to that of QFN packages. Solder paste is applied to the incorrect footprint on the measurement board, and the two drill holes on the ends of the SODIMM socket are used to align the interposer board with the pads on the measurement board. Once the boards are properly aligned, heat is applied to the area of U42 until the solder has melted, at which point the heat is removed. The final step is to verify the alignment of the two boards by X-Raying the boards. The SODIMM socket is then soldered to the SODIMM interposer board.

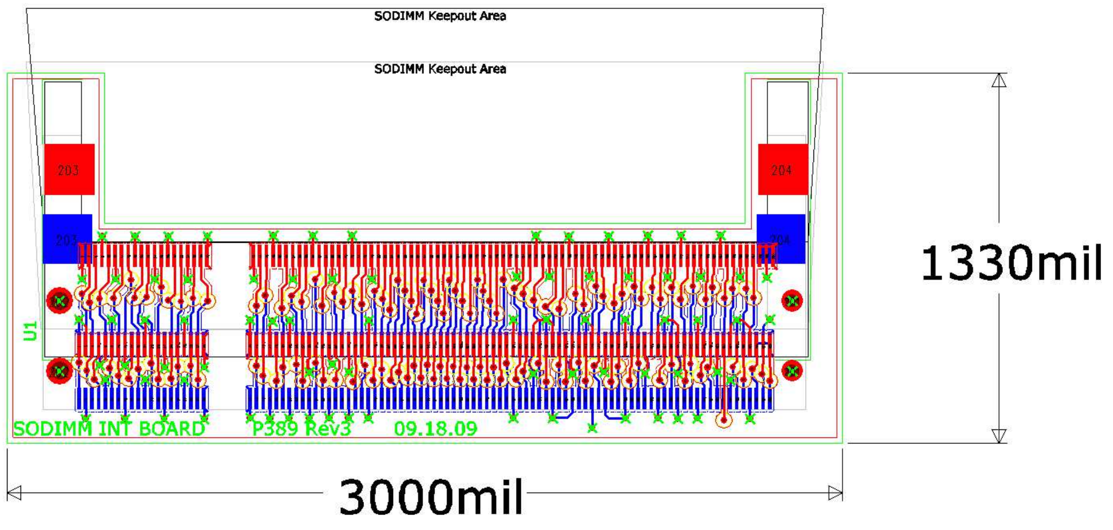

Figure 4.24: SODIMM Interposer Board Assembly Top Diagram.

4.3.1.4 Trigger Output Modification

Instrument I/O are typically protected against electrostatic discharge (ESD), which can occur as a result of excess static build-up on the user. A common method of protecting against ESD is to use a diode connected to sensitive I/O. An On-Semiconductor low capacitance diode, NUP4301MR6T1, was used to protect the measurement board trigger output, which is generated by a circuit in the Data Path FPGA operating at 31.25 MHz. Figure 4.25 shows the rising edge of the trigger pulse with the ESD diode.

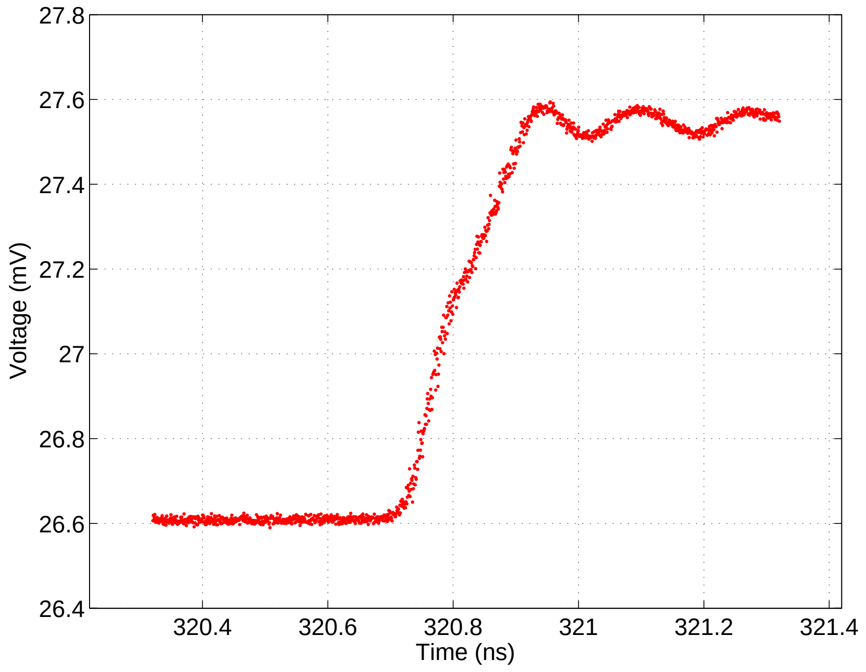

Figure 4.25: Rising edge of trigger output pulse with ESD diode, which has a capacitance to ground value of 1.6 pF.

While the diode used on the trigger output will protect against ESD damage, the capacitance to ground for the diode had an adverse effect on the trigger waveform quality. As can be seen in Figure 4.25, the added capacitance caused a reflection which appeared at the center of the rising edge and resulted in a rise time of 162.22 ps. By removing the ESD diode from the trigger output circuit, the reflection was removed resulting in a rise time of 77.78 ps. The rising edge of the trigger pulse without the ESD diode is shown in Figure 4.26. In this case, a compromise was made for performance over ESD protection. During normal operation, the trigger circuit with the ESD diode could potentially result in a false trigger of an oscilloscope. Figure 4.27 shows the two trigger output pulses together to emphasize the improvement in performance. The measurement setup is shown in Figure 4.28.

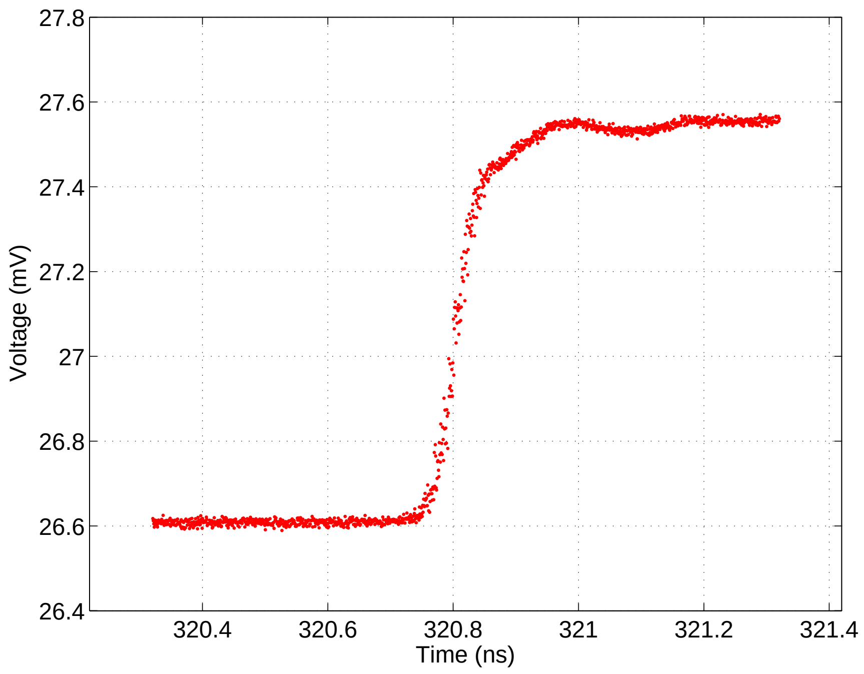

Figure 4.26: Rising edge of trigger output pulse without ESD diode

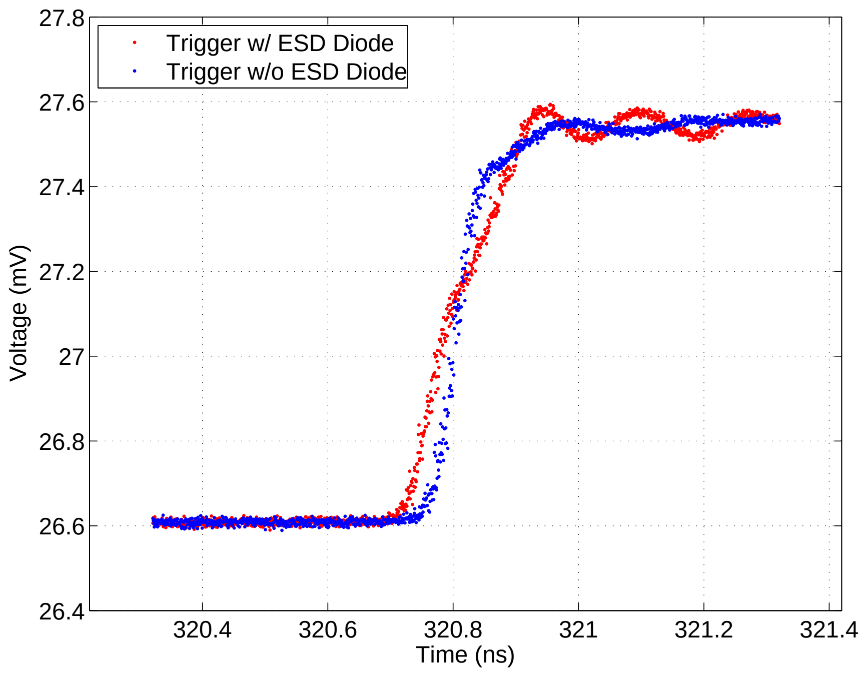

Figure 4.27: Rising edge of trigger output pulse with and without ESD diode

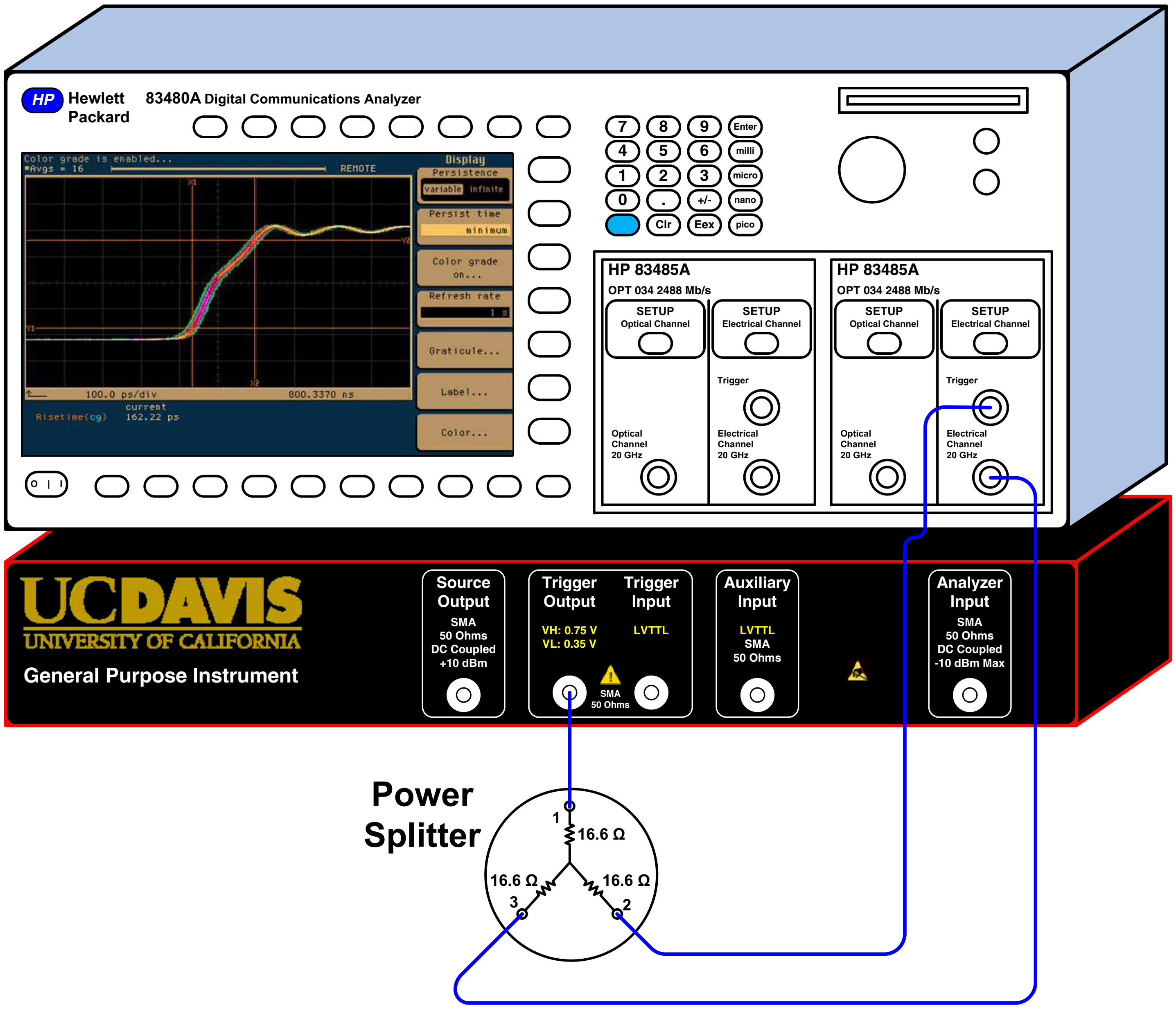

Figure 4.28: Trigger Output Measurement Setup

4.3.1.5 Power Supply Filtering

A Tektronix TDS3054C oscilloscope was used to verify the performance of the power supply circuit. A passive, high-impedance oscilloscope probe was used to measure the power supply signals with the oscilloscope in AC coupling mode. The measurement setup is shown in Figure 4.30. As can be seen in Figure 4.29, the power supply sufficiently attenuates the noise on the DC/DC converter output voltage to an amplitude voltage of 14.7 mV.

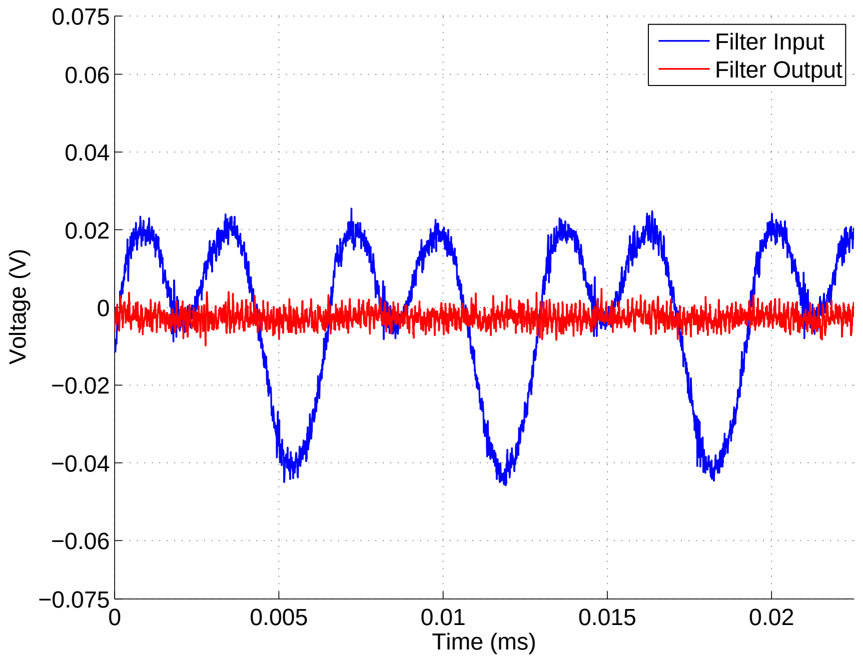

Figure 4.29: Power Supply Filter: Time Domain Measurement

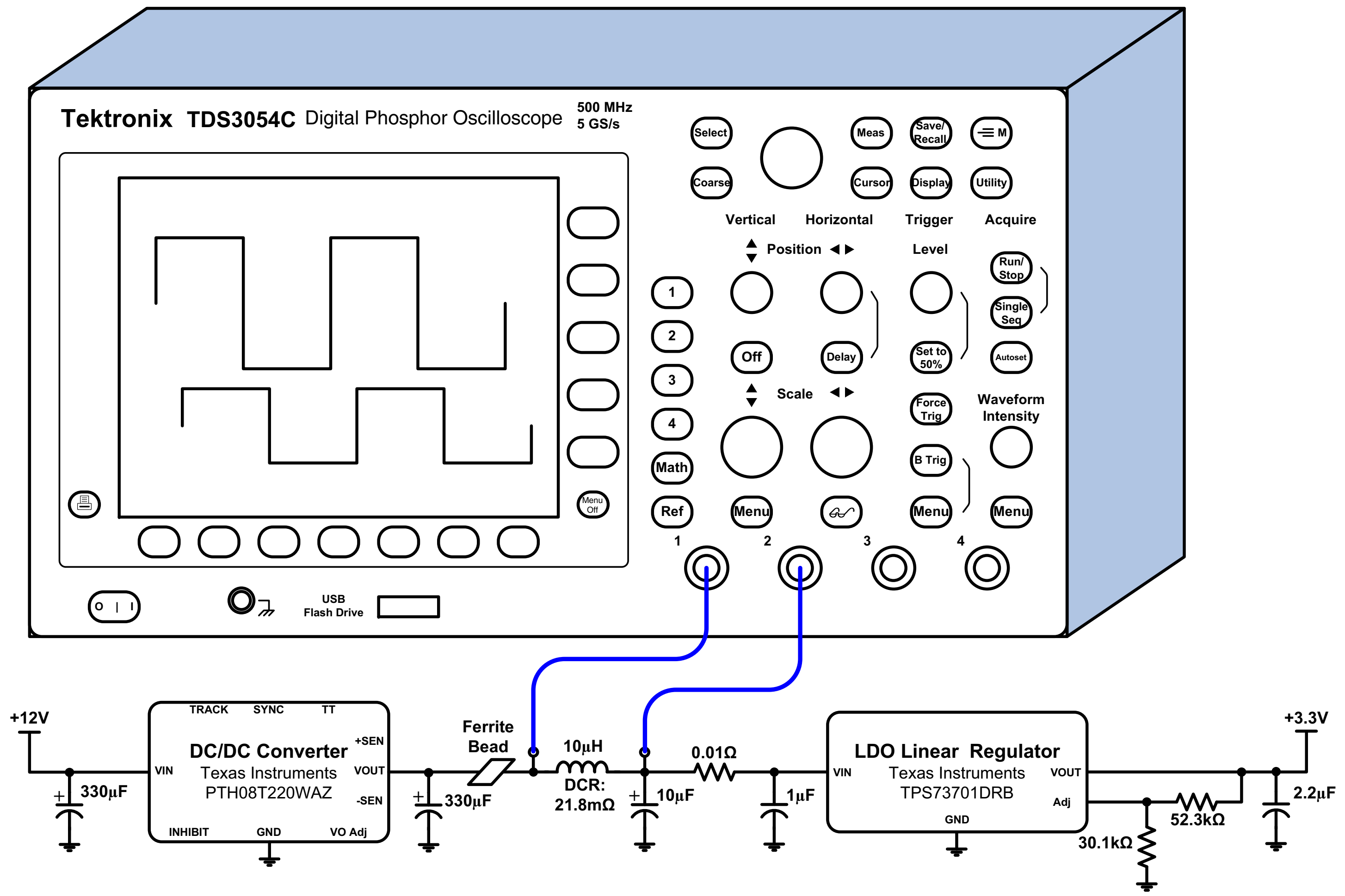

Figure 4.30: Power Supply Measurement Setup

4.3.1.6 Power Supply Sequencing

A Tektronix TDS3054C oscilloscope was used to verify the power-on sequence of the digital power supply circuits. A passive, high-impedance oscilloscope probe was used to measure the power supply signals with the oscilloscope in DC coupling mode. The oscilloscope was triggered on the track signal generated by the Texas Instruments TL7733BCDR supply supervisor, which is enabled when the sense voltage; +3.3 V, rises above +3.08 V. The turn-on rate of the track signal is set by an RC time constant as described in Section 4.2.4.3. The measurement setup is shown in Figure 4.33.

The measured power supply turn-on sequence is incorrect, but functional. The +1.2 V power supply, shown in orange in Figure 4.31, turns on approximately 0.745 ms after the +3.3 V supply has ramped up.

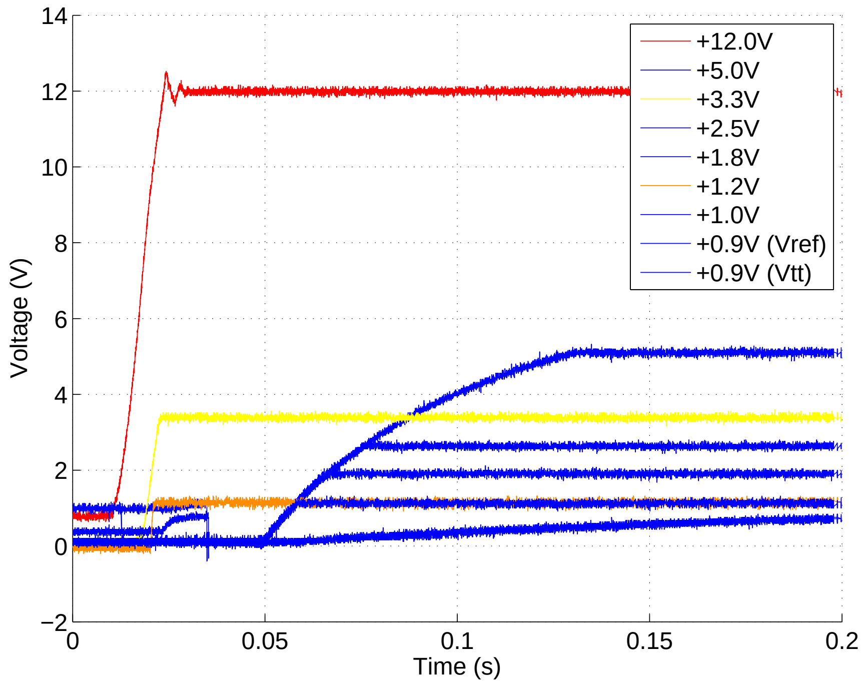

Figure 4.31: Measured Digital Power Supply Sequence

An incorrect soft-start capacitor value of 680 pF was connected to the soft-start pin of the TPS74201 linear regulator. By replacing the 680 pF capacitor with the soft-start capacitor value calculated in Equation 4.9, the proper power supply turn-on sequence was achieved. Figure 4.32 shows the measured power supply turn-on sequence with the new C_{SS} value of 10 nF.

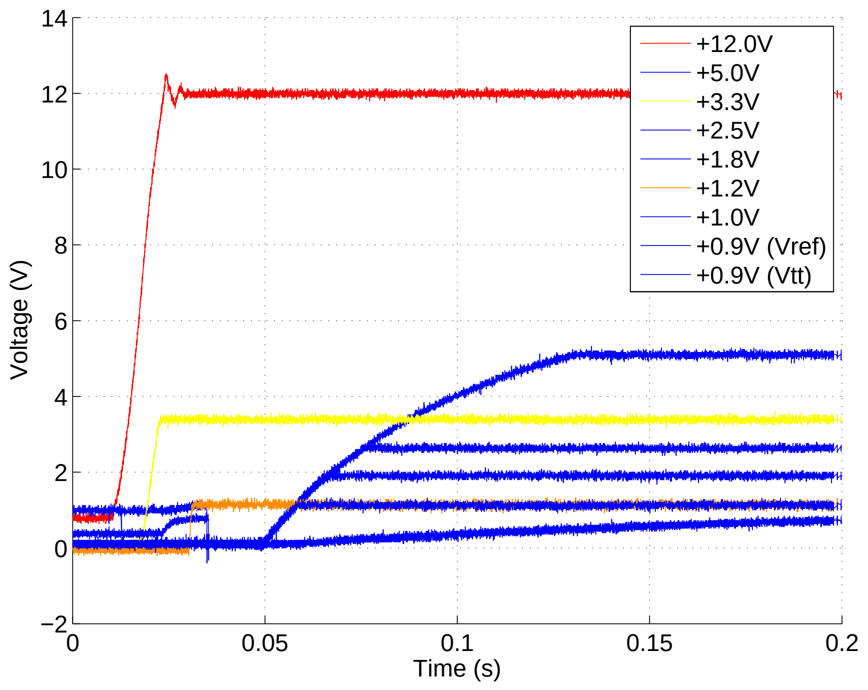

Figure 4.32: Measured Digital Power Supply Sequence with correct C_{SS} capacitor value of 10 nF.

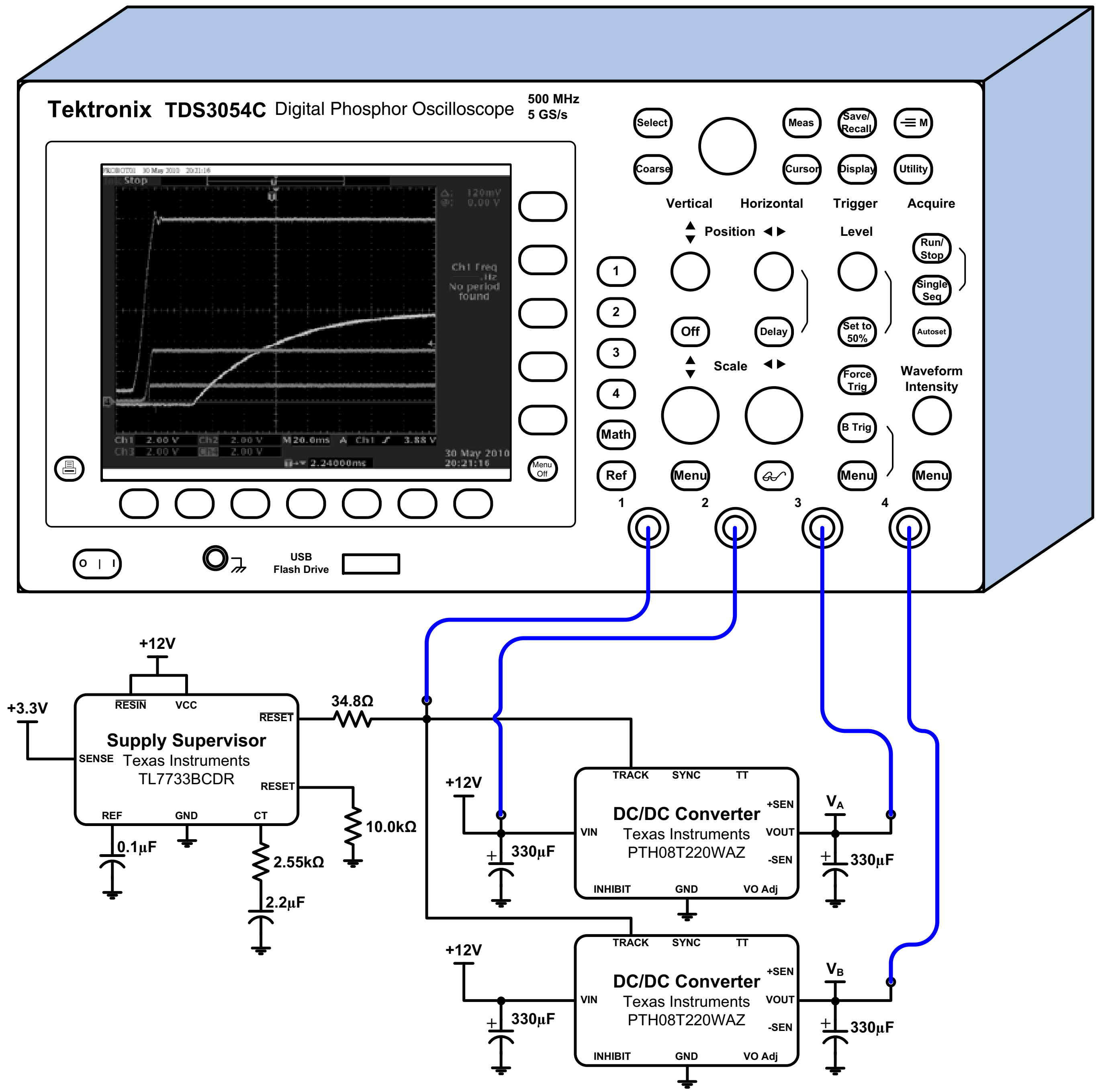

Figure 4.33: Power Supply Sequence Measurement Setup

4.3.1.7 Power Supply Consumption

The power supply consumption of the measurement board was measured using a digital multimeter. Measurements were taken at the inputs of each DC/DC converter and linear regulator via a series resistor of value 0.01 Ω. The voltage across the input resistor was measured, and the current was calculated using Equation 4.11.

$$I\ [\mathrm{A}] = \frac{V_{drop}}{R_{series}}\ [\frac{\mathrm{V}}{\Omega}] \tag{4.11}$$

The total power supply consumption was calculated by measuring the current at each DC/DC converter and linear regulator running on +12 V, and the results are shown in Table 4.13.

Name	**V$_{IN}$ (V)**	**V$_{OUT}$ (V)**	**I (A)**	**Power (W)**
Digital				
P5VD	12.0	5.0	0.0798	0.95760
P3V3D	12.0	3.3	0.2475	2.97000
P2V5D	12.0	2.5	0.3259	3.91080
P1V8D	12.0	1.8	0.2261	2.71320
P1V0D	12.0	1.0	0.1559	1.87080
P0V9D (VTT/VREF)	12.0	0.9	0.0793	0.95160
P1V3D_ASAP	12.0	1.3	0.0028	0.03360
P1V0D_ASAP	12.0	1.0	0.0036	0.04320
Analog				
P8VA_CLKDIV	12.0	8.0	0.0323	0.38610
P5V5A	12.0	5.5	0.8970	10.7245
P2V5A	12.0	2.5	0.0525	0.62780
N6VA	12.0	-6.0	0.0618	0.73900
			Total Power	25.9283

Table 4.13: Power Supply Consumption: First Stage

The measurement results of the power consumption for each linear regulator are shown in Table 4.14.

Name	**V$_{IN}$ (V)**	**V$_{OUT}$ (V)**	**I (A)**	**Power (W)**
Digital				
P1V2D	3.3	1.2	0.2270	0.74810
Analog				
P1V8A_SS	2.5	1.8	0.0287	0.07240
N5V2A_SS	-6	-5.2	0.0210	0.12450
N5V2A_AIF	-6	-5.2	0.0425	0.25140
N2V5A_AIF	-5.2	-2.5	0.0472	0.24670
P5V2A	5.5	5.2	0.4421	2.41110
P3V3A_CLK	5.5	3.3	0.4541	2.46800
P3V3A_CLKDIV	5.5	3.3	0.4023	2.19410
P3V3A_SS	5.5	3.3	0.0596	0.32630
P3V3A_AIF	5.5	3.3	0.1372	0.74990
P3V3D_AIF	5.5	3.3	0.0815	0.44570
P2V5A_AIF	5.5	2.5	0.0364	0.19910
P2V5A_TRIG	5.5	2.5	0.0550	0.30120

Table 4.14: Power Supply Consumption: Second Stage

Chapter 5

Future Work and Conclusion

The General Purpose Instrument, shown in Figures 1.1 and 5.1, is a successful development platform that can be used for a wide variety of AsAP DSP software prototyping. This platform can be used to target applications from software defined radios to cognitive radio. The signal bandwidth of the front end designs exceeded my initial design goals of a frequency range from DC to 110 MHz by 15 MHz.

Future work on both the signal source and signal analyzer can further improve the performance and usability for prototyping AsAP applications. During the verification of the signal source and signal analyzer a mistake was discovered in the PLL loop-filter of the high-speed clock generation circuit. Improvement in the loop-filter design has the potential to further increase the performance of both front end designs. Once the PLL loop-filter has been addressed, an investigation of the system jitter performance would provide more information on the system noise floor and may help identify circuits that can be modified to improve the bandwidth and dynamic range.

Due to the size and scope of this project, there was not enough time to fully turn-on the AsAP processors and interface them with the DSP sub-system. Adding this capability will allow future research students to prototype AsAP applications.

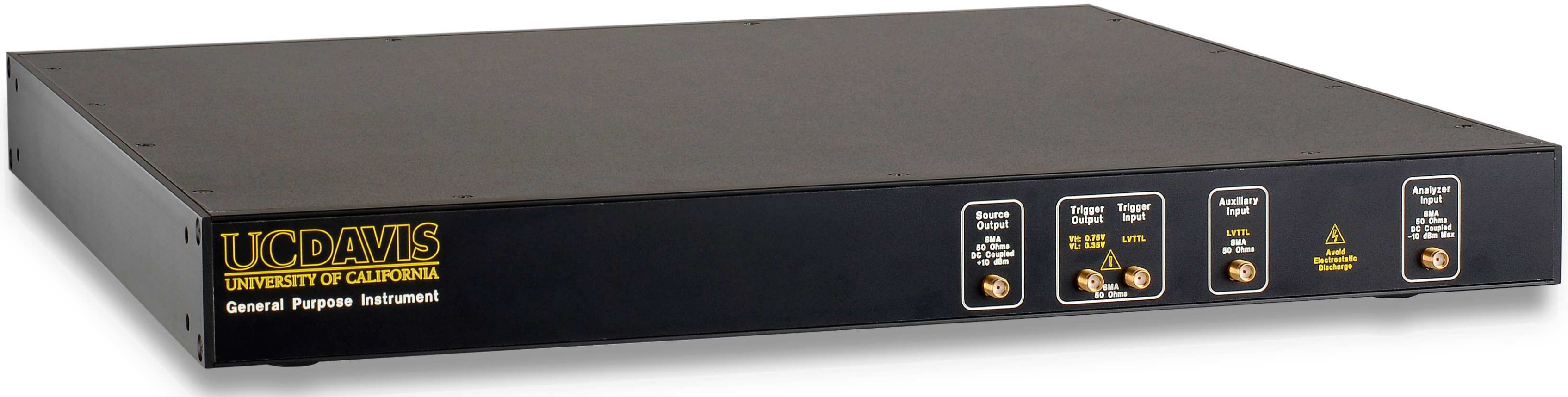

Figure 5.1: General Purpose Instrument ISO View

Appendix A

Waveform Generation and Play Back

The signal source of the General Purpose Instrument can play back waveform files up to 32 Mbits in length. The General Purpose Instrument has a variety of waveforms preloaded onto an internal 2 GB microSD card, including:

- Sinusoid Waveforms ranging in frequency from 1 kHz to 250 MHz and power levels of 0 dBFS, −6 dBFS, and −12 dBFS.
- Ramp Waveforms ranging in frequency from 1 MHz to 10 MHz.
- Square Waveforms ranging in frequency from 1 MHz to 25 MHz.
- Burst Waveforms
- Comb Waveforms
- Multitone Waveforms

Custom waveform files are also supported, but these files must be formatted properly and meet certain length requirements for waveform play back and loading to be successful. The following topics will be addressed:

- Waveform File Format
- Waveform Replication

- Waveform Parameters
- Waveform Generation

A.1 Waveform File Format

The signal source of the General Purpose Instrument can play back custom waveform files up to 32 Mbits long (e.g., 2^{25}). A waveform file is made up of 16 bit signed 2's complement hexadecimal samples. The bit length of the waveform file must be a multiple of 128 bits. If the created waveform is not a multiple of 128 bits, or is less than 256 bits, the waveform must be replicated until it is a multiple of 128 bits and at least 256 bits. In some cases, the waveform created may not fit in the available amount of memory when replicated.

While the signal source can support up to 32 Mbits of waveform data using QDR-II SRAM, the Control FPGA software limits the size of waveform files to at most 24 Mbits. This limitation is imposed as a result of an 8 MB RAM disk, which is created in the DDR SDRAM memory and used by the software to buffer waveform files as they are transferred from the microSD card and loaded into the waveform memory. A waveform file containing 24 Mbit is exactly 8 MB when stored in ASCII format. Future support of 32 Mbit long waveforms can be added by modifying the software in one of two ways:

1. Increase the size of the RAM disk such that it will allow waveform files of 32 Mbits in length to be played back.
2. Require the waveform files to be stored in a binary format, which will reduce the overall file size.

A.1.1 Filename Support

The General Purpose Instrument requires waveform filenames to be in a DOS 8.3 format. Only the following characters are allowed in the filename:

- A-Z
- a-z
- 0-9
- –,‾

A.1.2 File Header

The waveform file will contain a header with the following parameters:

- **patternname**: Pattern Name; myfile.usr
- **patterntype**: Pattern Type; sram or bram
- **patternlength**: Pattern Length; number of bits
- **readstartaddressa**: Pattern A Start Address
- **readstopaddressa**: Pattern A Stop Address
- **readstartaddressb**: Pattern B Start Address; currently not used
- **readstopaddressb**: Pattern B Stop Address; currently not used
- **density**: Pattern Mark/Space Density; range 0 to 1000
- **description**: Pattern Description; 256 character limit
- **triggerword**: Pattern Trigger Word; currently not used
- **bitshift**: Pattern Bit Shift; currently not used
- **bitshiftindex**: Pattern Bit Shift Index; currently not used
- **crc**: Pattern CRC Checksum; currently not used
- **version**: Pattern Utility Version; 1.0

A sample waveform file header is shown in Listing A.1.

Listing A.1: Example Waveform File Header

```
##########################################################
#description=This is where the description goes.
#patternname=mypattern.pat
#patterntype=sram
#patternlength=256
#readstartaddressa=0x0
#readstopaddressa=0x1
#readstartaddressb=0
#readstopaddressb=0
#triggerword=0
#patternstatistics=
#density=500.000
#bitshift=no
#bitshiftindex=0
#crc=x
#version=1.0
##########################################################
```

A.1.3 Sample Waveform File

The following is an example waveform file with a description:

Listing A.2: Example Waveform File

```
##################################################################
#description=This is where the description goes.
#patternname=mypattern.usr
#patterntype=sram
#patternlength=256
#readstartaddressa=0x0
#readstopaddressa=0x1
#readstartaddressb=0
#readstopaddressb=0
#triggerword=0
#patternstatistics=
#density=50.000
#bitshift=no
#bitshiftindex=0
#crc=x
#version=1.0
##################################################################
#begin
AAFC0418
51E459D4
FA1C49B5
BD8D2EE6
AAFC0418
51E459D4
FA1C49B5
BD8D2EE6
#end
```

A.1.3.1 File Data

The waveform data will be written to the file using hexadecimal format with 32 bits or two 16 bit samples per-line. The data payload will be preceded with a '#begin' tag and followed by an '#end' tag. An example of the waveform data is shown in Listing A.3.

Listing A.3: Example Waveform Data

```
#begin
AAFC0418
51E459D4
FA1C49B5
BD8D2EE6
AAFC0418
51E459D4
FA1C49B5
BD8D2EE6
#end
```

A.2 Waveform Replication

Technically, there is no minimum requirement for waveform length, but the signal source of the General Purpose Instrument requires waveforms to be at least 256 bits long. If a waveform is created for which L < 256 is true, then the waveform will need to be replicated until it meets the minimum length requirement. After the waveform has been replicated to a length of at least 256 bits, it can then be tested to determine if further replication is required. Waveforms must always end on a 128 bit boundary; in other words the waveform length must be a multiple of 128 bits. A script can be used to calculate the total number of bits in the waveform and determine if the total is a multiple of 128 bits. Equation A.1 shows how to determine if the waveform length is a multiple of 128 bits.

$$L_{mod} = mod\,(L, 128) \tag{A.1}$$

If L $\geq$ 256 is true and L$_{mod}$ is equal to zero, then waveform replication is not required. If L $\geq$ 256 is true and L$_{mod}$ is *not* equal to zero, then waveform replication is required. Algorithm A.2.1 shows how to determine the replicated length, where L is the original waveform length and the minimum waveform length is 256 bits.

Algorithm A.2.1: Waveform Replication

```
if ((mod(L,128) ≠ 0) & (mod(L,32) == 0)) then
    $L_{rep} = L \cdot mod(L,128)$;
else if (mod(L,128) ≠ 0) then
    $L_{rep} = L \cdot 128$;
else
    $L_{rep} = L$;
```

Once the replicated waveform length has been calculated, it must be compared against the capacity of the desired waveform memory to determine if the waveform will fit. For QDR-II SRAM, the waveform length must be less than 32 Mbits. The equation used is: $\frac{L_{rep}}{32 \text{ Mbits}} \leq 1$.

- If the ratio is less than 1, then the replicated waveform will fit within the QDR-II SRAM.
- If the ratio is greater than 1, then the waveform will not fit in the QDR-II SRAM.

For Block RAM, the waveform length must be less than 2 Mbits. The equation used is: $\frac{L_{rep}}{2 \text{ Mbits}} \leq 1$.

- If the ratio is less than 1, then the replicated waveform will fit within the Block RAM.
- If the ratio is greater than 1, then the waveform will not fit in the Block RAM.

A.3 Waveform Parameters

The file header defined in Section A.1.2 contains several parameters that need to be calculated from the waveform data, including:

- Waveform Bit Length
- Waveform Memory Stop Address
- Waveform Mark Density

A.3.1 Waveform Bit Length

Once it has been determined that the waveform requires replication, the waveform length will be re-calculated from the replicated waveform file. The waveform length must be checked to ensure that it will fit within the appropriate waveform memory. The QDR-II SRAM is constructed as 256-kwords x 128-bits (or 2^{25} bits). The Block RAM is constructed as 16-kwords x 128-bits (or 2^{21} bits). The total bit length of the replicated waveform data can be calculated using Equation A.2.

$$L_{rep} = N_{samples} \cdot 16 \tag{A.2}$$

A.3.2 Waveform Memory Stop Address

Once the replicated waveform length has been determined, the memory stop address must be calculated for the waveform file. The memory stop address is used by the Data Path FPGA to determine when to roll-over back to the start address, which is typically 0x0. The stop address for both the Block RAM and QDR-II SRAM is shown in Equation A.3.

$$Address_{STOP} = \left(\frac{L_{rep}}{128} - 1 \right) \tag{A.3}$$

The QDR-II SRAM stop address will be stored as an 18-bit hexadecimal value and the Block RAM stop address will be stored as an 14-bit hexadecimal value. The stop address will be written to the header of the waveform file.

A.3.3 Waveform Mark Density

The Mark Density of the waveform bits must be calculated and stored in the header of the waveform file. Equation A.4 can be used to calculate the Mark Density. The resulting Mark Density will be a value in the range of 1.00 to 1000.00.

$$MD = \left(\frac{Number\ of\ Ones}{Number\ of\ Bits} \right) \cdot 1000 \tag{A.4}$$

A.4 Waveform Generation

Several scripts were written in a variety of scripting languages to facilitate the generation of waveform files suitable for loading and play back from the signal source of the General Purpose Instrument. The scripting language used to generate waveform files is dependent on the type analysis required to verify the waveform is correct. Simple waveforms can be generated using either Perl or Bash scripts. More complex waveforms can be generated using Matlab, which excels at plotting data in both the time and frequency domain. Some of the scripts are used specifically for generating waveform files that contain a single cycle of a waveform that, when played back, repeatedly generate a continuous waveform. Others are targeted at analyzing, replicating, and reformatting the single cycle waveform files so that they meet the waveform file requirements outlined in Sections A.1 to A.3.

The following scripts were used to generate waveform files:

- wave_array.m - Sinusoid Waveform Generation
- square.pl - Square Waveform Generation
- ramp.pl - Ramp Waveform Generation

The output waveform files generated by the above scripts are then filtered by the sig2hex.pl script, which is described in detail in Appendix Chapter B. Some of the scripts above can iterate over a frequency range to generate multiple files. The generation of waveform files suitable for loading onto the General Purpose Instrument can be further scripted using a Bash script.

A.4.1 Sinusoid Waveform Generation

A sinusoid waveform can be generated in Matlab using either the sin() function with a 90° phase offset or the cos() function. A for loop was employed to iterate over the frequency ranges described in Table A.1.

$\mathbf{F}_{start}$	$\mathbf{F}_{stop}$	$\mathbf{F}_{inc}$	**nSamps**
1 kHz	9 kHz	1 kHz	1048576
10 kHz	1 MHz	10 kHz	1048576
1 MHz	250 MHz	1 MHz	16384

Table A.1: Sinusoid Waveform Frequency Range Matlab script parameters

Equation A.5 describes the function used to generate the sinusoid waveform.

$$y[n] = A \cdot cos\left(2 \cdot \pi \cdot F_c \cdot x[n]\right) \tag{A.5}$$

The function *x[n]* is described by Equation A.6, where *nSamps* represents the number of waveform samples and varies for each frequency range.

$$x[n] = \frac{(0:1:(nSamps-1))}{F_s} \tag{A.6}$$

A scale factor of A, described in Equation A.7, was used to scale each sinusoid waveform generated.

$$A = \frac{10^{\left(\frac{A_{dBm}}{20}\right)}}{\sqrt{\frac{\frac{1000}{2}}{50}}} \tag{A.7}$$

The parameter A$_{dBm}$, shown in Equation A.7, was set to 9.95 dBm, or slightly less than 2 V$_{\mathrm{pkpk}}$. Once the sinusoid waveform function is generated, the Matlab script will then extract a single cycle and write the waveform data to a file in a format suitable for processing by the waveform conversion script, sig2hex.pl. A sample sinusoid waveform generation Matlab script for the frequency range of 10 MHz to 250 MHz in 1 MHz steps is shown in Listing A.4.

Listing A.4: Sinusoid Waveform Generation Matlab Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% wave_array.m script
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% VCL Confidential, Copyright   2009 UC Davis ECE Department
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% created on:    11/02/2009
% created by:    jwwebb
% last edit on: $DateTime: $
% last edit by: $Author: $
% revision:      $Revision: $
% comments:      Generated
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Sine Waveform Generation
%
% This matlab script generates a sine waveform file for the
% Data Path FPGA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;clc;close all;
PrintOnEps = 0;
PlotMe = 0;
WriteMe = 1;

for f = 1:1:250;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Generate the Sine Waveform:
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fc  = f*1e6; % in Hz
    fs  = 500e6; % in Hz
    amplitude_dBm = 9.95; % Equivalent to slightly less than 2 Vpkpk
    amplitude_V = 10^(amplitude_dBm/20)/sqrt(1000/2/50);  %in volts
    nSamps = 16384;
    nData = 0:(nSamps-1);
    xData = nData/fs;
    yData = amplitude_V.*cos(2*pi*fc*xData);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Write the Sine Waveform to a Text File in 2's Complement Decimal.
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if WriteMe
        getOnes = find(yData == amplitude_V);
        start = 1;
        stop = getOnes(2)-1;
        fData = yData(start:stop);
        stopAddr = dec2hex(((length(fData)*16*128)/128)-4);
        filename = sprintf('./file_in/pat0dBFS/sine%dm.pat',fc/1e6);
        fid = fopen(filename,'w');
        fprintf(fid,'########################################################################\n'
            );
        fprintf(fid,'#description=Sine␣Waveform:␣Fc␣=␣%dMHz,␣Fs␣=␣%dMHz\n', fc/1e6, fs/1e6
            );
        fprintf(fid,'#patternname=%s\n', filename);
        fprintf(fid,'#patterntype=sram\n');
        fprintf(fid,'#patternlength=%d\n',(length(fData)*128));
        fprintf(fid,'#readstartaddressa=0x0\n');
        fprintf(fid,'#readstopaddressa=0x%s\n',stopAddr);
        fprintf(fid,'#readstartaddressb=0\n');
        fprintf(fid,'#readstopaddressb=0\n');
        fprintf(fid,'#triggerword=0\n');
        fprintf(fid,'#patternstatistics=\n');
        fprintf(fid,'#density=0.750\n');
        fprintf(fid,'#bitshift=no\n');
        fprintf(fid,'#bitshiftindex=0␣\n');
        fprintf(fid,'#crc=???\n');
        fprintf(fid,'#version=1.0\n');
        fprintf(fid,'########################################################################\n'
            );
        fprintf(fid,'#begin\n');
        for j = 1:1:length(fData);
            y1 = fData(j);
            fprintf(fid,'%2.16f',y1);
            fprintf(fid,'\n');
        end;
        fprintf(fid,'#end\n');
        fclose(fid);
    end
end
```

Listing A.5 shows a sample waveform file for a 100 MHz sinusoid waveform.

Listing A.5: Example Sinusoid Waveform File

```
####################################################################
#description=Sine Waveform: Fc = 100MHz, Fs = 500MHz
#patternname=sine100m.pat
#patterntype=sram
#patternlength=640
#readstartaddressa=0x0
#readstopaddressa=0x13C
#readstartaddressb=0
#readstopaddressb=0
#triggerword=0
#patternstatistics=
#density=0.750
#bitshift=no
#bitshiftindex=0
#crc=???
#version=1.0
####################################################################
#begin
1.0000000000000000
0.3090169943749475
-0.8090169943749473
-0.8090169943749478
0.3090169943749472
#end
```

A.4.2 Ramp Waveform Generation

A ramp, or triangle, waveform can be generated by incrementing a sample value from $-N$ to N and then decrementing a sample value from N to $-N$. The period of the ramp waveform is determined by the number of samples present on the incline and decline of the ramp waveform. A Perl script, ramp.pl, was developed to generate a ramp waveform with a frequency in the range of 1 kHz to 10 MHz. Table A.2 outlines the parameters used by the ramp.pl script to determine the number of samples required on the incline and decline of the ramp waveform to achieve the desired ramp waveform frequency.

Parameter	**Value**	**Description**
DACRES	16 bits	High-Speed DAC Resolution
MAXVAL	2^{15}	Maximum Signed 2's Complement Value
MINVAL	-2^{15}	Minimum Signed 2's Complement Value
F_s	500 MHz	High-Speed DAC Sample Frequency
F_{ramp}	1 kHz to 10 MHz	Desired Ramp Waveform Frequency

Table A.2: ramp.pl Perl script parameters

The total number of samples required to achieve the desired ramp waveform frequency can be calculated using Equation A.8.

$$NumSamples = \left(\frac{\frac{1}{F_{ramp}}}{\frac{1}{F_s}}\right) = \left(\frac{F_s}{F_{ramp}}\right) \tag{A.8}$$

The number of samples required for the incline or decline is simply *NumSamples* divided by two. The sample increment is common to both the incline and decline of the ramp waveform, and can be calculated using Equation A.9.

$$RampInc = \left(\frac{MAXVAL - MINVAL}{\frac{NumSamples}{2}}\right) = \left(\frac{2^{16} \cdot 2}{NumSamples}\right) = \left(\frac{2^{17}}{NumSamples}\right) \tag{A.9}$$

Algorithm A.4.1 describes how the ramp waveform is created using the parameters listed in Table A.2 and calculated in Equations A.8 and A.9 parameters. Upon completion of the algorithm, the array variable *ramp* is written to a waveform file in a format suitable for processing by the waveform conversion script, sig2hex.pl.

Algorithm A.4.1: Ramp Waveform Generation Algorithm

```
Data: RampInc=equally spaced increment value from min to max DAC value.
Data: idx = 1
Data: mdx = 0
begin
    for $(p = -2^{15}, p \leq 2^{15}, p = p + RampInc)$ do
        ramp(idx) = $\frac{p}{2^{15}}$;
        idx = idx + 1;
    for $(m = 2^{15}, m \geq (-2^{15} + RampInc), m = m - RampInc)$ do
        if $(mdx > 0)$ then
            ramp(idx) = $\frac{m}{2^{15}}$;
            idx = idx + 1;
        mdx = mdx + 1;
```

Listing A.6 shows a sample waveform file for a 10 MHz ramp waveform.

Listing A.6: Example Ramp Waveform File

```
##################################################################
#description=Ramp Waveform: Fc = 10.0 MHz, Fs = 500.0 MHz
#patternname=ramp_10mhz.pat
#patterntype=sram
#patternlength=25600
#readstartaddressa=0x0
#readstopaddressa=0xc7
#readstartaddressb=0
#readstopaddressb=0
#triggerword=0
#patternstatistics=
#density=0.750
#bitshift=no
#bitshiftindex=0
#crc=???
#version=1.0
##################################################################
#begin
-1.000000000000000
-0.920000000000000
-0.840000000000000
-0.760000000000000
-0.680000000000000
-0.600000000000000
-0.520000000000000
```

```
-0.440000000000000
-0.360000000000000
-0.280000000000000
-0.200000000000000
-0.120000000000000
-0.040000000000000
0.040000000000000
0.120000000000000
0.200000000000000
0.280000000000000
0.360000000000000
0.440000000000000
0.520000000000000
0.600000000000000
0.680000000000000
0.760000000000000
0.840000000000000
0.920000000000000
1.000000000000000
0.920000000000000
0.840000000000000
0.760000000000000
0.680000000000000
0.600000000000000
0.520000000000000
0.440000000000000
0.360000000000000
0.280000000000000
0.200000000000000
0.120000000000000
0.040000000000000
-0.040000000000000
-0.120000000000000
-0.200000000000000
-0.280000000000000
-0.360000000000000
-0.440000000000000
-0.520000000000000
-0.600000000000000
-0.680000000000000
-0.760000000000000
-0.840000000000000
-0.920000000000000
#end
```

More information, including the Perl code, can be found in Appendix Chapter D.

A.4.3 Square Waveform Generation

A square waveform can be generated by assigning the sample value the maximum value for half of the period, and then assigning the sample value the minimum value for the remaining half of the period. The period of the square waveform is determined by the number of samples present on the positive and negative pulses of the square waveform. A Perl script, square.pl, was developed to generate a square waveform with a frequency in the range of 1 kHz to 25 MHz. Table A.3 outlines the parameters used by the square.pl script to determine the number of samples required on the positive and negative pulses of the square waveform to achieve the desired square waveform frequency.

Parameter	**Value**	**Description**
DACRES	16 bits	High-Speed DAC Resolution
MAXVAL	2^{15}	Maximum Signed 2's Complement Value
MINVAL	-2^{15}	Minimum Signed 2's Complement Value
F_s	500 MHz	High-Speed DAC Sample Frequency
F_{square}	1 kHz to 25 MHz	Desired Square Waveform Frequency

Table A.3: square.pl Perl script parameters

The total number of samples required to achieve the desired square waveform frequency can be calculated using Equation A.10.

$$NumSamples = \left(\frac{\frac{1}{F_{square}}}{\frac{1}{F_s}}\right) = \left(\frac{F_s}{F_{square}}\right) \tag{A.10}$$

The number of samples required for the positive or negative pulses is simply *NumSamples* divided by two. Algorithm A.4.2 describes how the square waveform is created using the parameters listed in Table A.3 and calculated in Equation A.10. Upon completion of the algorithm, the array variable *square* is written to a waveform file in a format suitable for processing by the waveform conversion script, sig2hex.pl.

Algorithm A.4.2: Square Waveform Generation Algorithm

```
Data: idx = 1
Data: mdx = 0
begin
    square(idx) = 0;
    idx = idx + 1;
    for $\left(p = 0, p < \left(\frac{NumSamples}{2}\right), p = p + 1\right)$ do
        square(idx) = $\frac{2^{15}}{2^{15}}$;
        idx = idx + 1;
    square(idx) = 0;
    idx = idx + 1;
    for $\left(m = 0, m < \left(\frac{NumSamples}{2}\right), m = m + 1\right)$ do
        square(idx) = $-\frac{2^{15}}{2^{15}}$;
        idx = idx + 1;
```

Listing A.7 shows a sample waveform file for a 25 MHz square waveform.

Listing A.7: Example Square Waveform File

```
################################################################
#description=Square Waveform: Fc = 25.0 MHz, Fs = 500.0 MHz
#patternname=squ_25mhz.pat
#patterntype=sram
#patternlength=33792
#readstartaddressa=0x0
#readstopaddressa=0x107
#readstartaddressb=0
#readstopaddressb=0
#triggerword=0
#patternstatistics=
#density=0.750
#bitshift=no
#bitshiftindex=0
#crc=???
#version=1.0
################################################################
#begin
0.000000000000000
1.000000000000000
1.000000000000000
1.000000000000000
1.000000000000000
```

```
1.000000000000000
1.000000000000000
1.000000000000000
1.000000000000000
1.000000000000000
1.000000000000000
0.000000000000000
−1.000000000000000
−1.000000000000000
−1.000000000000000
−1.000000000000000
−1.000000000000000
−1.000000000000000
−1.000000000000000
−1.000000000000000
−1.000000000000000
−1.000000000000000
#end
```

More information, including the Perl code, can be found in Appendix Chapter C.

A.4.4 Waveform Generation Automation

It is imperative to automate the generation of waveform files as the number of waveform files that need to be created increases beyond a few files. As an example, the spurious-free dynamic range is typically measured at three power levels over the frequency range from 1 kHz to 250 MHz.

- 0 dBFS
- −6 dBFS
- −12 dBFS

The initial waveform files are generated using wave_array.m, described in Section A.4.1, and are then converted to the internal waveform file format for each power level using the sig2hex.pl script. A Bash script, shown in Listing A.8, was developed to automate the conversion and scaling of the waveform files for use with the SFDR measurement.

Listing A.8: Waveform Generation Automation Bash Script

```
#!/bin/sh

#-----------------------------------------------------------------------------
# SFDR Waveform File Generation Module
# filename:  gen_sfdr.sh
#
#   by Jeremy Webb
#
#   Rev 1.1, November 20, 2009
#
#   This utility is intended to generate the Waveform Files for
#   measuring SFDR. This script will generate sine waveform files
#   for the Measurement board with the following scale factors:
#
#           * +0dBFS  = 20*log10(2^15/2^15)
#           * -6dBFS  = 20*log10(2^14/2^15)
#           * -12dBFS = 20*log10(2^13/2^15)
#
#   An example usage is:
#
#           ./gen_sfdr.sh
#
#   Revision History:
#       1.0 11/02/2009  Initial release
#       1.1 11/20/2009  Combined all waveform gen into one script.
#
#   Please report bugs, errors, etc.
```

```
#------------------------------------------------------------------------

echo "Generating Hex Pattern Files for 0dBFS scale ..."
echo ""
echo "Frequency Range: 1MHz to 250MHz"
echo ""
X=1
while [ $X -le 250 ]
do
  ./sig2hex.pl -i ./file_in/pat0dBFS/sine${X}m.pat -o ./fo/pwr0dBFS/sine${X}m.pat -m 0 -r
      16 -s 15
  X=$((X+1))
done
echo "Frequency Range: 10kHz to 990kHz"
echo ""
X=10
while [ $X -le 990 ]
do
  ./sig2hex.pl -i ./file_in/pat0dBFS/sine${X}k.pat -o ./fo/pwr0dBFS/sine${X}k.pat -m 0 -r
      16 -s 15
  X=$((X+10))
done
echo "Frequency Range: 1kHz to 9kHz"
echo ""
X=1
while [ $X -le 9 ]
do
  ./sig2hex.pl -i ./file_in/pat0dBFS/sine${X}k.pat -o ./fo/pwr0dBFS/sine${X}k.pat -m 0 -r
      16 -s 15
  X=$((X+1))
done

echo "Generating Hex Pattern Files for -6dBFS scale ..."
echo ""
echo "Frequency Range: 1MHz to 250MHz"
echo ""
X=1
while [ $X -le 250 ]
do
  ./sig2hex.pl -i ./file_in/pat0dBFS/sine${X}m.pat -o ./fo/pwrn6dBFS/sine${X}m.pat -m 0 -r
       16 -s 14
  X=$((X+1))
done
echo "Frequency Range: 10kHz to 990kHz"
echo ""
X=10
while [ $X -le 990 ]
do
```

```
  ./sig2hex.pl -i ./file_in/pat0dBFS/sine${X}k.pat -o ./fo/pwrn6dBFS/sine${X}k.pat -m 0 -r
       16 -s 14
  X=$((X+10))
done
echo "Frequency Range: 1kHz to 9kHz"
echo ""
X=1
while [ $X -le 9 ]
do
  ./sig2hex.pl -i ./file_in/pat0dBFS/sine${X}k.pat -o ./fo/pwrn6dBFS/sine${X}k.pat -m 0 -r
       16 -s 14
  X=$((X+1))
done

echo "Generating Hex Pattern Files for -12dBFS scale..."
echo ""
echo "Frequency Range: 1MHz to 250MHz"
echo ""
X=1
while [ $X -le 250 ]
do
  ./sig2hex.pl -i ./file_in/pat0dBFS/sine${X}m.pat -o ./fo/pwrn12dBFS/sine${X}m.pat -m 0 -
      r 16 -s 13
  X=$((X+1))
done
echo "Frequency Range: 10kHz to 990kHz"
echo ""
X=10
while [ $X -le 990 ]
do
  ./sig2hex.pl -i ./file_in/pat0dBFS/sine${X}k.pat -o ./fo/pwrn12dBFS/sine${X}k.pat -m 0 -
      r 16 -s 13
  X=$((X+10))
done
echo "Frequency Range: 1kHz to 9kHz"
echo ""
X=1
while [ $X -le 9 ]
do
  ./sig2hex.pl -i ./file_in/pat0dBFS/sine${X}k.pat -o ./fo/pwrn12dBFS/sine${X}k.pat -m 0 -
      r 16 -s 13
  X=$((X+1))
done
```

A.5 Waveform Play Back

Waveforms can be loaded into the signal source for generation by the General Purpose Instrument using remote commands via the USB interface. A list of patterns available for play back on the General Purpose Instrument can be viewed by executing the command shown in Listing A.9.

Listing A.9: Waveform File Name List Command

```
ls pattern

where pattern is the name of the directory containing the waveform files
available for play back on the General Purpose Instrument via the signal source
output.
```

Waveform files can be loaded from the microSD card into either the Block RAM or the QDR-II SRAM waveform memory by executing the command shown in Listing A.10.

Listing A.10: Waveform Load Command

```
src_load [1|0] [pattern\type\name.pat]

where [1|0] selects either Block RAM (0) or QDR-II SRAM (1) as the playback
memory, and [pattern\type\name.pat] is the path to the pattern file.
```

Appendix B

Waveform Conversion Perl Script

This chapter describes the sig2hex waveform conversion Perl script, which is used to convert Matlab waveform files into a format supported by the Measurement board for waveform playback.

B.1 NAME

sig2hex.pl - Waveform Signed 2's Compliment Decimal-to-Hexadecimal Converter

B.2 SYNOPSIS

```
sig2hex.pl [-h] [-v] [-i <FILE>] [-o <FILE>] [-r <RES>] [-s <SCALE>] [-m <TYPE>]

Help Options:
 -h          Print Help.
 -v          Verbose: Print Debug Information.
 -i <FILE>   Waveform Input filename.
 -o <FILE>   Waveform Output filename.
 -r <RES>    Waveform Resolution (Default: 16).
 -s <SCALE>  Waveform Maximum Value: 2^N (Default: 15).
 -m <TYPE>    Waveform Type; 0: QDR-II SRAM, 1: BRAM.

 Example:
     ./sig2hex.pl -v -i sample.pat -o mypat.pat -r 16
     ./sig2hex.pl -i ./file_in/sine1m.pat -o ./fo/sine1m.pat -m 0 -r 16 -s 15
```

B.3 OPTIONS

-h

Show the brief help information.

-v

Show debug information.

-i FILE

Waveform input filename containing a single cycle of the desired waveform in signed 2's complement decimal format.

-o FILE

Waveform output filename containing a replicated waveform in hexadecimal format. The output file contains a header, which is used by the Control FPGA on the Measurement board to aid in the loading of the pattern into the desired pattern memory. The hexadecimal data is formatted as eight hexadecimal digits per line, and the data is encompassed by a #begin/#end tag.

-r RES

Resolution of the digital-to-analog converter generating the waveforms.

-s SCALE

Scale of the generated waveform. This option can be used to scale the waveform by a power of 2. For example, 0dBFS refers to a SCALE value of 15, -6dBFS refers to a SCALE value of 14, -12dBFS refers to a SCALE value of 13, etc.

-m TYPE

Select the waveform memory type. A 1 indicates Block RAM, and a 0 indicates QDR-II SRAM.

B.4 DESCRIPTION

sig2hex.pl is used to convert waveform files created using Matlab into a format suitable for playback on the Measurement Board. **sig2hex.pl** will read in a waveform file with a single signed 2's complement decimal value per line. It will quantize the data based on the desired resolution as follows:

$$y[n] = round\left(\frac{x[n] \cdot 2^{20}}{2^{20}}\right) \tag{B.1}$$

The quantized waveform data will then be converted to a hexadecimal signed 2's complement value with the desired resolution. The data will then be written out to a file for use with the Measurement Board Data Path FPGA.

B.5 SUBROUTINES

Get Waveform File

The **getFile** sub-routine will open the input waveform file in signed 2's complement format and store the data into a hash as an array. In addition, the **getFile** will also store the total number of lines in the file into a hash.

Parse Waveform File

The **parseFile** sub-routine will parse the waveform file to find the #begin and #end tags, which identify the location of the waveform data in the file array. After locating the waveform data, **parseFile** will then extract the waveform data from the file array into its own array and store both the array and the number of waveform samples in a hash.

Convert Signed 2's Complement Decimal to 16-bit Hexadecimal

The **convDec2Hex** sub-routine will quantize, scale, and convert the signed 2's complement decimal data into hexadecimal data. It will then write the data to a waveform file suitable for loading onto the Measurement board.

ALGORITHM DESCRIPTION

1. Determine maximum value of a hexadecimal sample using the RES value provided by the user.

   a. `$maxValue = 2`$^{\$RES}$

   b. The default RES value for the Measurement board DAC is 16.

2. Determine the scale coefficient for the quantized sample data.

   a. `$maxScale = 2`$^{\$SCALE}$

   b. The default SCALE value for the Measurement board DAC is 15.

3. Grab a signed 2's complement decimal value from the data array.

4. Quantize signed 2's complement decimal value to RES bits.

   a. `$datQuant = ceil($datIn * $maxValue) / $maxValue`

5. Scale the quantized sample data by 2^{SCALE}.

   a. `$datScale = $datQuant * ($maxScale - 1)`

6. Convert the quantized and scaled data to hexadecimal.

   a. `$hexValue = dec2hex($datScale)`

7. Test hexadecimal value to ensure that it contains 4 hexadecimal digits.

8. Store the hexadecimal value in an array in the order provided by the input waveform file.

9. Determine if the hexadecimal waveform data meets the length requirements of the Measurement board and replicate the waveform if necessary.

   a. Waveform must be a minimum of 256 binary bits long.

   b. Waveform length must be a multiple of 128 bits.

   c. Waveform length must meet maximum requirements depending on the waveform memory type. Block RAM can support up to 2^{21} bits. QDR-II SRAM can support up to 2^{25} bits.

10. Calculate waveform stop address.

    a. `$stopAddr = ( ($LENBITS / 128) - 1 )`

11. Determine the mark density, or ratio of 1's and 0's, of the waveform.

12. Format waveform file header, and write to an output waveform file.

13. Write the 16-bit hexadecimal value to an output waveform file.

14. Write the replicated waveform to a file in decimal format for verification of waveform in Matlab.

Test Hexadecimal Values

The **testhex** sub-routine will receive a single hexadecimal value. If the hexadecimal value is less than 4 digits, then **testhex** will pre-pend the appropriate number of zeros in order to provide a complete 16-bit hexadecimal value. An error flag is also provided to determine if the hex value is more than 4 digits.

Calculate the Replicated Waveform Length

The **repCalc** sub-routine will receive a hexadecimal waveform array, and determine if its overall bit length meets the length requirements of the Measurement board. If the requirements are not met, then the **repCalc** sub-routine will replicate the pattern as necessary.

ALGORITHM DESCRIPTION

1. Determine if the hexadecimal waveform data meets the minimum length requirement of 256 bits.

   a. Replicate the hexadecimal waveform data such that it meets the minimum length requirement.

2. Is the waveform length a multiple of 128 bits?

3. Is the waveform length a multiple of 32 bits?

4. Is the waveform length a multiple of 2 bits?

5. Determine if the hexadecimal waveform data needs to be further replicated.

   a. If the waveform length is a multiple of 32 bits but not a multiple of 128 bits, then the waveform must be replicated by `$REPNUM = ($LENBITS * 16) % 128`.

   b. If the waveform length is not a multiple of 32 bits and not a multiple of 128 bits, then the waveform must be replicated by `$REPNUM = 128`.

   c. If the waveform length is a multiple of 128 bits, then the waveform does not require replication.

6. Replicate the hexadecimal waveform data if necessary.

7. Determine if the hexadecimal waveform data meets the maximum length requirements depending on the waveform type selected.

   a. Block RAM can support up to 2^{21} bits.

   b. QDR-II SRAM can support up to 2^{25} bits.

8. Calculate waveform stop address.

   a. `$stopAddr = ( ($LENBITS / 128) - 1 )`

9. Determine the mark density, or ratio of ones to total bits, of the waveform.

Calculate Mark Density of Waveform File

The **calcMD** sub-routine will determine the total number of logic ones in the waveform file, and calculate the mark density. Mark density is the ratio of logic ones to the total number of bits.

Decimal to Hexadecimal Conversion

The **dec2hex** sub-routine will convert a decimal value into its hexadecimal equivalent.

Hexadecimal to Binary Conversion

The **hex2bin** sub-routine will convert a hexadecimal value into its binary equivalent.

B.6 CODE

Listing B.1: Waveform Conversion Perl Script

```
#!/usr/bin/env perl
# vim:ts=4:sw=4:expandtab:cindent

#*****************************************************************
#
# sig2hex.pl module
#
#*****************************************************************
#
# VCL Confidential Copyright   2009 UC Davis, ECE Department
#
#*****************************************************************
#
# created on:   05/18/2009
# created by:   jwwebb
# last edit on: $DateTime: $
# last edit by: $Author: $
# revision: $Revision: $
# comments: Generated
#
#*****************************************************************
# Revision List:
#
#       1.0 05/18/2009  Initial release
#       1.1 11/20/2009  Added capability to scale
#                       the waveform from Matlab
#                        before converting to Hex
#                        and replicating.
#               1.2      01/01/2011      Add Perl POD documentation.
#
#   Please report bugs, errors, etc.
#*************************************************************
# Waveform Signed 2's Compliment Decimal-to-Hex Converter
#
#  This utility is intended to read in a waveform with a single
#  signed 2's complement decimal value per line. The utility will
#  quantize the data based on the desired resolution as follows:
#
#   y_n_scaled = round(x_n*2^20)/2^20;
#
#  The quantized waveform data will then be converted to a
#  hexadecimal signed 2's complement value with the desired
#  resolution. The data will then be written out to a file for
#  use with the Measurement Board Data Path FPGA.
#
#  Usage Information:
#
#        Usage: ./sig2hex.pl [-h] [-v] [-f <FILE>] [-r <RES>]
#
#           -h        Print Help.
#           -v        Verbose: Print Debug Information.
#           -f <FILE>    Input Waveform File (Decimal)
#           -r <RES>     Waveform Resolution (Default: 16)
#
#           Example:
#               ./sig2hex.pl -v -f sample.pat -r 16
#
#*************************************************************


#*************************************************************
# CPAN Modules
#*************************************************************
use strict;
use Getopt::Std;
#use Math::Round qw (round);
use POSIX;
use Data::Dumper;

#*************************************************************
# Constants and Variables:
#*************************************************************
my (%opts)=();
my ($filein);
my ($fileout);
my ($bram_nsram);
my ($res);
my ($scale);
my ($debug);
my (%patH, $pat_rH);

#*************************************************************
# Retrieve command line argument
#*************************************************************
getopts('hvi:o:r:m:s:',\%opts);

my $optslen = scalar( keys %opts );
print("Number_of_Options_on_Command-Line:_$optslen\n") if $opts{v};

#check for valid combination command-line arguments
if ( $opts{h} || !$opts{i} || ($optslen eq "0") ) {
     print_usage();
     exit;
}

# parse command-line arguments
$filein     = $opts{i};
$fileout    = $opts{o};
$res        = $opts{r};
$scale      = $opts{s};
$bram_nsram = $opts{m};
$debug      = $opts{v};

#*************************************************************
# Stuff input options into a Hash:
#*************************************************************
$patH{ 'wave_in' }  = $filein;
$patH{ 'wave_out' } = $fileout;
$patH{ 'resolution' }   = $res;
$patH{ 'scale' }    = $scale;
$patH{ 'bram_nsram' }   = $bram_nsram;
$patH{ 'debug' }    = $debug;

#*************************************************************
# Convert Waveform data:
#*************************************************************
if ($filein) {
     if (!(defined $res)) {
     $patH{ 'resolution' } = "16";
     print("WARNING:_No_Resolution_Provided,_Assuming_16-bit_Resolution!\n");
     }
     if (!(defined $scale)) {
```

```
    $patH{ 'scale' } = "15";
    print("WARNING: No Scale Provided, Assuming 2^15 max value!\n");
    }
    ##########################################################################
    # Get Waveform File:
    ##########################################################################
    $pat_rH = getFile(\%patH);
    ##########################################################################
    # Parse Waveform File:
    ##########################################################################
    $pat_rH = parseFile($pat_rH);
    ##########################################################################
    # Write data to File:
    ##########################################################################
    $pat_rH = convDec2Hex($pat_rH);
}


exit;
=pod

=head1 NAME

B<sig2hex.pl> - Waveform Signed 2's Compliment Decimal-to-Hexadecimal Converter

=head1 SYNOPSIS

  sig2hex.pl [-h] [-v] [-i <FILE>] [-o <FILE>] [-r <RES>] [-s <SCALE>] [-m <TYPE>]

  Help Options:
   -h           Print Help.
   -v           Verbose: Print Debug Information.
   -i <FILE>    Waveform Input filename.
   -o <FILE>    Waveform Output filename.
   -r <RES>     Waveform Resolution (Default: 16).
   -s <SCALE>   Waveform Maximum Value: 2^N (Default: 15).
   -m <TYPE>    Waveform Type; 0: QDR-II SRAM, 1: BRAM.

   Example:
       ./sig2hex.pl -v -i sample.pat -o mypat.pat -r 16
       ./sig2hex.pl -i ./file_in/pat0dBFS/sine1m.pat -o ./fo/sram/pwr0dBFS/sine1m.pat -m 0
     -r 16 -s 15

=head1 OPTIONS

=over 8

=item B<-h>
```

```
Show␣the␣brief␣help␣information.

=item␣B<-v>

Show␣debug␣information.

=item␣B<-i␣FILE>

Waveform␣input␣filename␣containing␣a␣single␣cycle␣of␣the␣desired␣waveform␣in␣signed␣2's
    complement decimal format.

=item B<-o FILE>

Waveform output filename containing a replicated waveform in hexadecimal format. The
    output file contains a header, which is used by the Control FPGA on the Measurement
    board to aid in the loading of the pattern into the desired pattern memory. The
    hexadecimal data is formatted as eight hexadecimal digits per line, and the data is
    encompassed by a #begin/#end tag.

=item B<-r RES>

Resolution of the digital-to-analog converter (DAC) generating the waveforms.

=item B<-s SCALE>

Scale of the generated waveform. This option can be used to scale the waveform by a power
    of 2. For example, 0dBFS refers to a SCALE value of 15, -6dBFS refers to a SCALE value
     of 14, -12dBFS refers to a SCALE value of 13, etc.

=item B<-m TYPE>

Select the waveform memory type. A 1 indicates Block RAM, and a 0 indicates QDR-II SRAM.

=back

=head1 DESCRIPTION

B<sig2hex.pl> is used to convert waveform files created using Matlab into a
format suitable for playback on the Measurement Board. B<sig2hex.pl> will
read in a waveform file with a single signed 2's␣complement␣decimal␣value␣per␣line.
It␣will␣quantize␣the␣data␣based␣on␣the␣desired␣resolution␣as␣follows:

=over␣4

=item␣*␣C<y[n]␣=␣round(x[n]␣*␣2**20)␣/␣2**20>

=back
```

```
The quantized waveform data will then be converted to a hexadecimal signed
2's complement value with the desired resolution. The data will then be written
out to a file for use with the Measurement Board Data Path FPGA.

=head1 SUBROUTINES

=cut

#****************************************************************
# Sub-routines
#****************************************************************

sub dienice {
    my($errmsg) = @_;
    print"$errmsg\n";
    exit;
}

sub print_usage {
    my ($usage);
    $usage = "\nUsage: $0 [-h] [-v] [-i <FILE>] [-o <FILE>] [-r <RES>] [-s <SCALE>] [-m <
        TYPE>]\n";
    $usage .= "\n";
    $usage .= "\t-h\t\tPrint Help.\n";
    $usage .= "\t-v\t\tVerbose: Print Debug Information.\n";
    $usage .= "\t-i <FILE>\tWaveform Input filename.\n";
    $usage .= "\t-o <FILE>\tWaveform Output filename.\n";
    $usage .= "\t-r <RES>\tWaveform Resolution (Default: 16).\n";
    $usage .= "\t-s <SCALE>\tWaveform Maximum Value: 2^N (Default: 15).\n";
    $usage .= "\t-m <TYPE>\tWaveform Type; 0: SRAM, 1: BRAM.\n";
    $usage .= "\n";
    $usage .= "\tExample:\n";
    $usage .= "\t\t$0 -v -i sample.pat -o mypat.pat -r 16 \n";
    $usage .= "\n";
    print($usage);
    return;
}

sub getFile {

=head2 Get Waveform File

The B<getFile> sub-routine will open the input waveform file in signed 2's
complement format and store the data into a hash as an array. In addition,
the B<getFile> will also store the total number of lines in the file into
a hash.
```

```
=cut

    ############################################################################
    # Get Waveform File:
    #
    #  The sub-routine getFile() will open the Waveform file
    #  and read its contents into an array. It will also determine
    #  the file length. The following parameters are created
    #
    #   * fileData:     @dataA
    #   * fileLen:      scalar(@dataA)
    #
    #  Usage: $pat_rH = getFile(\%patH);
    #
    ############################################################################
    my ($pat_rH) = shift;           # Read in user's variable.
    my (%patH) = %{ $pat_rH };       # De-reference hash.
    my ($debug) = $patH{'debug'};    # Print out Debug Info.

    ############################################################################
    # Open the waveform file, and read the results into an array for
    # manipulating the data array. Strip new lines and carriage returns
    # from remove string array, and initialize for loop variables. Close file
    # when done.
    ############################################################################
    open(inF, "<", $patH{ 'wave_in' }) or dienice ("$patH{ 'wave_in' } open failed");
    my @dataA = <inF>;
    close(inF);

    # Strip newlines
    foreach my $i (@dataA) {
        chomp($i); # Remove any \n line-feeds.
        $i =~ s/\r//g; # Remove any \r carriage-returns.
    }
    push (@{ $patH{ 'waveFileIn' } }, @dataA);

    ############################################################################
    # Determine number of lines
    ############################################################################
    $patH{ 'waveFileLen' } = scalar(@{ $patH{ 'waveFileIn' } });

    print("\n\n") if $debug;
    print("Total number of lines: $patH{ 'waveFileLen' }\n") if $debug;
    print("\n\n") if $debug;

    ############################################################################
    # Return data to user
    ############################################################################
    return \%patH;
}

sub parseFile {

=head2 Parse Waveform File

The B<parseFile> sub-routine will parse the waveform file to find the #begin
and #end tags, which identify the location of the waveform data in the file
array. After locating the waveform data, B<parseFile> will then extract the
waveform data from the file array into its own array and store both the array
and the number of waveform samples in a hash.

=cut

    ###########################################################################
    # Parse Waveform File
    #
    #  The sub-routine parseFile() will parse the input Waveform File
    #  and retrieve the following information:
    #
    #               #begin
    #               #end
    #
    #  This sub-routine will also extract the actual pattern data into an
    #  array for converting from decimal to hexadecimal.
    #
    #  Usage: $pat_rH = parseFile(\%patH);
    #
    ###########################################################################
    my ($pat_rH) = shift;            # Read in user's variable.
    my (%patH) = %{ $pat_rH };       # De-reference hash.
    my ($debug) = $patH{'debug'};    # Print out Debug Info.

    ###########################################################################
    # Search through $file for keywords.
    ###########################################################################
    my ($i) = 0;
    my ($j) = 0;
    my ($beginFound);
    my ($endFound);

    for ($i=0; $i < $patH{ 'waveFileLen' }; $i++) {
        if (${ $patH{ 'waveFileIn' } }[$i] =~ m/#begin/) {
            $beginFound = $i;
            print("Begin Line Number: $beginFound\n") if $debug;
        }
        if (${ $patH{ 'waveFileIn' } }[$i] =~ m/#end/) {
            $endFound = $i;
            print("End_Line_Number:_$endFound\n") if $debug;
        }
    }
    print("\n\n") if $debug;

    ############################################################################
    # Search through $file for waveform data and store into a data array.
    ############################################################################
    my (@wave_dataA);

    for ($j=($beginFound+1); $j < $endFound; $j++) {
        my ($tmp_data) = ${ $patH{ 'waveFileIn' } }[$j];
        $tmp_data =~ s/[\r\n]//;
        $tmp_data =~ s/[\r\n]//;
        push(@wave_dataA, $tmp_data);
    }
    my ($wave_len) = scalar(@wave_dataA);

    ############################################################################
    # Grab header from input file:
    ############################################################################
    push(@{ $patH{ 'Header' } }, @{ $patH{ 'waveFileIn' } }[0 .. $beginFound]);

    ############################################################################
    # Store variables into hash:
    ############################################################################
    $patH{ 'beginFound' } = $beginFound;
    $patH{ 'endFound' } = $endFound;
    $patH{ 'NumberSamples' } = $wave_len;
    push (@{ $patH{ 'waveDataDec' } }, @wave_dataA);

    ############################################################################
    # Return data to user
    ############################################################################
    return \%patH;
}

sub convDec2Hex {

=head2 Convert Signed 2's_Complement_Decimal_to_16-bit_Hexadecimal

The_B<convDec2Hex>_sub-routine_will_quantize,_scale,_and_convert_the_signed
2's complement decimal data into hexadecimal data. It will then write the
data to a waveform file suitable for loading onto the Measurement board.

=head3 ALGORITHM DESCRIPTION
```

```
=over 4

=item 1. Determine maximum value of a hexadecimal sample using the RES value provided by
    the user.

=over 4

=item a. C<$maxValue = 2**$RES>

=item b. The default RES value for the Measurment board DAC is 16.

=back

=item 2. Determine the scale coefficient for the quantized sample data.

=over 4

=item a. C<$maxScale = 2**$SCALE>

=item b. The default SCALE value for the Measurement board DAC is 15.

=back

=item 3. Grab a signed 2's complement decimal value from the data array.

=item 4. Quantize signed 2's complement decimal value to RES bits.

=over 4

=item a. C<$datQuant = ceil($datIn * $maxValue) / $maxValue>

=back

=item 5. Scale the quantized sample data by 2**SCALE.

=over 4

=item a. C<$datScale = $datQuant * ($maxScale - 1)>

=back

=item 6. Convert the quantized and scaled data to hexadecimal.

=over 4

=item a. C<$hexValue = dec2hex($datScale)>

=back
```

```
=item 7. Test hexadecimal value to ensure that it contains 4 hexadecimal digits.

=item 8. Store the hexadecimal value in an array in the order provided by the input
    waveform file.

=item 9. Determine if the hexadecimal waveform data meets the length requirements of the
    Measurement board and replicate the waveform if necessary.

=over 4

=item a. Waveform must be a minimum of 256 binary bits long.

=item b. Waveform length must be modulo-128.

=item c. Waveform length must meet maximum requirements depending on the waveform memory
    type. Block RAM can support up to 2**21 bits. QDR-II SRAM can support up to 2**25 bits
    .

=back

=item 10. Calculate waveform stop address.

=over 4

=item a. C<$stopAddr = ( ($LENBITS / 128) - 1 )>

=back

=item 11. Determine the mark density, or ratio of 1's and 0's, of the waveform.

=item 12. Format waveform file header, and write to an output waveform file.

=item 13. Write the 16-bit hexadecimal value to an output waveform file.

=item 14. Write the replicated waveform to a file in decimal format for verification of
    waveform in Matlab.

=back

=cut

    ##############################################################################
    # Convert Decimal to Hexadecimal:
    #
    #  The sub-routine convDec2Hex() will convert the signed 2's complement
    #  decimal data to hexadecimal data.
    #
```

```
#  Usage: $pat_rH = convDec2Hex($patH);
#
################################################################################
my ($pat_rH)     = shift;                     # Read in user's variable.
my (%patH)       = %{ $pat_rH };              # De-reference hash.
my ($nsamps)     = $patH{'NumberSamples'};    # Number of Waveform Samples
my ($res)        = $patH{'resolution'};       # DAC resolution
my ($scale)      = $patH{'scale'};            # Scale factor

################################################################################
# Convert Waveform Data from Signed 2's Complement Decimal to Hexadecimal
################################################################################
my ($maxValue) = (2**$res);
print("Maximum Value: $maxValue\n") if $debug;
my (@waveHexA);
my ($maxOValue) = (2**($scale));

my ($j) = 0;
for ($j=0; $j<$nsamps; $j++) {
    print("* Sample Number: $j\n") if $debug;
    # Grab current sample:
    my ($datIn) = ${ $patH{ 'waveDataDec' } }[$j];
    print("Sample Data ($j): $datIn\n") if $debug;

    # Quantize data to 'resolution' bits:
    #my ($numerator) = round($datIn*($maxValue));
    my ($numerator) = ceil($datIn*($maxValue));
    print("Numerator (rounded to nearest integer): $numerator\n") if $debug;
    my ($datQuant) = $numerator/($maxValue);
    print("Quantized Data (rounded to nearest integer): $datQuant\n") if $debug;

    # Convert from decimal to hexadecimal:
    my $hexValue = dec2hex($datQuant*($maxOValue-1));
    print("Quantized Data (Hex): $hexValue\n") if $debug;

    # Test Hexadecimal Value to see if there are 4 zeros:
    my (@test_hexA) = testhex($hexValue,$debug);
    # This is the value we would send to the Data Path FPGA.
    my ($test_hex4) = $test_hexA[1];

    # Grab 4 LSB hex values:
    my (@hexA) = split(//,$test_hex4);
    my ($hexLen) = scalar(@hexA);
    print("Number of hex digits: $hexLen\n") if $debug;
    my ($bit0) = $hexLen-1;
    my ($bit3) = $hexLen-4;
    print("Grabbing bits $bit3 to $bit0\n") if $debug;
    my ($newHexValue) = join("",@hexA[$bit3 .. $bit0]);
```

```
        print("Quantized Data (Hex, 4-digits): $newHexValue\n") if $debug;
        print("\n\n") if $debug;
        push(@waveHexA, $newHexValue);
    }
    my ($waveHexLen) = scalar(@waveHexA);
    print("Waveform Length: $waveHexLen\n");

    ##########################################################################
    # Write Hexadecimal 2's Complement Waveform Data to File:
    ##########################################################################
    my $newfile = $patH{ 'wave_out' };

    # check to make sure that the file doesn't exist.
    die "Oops! A file called '$newfile' already exists.\n" if -e $newfile;
    # Open Hex File:
    open(my $outF, ">", $newfile);

    # Replicate Array if Necessary:
    my (%repH,$rep_rH);
    my (@waveHexFinalA);
    push (@{ $repH{ 'waveFileIn' } }, @waveHexA);
    $repH{ 'debug' } = $debug;
    # Call repCalc():
    %repH = %{ repCalc(\%repH) };
    printf("Just finisned Replication Check\n");

    # Grab new Wavefile:
#     push(@waveHexFinalA, @{ $repH{'waveFileOut'} } );
    @waveHexFinalA = @{ $repH{'waveFileOut'} };
    # Grab Number of Samples:
    my ($patLen) = $repH{'numSamples'};
    print("Waveform Number of Samples: $patLen\n");
    # Grab Number of Bits:
    my ($numBits) = $repH{'numBits'};
    print("Waveform Number of Bits: $numBits\n");
    # Grab Stop Address in Hex:
    my ($stopAddr) = $repH{'stopAddr'};
    print("Waveform Stop Address: $stopAddr\n");
    # Grab Waveform Type:
    my $BORS = $repH{ 'waveType' };
    # Grab Waveform Mark Density:
    my $mkDen = $repH{'markDensity'};

    my ($mydesc) = 'empty';
    foreach my $h (@{ $patH{ 'Header' } }) {
        if ($h =~ m/desc/) {
            $mydesc = $h;
        }
```

```
}
my ($headFileName) = $newfile;
$headFileName =~ s/.*\///;
printf($outF "###################################################################\n");
printf($outF "#patternname=$headFileName\n");
printf($outF "#patterntype=$BORS\n");
printf($outF "#patternlength=$numBits\n");
printf($outF "#readstartaddressa=0x0\n");
printf($outF "#readstopaddressa=0x$stopAddr\n");
printf($outF "#readstartaddressb=0\n");
printf($outF "#readstopaddressb=0\n");
printf($outF "#triggerword=0\n");
printf($outF "#patternstatistics=\n");
printf($outF "#density=%2.4f\n", $mkDen);
printf($outF "#bitshift=no\n");
printf($outF "#bitshiftindex=0 \n");
printf($outF "#crc=???\n");
printf($outF "$mydesc\n");
printf($outF "###################################################################\n");
printf($outF "#begin\n");

my ($i) = 0;
for ($i=0; $i < $patLen; $i+=2) {
    printf($outF "$waveHexFinalA[$i]");
    printf($outF "$waveHexFinalA[$i+1]");
    printf($outF "\n");
}

printf($outF "#end\n");

close (outF);

########################################################################
# Write Decimal Waveform Data to Matlab File:
########################################################################
my $newfile = $patH{'wave_out'};
$newfile =~ s/\.pat//;
$newfile .= "_dec.m";

# check to make sure that the file doesn't exist.
die "Oops! A file called '$newfile' already exists.\n" if -e $newfile;
# Open Hex File:
open(my $out2F, ">", $newfile);

for ($i=0; $i < $waveHexLen; $i++) {
    my ($tmpD) = unpack('s',pack 's', hex($waveHexA[$i]));
    my ($index) = $i+1;
    printf($out2F "wave_p($index) = $tmpD;");
```

```
        printf($out2F "\n");
    }

    close(out2F);

    ##########################################################################
    # Return data to user
    ##########################################################################
    return \%patH;
}

sub testhex {

=head2 Test Hexadecimal Values

The B<testhex> sub-routine will receive a single
hexadecimal value. If the hexadecimal value is less
than 4 digits, then B<testhex> will pre-pend the
appropriate number of zeros in order to provide a
complete 16-bit hexadecimal value. An error flag is
also provided to determine if the hex value is more
than 4 digits.

=cut

    ##########################################################################
    # Test Hexadecimal Values:
    #
    #  The sub-routine testhex() will receive a single
    #  hexadecimal value. If the hexadecimal value is less
    #  than 4 digits, then testhex() will prepend the
    #  appropriate number of zeros in order to provide a
    #  complete 16-bit hexadecimal value. An error flag is
    #  also provided to determine if the hex value is more
    #  than 4 digits.
    #
    #   @hexOut = (orig, hex4, len, hflag);
    #
    #  Usage: my (@hexOut) = testhex($hexIn,$debug);
    #
    ##########################################################################
    my ($hexIn) = shift;
    my ($debug) = shift;
    my (@hexInA) = split(//,$hexIn);
    my ($hexInALen) = scalar(@hexInA);
    print("Hex In: $hexIn\n") if ($debug);
    print("Hex Length In: $hexInALen\n") if ($debug);
    my (@hexOut);
    my ($i);
    my ($hexDiff) = 4-$hexInALen;
    my ($hFlag) = 0;

    if ($hexDiff == 0) {
        print("hex length is 4\n") if ($debug);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
    } elsif ($hexDiff > 0) {
        for ($i=0; $i<$hexDiff; $i++) {
            unshift(@hexInA,0);
        }
        push(@hexOut, $hexIn);
        push(@hexOut, join("", @hexInA));
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
        print("hex length is $hexInALen, add $hexDiff zeros to pad to 4\n") if ($debug);
        print("New Hex: $hexOut[1]\n") if ($debug);
    } else {
        $hFlag = 1;
        push(@hexOut, $hexIn);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
        print("hex length is: $hexInALen\n") if $debug;
    }

    print Dumper(@hexOut) if $debug;

    return(@hexOut);
}

sub repCalc {

=head2 Calculate the Replicated Waveform Length

The B<repCalc> sub-routine will receive a hexadecimal waveform array, and
determine if its overall bit length meets the length requirements of the
Measurement board. If the requirements are not met, then the B<repCalc>
sub-routine will replicate the pattern as necessary.

=head3 ALGORITHM DESCRIPTION

=over 4
```

```
=item 1. Determine if the hexadecimal waveform data meets the minimum length requirement
    of 256 bits.

=over 4

=item a. Replicate the hexadecimal waveform data such that it meets the minimum length
    requirement.

=back

=item 2. Is the waveform length modulo-128?

=item 3. Is the waveform length modulo-32?

=item 4. Is the waveform length modulo-2?

=item 5. Determine if the hexadecimal waveform data needs to be further replicated.

=over 4

=item a. If the waveform length is modulo-32 but not modulo-128, then the waveform must be
     replicated by C<$REPNUM = ($LENBITS * 16) % 128>.

=item b. If the waveform length is not modulo-32 and not modulo-128, then the waveform
    must be replicated by C<$REPNUM = 128>.

=item c. If the waveform length is modulo-128, then the waveform does not require
    replication.

=back

=item 6. Replicate the hexadecimal waveform data if necessary.

=item 7. Determine if the hexadecimal waveform data meets the maximum length requirements
    depending on the waveform type selected.

=over 4

=item a.  Block RAM can support up to 2**21 bits.

=item b.  QDR-II SRAM can support up to 2**25 bits.

=back

=item 8. Calculate waveform stop address.

=over 4
```

```
=item a. C<$stopAddr = ( ($LENBITS / 128) - 1 )>

=back

=item 9. Determine the mark density, or ratio of ones to total bits, of the waveform.

=back

=cut

    ##########################################################################
    # Calculate the Replicated Length:
    #
    #  The sub-routine repCalc() will receive a waveform array, and determine
    #  if the waveform array needs to be replicated such that it meets the
    #  waveform requirements of the Data Path FPGA:
    #
    #  Usage: my (@AOUT) = repCalc(@AIN);
    #
    ##########################################################################
    my ($rep_rH) = shift;            # Read in user's variable.
    my (%repH)   = %{ $rep_rH };     # De-reference hash.
    my ($debug)  = $repH{'debug'};   # Print out Debug Info.
    my (@AIN);
    push(@AIN, @{ $repH{ 'waveFileIn' } });

    ##########################################################################
    # Constants and Variables:
    ##########################################################################
    my (%repH, $rep_rH);
    my ($dacres) = 16;

    # Calculate length of waveform in bits:
    my ($Alenin) = scalar(@AIN)*$dacres;
    print("Array Length: $Alenin\n");

    # Check to see if waveform is less than or 256-bits:
    my (@Atmp);
    my ($i);
    my ($Alen);
    push(@Atmp, @AIN);
    if ($Alenin < 256) {
        print("Array length less than 256-bits.\n");
        print("Actual Array Length: $Alenin.\n");
        for ($i = 0; $i < 129; $i++) {
            my ($Alentmp) = scalar(@Atmp)*$dacres;
            print("Length = $Alentmp.\n");
            if ($Alentmp < 256) {
                push(@Atmp, @Atmp);
            } else {
                $Alen = $Alentmp;
                print("Replicated $i times to meet minimum bit length of 256.\n");
                print("New length of waveform: $Alen\n");
                $i = 129;
            }
            print("Iteration $i\n");
        }
    }
    my ($Alentmp2) = scalar(@Atmp);
    print("Length of Atmp: $Alentmp2\n");

    print Dumper(@Atmp) if $debug;

    # Determine replication factor:
    my ($REPNUM) = 1;
    my ($Alen_mod_128) = $Alentmp2*$dacres % 128;
    my ($Alen_mod_32) = $Alentmp2*$dacres % 32;
    my ($Alen_mod_2) = $Alentmp2*$dacres % 2;
    printf("Length: %d, Length mod 128: %d\n", $Alentmp2, $Alen_mod_128);
    if (($Alen_mod_128 ne 0) and ($Alen_mod_32 eq 0) and ($Alen_mod_2 eq 0)) {
        print("I am not mod 128, but I am mod 32.\n");
        $REPNUM = $Alentmp2*16 % 128;
    } elsif ($Alen_mod_128 ne 0) {
        print("I am not mod 128.\n");
        $REPNUM = 128;
    } else {
        print("I meet the requirements.\n");
        $REPNUM = 1;
    }

    # Replicate array 'REPNUM' times:
    my (@AOUT);
    my ($k) = 1;
    if ($REPNUM eq 1) {
        print("No replication necessary.\n");
        push(@AOUT, @Atmp);
    } else {
        print("Replicate Array $REPNUM times.\n");
        my ($REPNUM) = $REPNUM/8;
        if ($REPNUM < 1) {
            print("Error: replication factor less than 1.\n");
            $REPNUM = 1;
        }
        print("Replicate Array $REPNUM times.\n");
        for($k=1; $k <= $REPNUM; $k++) {
            push(@AOUT, @Atmp);
        }
    }
    print Dumper(@AOUT) if $debug;

    # Determine if length will fit in Block RAM or QDR-II SRAM:
    my ($SAMPLES) = scalar(@AOUT);
    my ($LENBITS) = $SAMPLES*16;
    my ($BORS);
    printf("Final Waveform Length, samples: %d, bits: %d\n", $SAMPLES, $LENBITS);
    if ($patH{ 'bram_nsram' } =~ /1/) {
        if ($LENBITS > 2**21) {
            print("* ERROR: Waveform will not fit in Block RAM.\n");
            exit;
        } else {
            print("Waveform will fit in Block RAM.\n");
        }
        $BORS = 'bram';
    }
    if ($patH{ 'bram_nsram' }=~ /0/) {
        if ($LENBITS > 2**25) {
            print("* ERROR: Waveform will not fit in QDR-II SRAM.\n");
            exit;
        } else {
            print("Waveform will fit in QDR-II SRAM.\n");
        }
        $BORS = 'sram';
    }

    # Calculate Stop Address:
    my ($stopAddr);
    my ($stopAddrHex);
    if ($BORS =~ m/bram/) {
        print("Waveform will fit in Block RAM.\n");
        $stopAddr = (($LENBITS/128)-1);
        $stopAddrHex = dec2hex($stopAddr);
    } else {
        print("Waveform will fit in QDR-II SRAM.\n");
        $stopAddr = (($LENBITS/128)-1);
        $stopAddrHex = dec2hex($stopAddr);
    }
    print("Stop Address: $stopAddr\n");
    print("Stop Address (Hex): $stopAddrHex\n");

#     push (@{ $repH{ 'waveFileOut' } }, @AOUT);
    $repH{ 'waveFileOut' } = \@AOUT;
    $repH{ 'waveType' } = $BORS;
    $repH{ 'numSamples' } = $SAMPLES;
    $repH{ 'numBits' } = $LENBITS;
    $repH{ 'stopAddr' } = $stopAddrHex;

    # Calculate Mark Density:
#     %repH = %{ calcMD(\%repH) };
    $repH{'markDensity'} = "500";

    ###########################################################################
    # Return data to user
    ###########################################################################
    return \%repH;
}

sub calcMD {

=head2 Calculate Mark Density of Waveform File

The B<calcMD> sub-routine will determine the total number of logic ones in
the waveform file, and calculate the mark density. Mark density is the ratio
of logic ones to the total number of bits.

=cut

    ###########################################################################
    # Calculate Mark Density of Waveform File:
    #
    #  The sub-routine calcMD() will calculate the mark density of the
    #  Waveform File.
    #
    #  Usage: $pat_rH = calcMD(\%patH);
    #
    ###########################################################################
    my ($pat_rH) = shift;             # Read in user's variable.
    my (%patH)   = %{ $pat_rH };      # De-reference hash.
    my ($debug)  = $patH{'debug'};    # Print out Debug Info.
    my (@dataA) = @{ $patH{'waveFileOut'} };

    ###########################################################################
    # Determine number of 1's and Total Bits:
    ###########################################################################
    my ($i) = 0;
    my ($j) = 0;
    my ($cnt_ones) = 0;
    my ($cnt_bits) = 0;
    my ($tmpH);
    my ($tmpD);
    my ($tmpB);
    my (@tmpBA);
    my (@tmpBinA);
    printf("Calculate Mark Density\n");
    for ($i=0; $i < $patH{ 'numSamples' };$i++) {
        $tmpH = $dataA[$i];
        my (@tmpHA) = split(//,$tmpH);
        foreach my $j (@tmpHA) {
            push(@tmpBinA, hex2bin($j));
        }
        my ($tmpJW) = join("",@tmpBinA);
        @tmpBA = split(//,$tmpJW);
        my ($Blen) = scalar(@tmpBA);
        print("Blen: $Blen\n") if $debug;
        for my $x (@tmpBA) {
            $cnt_ones += $x;
            $cnt_bits += 1;
        }
#          printf("index: %d\n", $i);
    }
    printf("Done calculating Mark Density\n");

    print("Total number of bits: $cnt_bits\n");
    print("Total number of ones: $cnt_ones\n");

    my ($mkDen) = ($cnt_ones/$cnt_bits)*100;
    print("Mark Density: $mkDen\n");
    $patH{'markDensity'} = $mkDen;

    ##########################################################################
    # Return data to user
    ##########################################################################
    return \%patH;
}

sub dec2hex($) {

=head2 Decimal to Hexadecimal Conversion

The B<dec2hex> sub-routine will convert a decimal value into its
hexadecimal equivalent.

=cut

    ##########################################################################
    # Decimal to Hexadecimal Conversion:
    #
    #  The sub-routine dec2hex() will convert a decimal value into its
    #  hexadecimal equivalent.
    #
    #  Usage: $hout = dec2hex($din);
    #
    ###########################################################################
    my( $dec ) = shift;
    return sprintf("%x", $dec );
}

sub hex2bin {

=head2 Hexadecimal to Binary Conversion

The B<hex2bin> sub-routine will convert a hexadecimal value into its
binary equivalent.

=cut

    ###########################################################################
    # Hexadecimal to Binary Conversion:
    #
    #  The sub-routine hex2bin() will convert a hexadecimal value into its
    #  binary equivalent.
    #
    #  Usage: $bout = hex2bin($hin);
    #
    ###########################################################################
    my $hex = shift;
    my $binary;
    my %h2b = (0 => "0000", 1 => "0001", 2 => "0010", 3 => "0011",
               4 => "0100", 5 => "0101", 6 => "0110", 7 => "0111",
               8 => "1000", 9 => "1001", a => "1010", b => "1011",
               c => "1100", d => "1101", e => "1110", f => "1111",
              );
    ($binary = $hex) =~ s/(.)/$h2b{lc $1}/g;
    return ($binary);
}
```

Appendix C

Square Waveform Generation Perl Script

This chapter describes the square waveform generation Perl script, and how it is used to generate square waveforms.

C.1 NAME

square.pl - Square Waveform File Generation Script

C.2 SYNOPSIS

```
square.pl [-h] [-v] [-f <FILE>] [-s <FREQ>]

Help Options:
-h             Print Help.
-v             Verbose: Print Debug Information.
-f <FILE>      New Matlab filename.
-s <FREQ>      Requested Frequency.

Example:
    ./square.pl -v -f sample.m -s 1e6
```

C.3 OPTIONS

-h

Show the brief help information.

-v

Show debug information.

-f FILE

Square waveform Matlab filename. The pattern file will use the same filename, but with a *.pat* file extension.

-s FREQ

Square waveform frequency in the range of DC to 250 MHz. A sample rate of 500 MHz is assumed.

C.4 DESCRIPTION

square.pl is used to generate a square waveform file, which can be converted to the appropriate format for playback on the Measurement board by the sig2hex.pl script. The **square.pl** script will generate two files:

- Matlab waveform file for plotting and further analysis.
- Waveform file for conversion by the sig2hex.pl script.

ALGORITHM DESCRIPTION

Square Waveform Parameters

The **square.pl** script will use the FREQ parameter to calculate the following parameters:

- Square Waveform Period (`$reqPeriod`).
- Number of samples in the square waveform file (`$numPoints`).
- Number of samples in the positive pulse of the square waveform (`$posPoints`).
- Number of samples in the negative pulse of the square waveform (`$negPoints`).

In addition, the **square.pl** script will define the following parameters:

- Sample frequency (`$fs`).
- Sample period (`$sampPeriod`).

- High-speed DAC resolution (`$dacres`).
- Maximum signed 2's complement decimal value (`$res`).

File Headers

The **square.pl** script will then use the calculated variables to create the file headers for both the Matlab file and the waveform file. The header of the waveform file requires additional parameters to be calculated:

- Waveform length in bits (`$numBits`).
- Waveform Stop Address (`$stopAddr`).

The waveform length is calculated using the high-speed DAC resolution and the number of sample points required for the square waveform.

Positive and Negative Pulse Generation

Using the `$posPoints` and `$negPoints` parameters previously calculated, the **square.pl** script will generate the positive and negative pulses of the square waveform using the following steps:

1. Store a sample value of 0V.
2. Store the maximum value of the high-speed DAC `$posPoints` times.
3. Store a sample value of 0V.
4. Store the minimum value of the high-speed DAC `$negPoints` times.

File Footers

The **square.pl** script will create a file header for both the Matlab file and the waveform file. The Matlab file will contain code to plot the generated square waveform. The waveform file will contain the tag "#end", which indicates to the sig2hex.pl script that the end of the data payload has been reached.

C.5 SUBROUTINES

Decimal to Hexadecimal Conversion

The **dec2hex** sub-routine will convert a decimal value into its hexadecimal equivalent.

Test Hexadecimal Values

The **testhex** sub-routine will receive a single hexadecimal value. If the hexadecimal value is less than 4 digits, then **testhex** will pre-pend the appropriate number of zeros in order to provide a complete 16-bit hexadecimal value. An error flag is also provided to determine if the hex value is more than 4 digits.

C.6 CODE

Listing C.1: Square Waveform Generation Perl Script

```
#!/usr/bin/env perl
# vim:ts=4:sw=4:expandtab:cindent

#*****************************************************************
#
# square.pl module
#
#*****************************************************************
#
# VCL Confidential Copyright   2009 UC Davis, ECE Department
#
#*****************************************************************
#
# created on:    11/25/2009
# created by:    jwwebb
# last edit on: $DateTime: $
# last edit by: $Author: $
# revision: $Revision: $
# comments: Generated
#
#*****************************************************************
# Revision List:
#
#        1.0 11/25/2009  Initial release
#        1.1 01/01/2010  Add Perl POD documentation
#
#*****************************************************************
# Square Waveform File Generation
#
#  This utility is intended to generate a square waveform for
#  playback on the Measurement board.
#
#  Usage Information:
#
```

```
#        Usage: ./square.pl [-h] [-v] [-f <FILE>] [-s <FREQ>]
#
#          -h         Print Help.
#          -v         Verbose: Print Debug Information.
#          -f <FILE>    New Matlab filename.
#          -s <FREQ>    Requested Frequency.
#
#          Example:
#              ./square.pl -v -f sample.m -s 1e6
#
#****************************************************************
use strict;

#****************************************************************
# CPAN Modules
#****************************************************************
use Getopt::Std;
use FileHandle;
use POSIX;
use Fcntl;                           # File control (lock, etc...)
use SDBM_File;                       # Simple database
use Carp;                            # Warnings/Errors for modules
use File::Basename;
use File::Path;
use Data::Dumper;

#****************************************************************
# Centellax Modules
#****************************************************************
use CentellaxATE;                     # System setup
use meas_utils;

#****************************************************************
# Constants and Variables:
#****************************************************************
my (%opts)=();
my ($file);
my ($freq);
my ($debug);

#****************************************************************
# Retrieve command line argument
#****************************************************************
getopts('hvf:s:',\%opts);

#check for valid combination command-line arguments
if ($opts{h} || !$opts{f}) {
    print_usage();
    exit;
}

# parse command-line arguments
$file = $opts{f};
$freq = $opts{s};
$debug = $opts{v};
=pod

=head1 NAME

B<square.pl> - Square Waveform File Generation Script

=head1 SYNOPSIS

  square.pl [-h] [-v] [-f <FILE>] [-s <FREQ>]

  Help Options:
  -h          Print Help.
  -v          Verbose: Print Debug Information.
  -f <FILE> New Matlab filename.
  -s <FREQ> Requested Frequency.

  Example:
      ./square.pl -v -f sample.m -s 1e6

=head1 OPTIONS

=over 8

=item B<-h>

Show the brief help information.

=item B<-v>

Show debug information.

=item B<-f FILE>

Square waveform Matlab filename. The pattern file will use the same filename, but with a I
    <.pat> file extension.

=item B<-s FREQ>

Square waveform frequency in the range of DC to 250 MHz. A sample rate of 500 MHz is
    assumed.
```

```
=back

=head1 DESCRIPTION

B<square.pl> is used to generate a square waveform file, which can be converted
to the appropriate format for playback on the Measurement board by the sig2hex.pl script.
The B<square.pl> script will generate two files:

=over 4

=item * Matlab waveform file for plotting and further analysis.

=item * Waveform file for conversion by the sig2hex.pl script.

=back

=cut

# check to make sure that the Matlab file doesn't exist.
die "Oops! A file called '$file' already exists.\n" if -e $file;
open (SF1,">$file") || die "Can't open $file!  $!\n";

# check to make sure that the Pattern file doesn't exist.
my ($patfile) = $file;
$patfile =~ s/\..*$/.pat/;
die "Oops! A file called '$patfile' already exists.\n" if -e $patfile;
open (SF2,">$patfile") || die "Can't open $patfile!  $!\n";

##########################################################################
# AutoFlush FileHandles:
##########################################################################
autoflush SF1 1;                # Immediate writes
autoflush SF2 1;                # Immediate writes
autoflush STDOUT 1;             # Immediate writes

=head2 ALGORITHM DESCRIPTION

=head3 Square Waveform Parameters

The B<square.pl> script will use the FREQ parameter to calculate
the following parameters:

=over 4

=item * Square Waveform Period (C<$reqPeriod>).

=item * Number of samples in the square waveform file (C<$numPoints>).
```

```
=item * Number of samples in the positive pulse of the square waveform (C<$posPoints>).

=item * Number of samples in the negative pulse of the square waveform (C<$negPoints>).

=back

In addition, the B<square.pl> script will define the following
parameters:

=over 4

=item * Sample frequency (C<$fs>).

=item * Sample period (C<$sampPeriod>).

=item * High-speed DAC resolution (C<$dacres>).

=item * Maximum signed 2's complement decimal value (C<$res>).

=back

=cut

#***************************************************************
# Square Waveform Parameters:
#***************************************************************
my ($res) = 15;
my ($dacres) = 16;
my ($fs) = 500e6;
$freq += 0;
my ($sampPeriod) = 1/$fs;
my ($reqPeriod) = 1/$freq;
my ($numPoints) = $reqPeriod/$sampPeriod;
my ($posPoints) = ceil($numPoints/2);
my ($negPoints) = ceil($numPoints/2);

# Print Parameters:
select(STDOUT);
printf("Sample Period:     % .4e\n", $sampPeriod);
printf("Request Period:    % .4e\n", $reqPeriod);
printf("Positive Ramp:     % .4f\n", $posPoints);
printf("Negative Ramp:     % .4f\n", $negPoints);
printf("Number Points:     % .4f\n", $numPoints);

=head3 File Headers

The B<square.pl> script will then use the calculated variables
to create the file headers for both the Matlab file and the
```

```
waveform␣file.␣The␣header␣of␣the␣waveform␣file␣requires␣additional
parameters␣to␣be␣calculated:

=over␣4

=item␣*␣Waveform␣length␣in␣bits␣(C<$numBits>).

=item␣*␣Waveform␣Stop␣Address␣(C<$stopAddr>).

=back

The␣waveform␣length␣is␣calculated␣using␣the␣high-speed␣DAC␣resolution
and␣the␣number␣of␣sample␣points␣required␣for␣the␣square␣waveform.

=cut

#****************************************************************
#␣Write␣Matlab␣Header:
#****************************************************************
my␣($matHead)␣=␣<<"MATHEAD";
%****************************************************************
%
%␣VCL␣Confidential␣Copyright␣ ␣2011␣UC␣Davis,␣ECE␣Department
%
%****************************************************************
%
%␣Module:␣␣␣␣␣␣␣␣$file
%␣Created␣on:␣␣␣␣␣␣␣␣Sun␣Jan␣01␣16:27:11␣2011
%\%\␣Executed␣by:␣␣␣␣␣␣␣␣jwwebb
%
%\%\␣board␣name:␣␣␣␣␣␣␣␣␣MSEE␣Thesis␣Measurement␣Board
%\%\␣board␣number:␣␣␣␣␣␣␣p342
%\%\␣board␣rev:␣␣␣␣␣␣001
%
%****************************************************************
%
%␣Column␣Headers␣for␣measurement
%␣␣␣\$1␣=␣time;␣Time␣(sec)
%␣␣␣\$2␣=␣volt;␣Voltage␣(V)
%
clear;clc;close␣all;
PrintOnEps␣=␣1;

square(1)␣=␣0.0000;
MATHEAD

select(SF1);
printf("$matHead");
```

```
select(STDOUT);

#****************************************************************
# Write Pattern Header:
#****************************************************************
my ($numBits) = ($numPoints + 2)*$dacres; # account for the two samples at 0V.
my ($mod128) = $numBits % 128;
my ($patLen) = ($mod128 > 0) ? $mod128*$numBits : $numBits;
my ($stopAddr) = ($patLen/128) - 1;
my ($stopAddrH) = dec2hex($stopAddr);
my ($dispFs) = Suffix($fs, "Hz", 1);
my ($dispFc) = Suffix($freq, "Hz", 1);
my ($patHead) = <<"PATHEAD";
################################################################
#patternname=$patfile
#patterntype=sram
#patternlength=$patLen
#readstartaddressa=0x0
#readstopaddressa=0x$stopAddrH
#readstartaddressb=0
#readstopaddressb=0
#triggerword=0
#patternstatistics=
#density=0.750
#bitshift=no
#bitshiftindex=0
#crc=???
#description=Square Waveform: Fc = $dispFc, Fs = $dispFs
################################################################
#begin
0.000000000000000
PATHEAD

select(SF2);
printf("$patHead");
select(STDOUT);

=head3 Positive and Negative Pulse Generation

Using the C<$posPoints> and C<$negPoints> parameters previously
calculated, the B<square.pl> script will generate the positive
and negative pulses of the square waveform using the following steps:

=over 4

=item 1. Store a sample value of 0V.

=item 2. Store the maximum value of the high-speed DAC C<$posPoints> times.
```

```
=item␣3.␣Store␣a␣sample␣value␣of␣0V.

=item␣4.␣Store␣the␣minimum␣value␣of␣the␣high-speed␣DAC␣C<$negPoints>␣times.

=back

=cut

#****************************************************************
#␣Generate␣Positive␣Pulse:
#****************************************************************
my␣($pidx)␣=␣2;
for␣(my␣$p␣=␣0;␣$p␣<␣$posPoints;␣$p++)␣{
␣␣␣␣select(STDOUT);
␣␣␣␣printf("I:␣%d,␣P:␣%.0f\n",␣$pidx,␣$p)␣if␣$debug;
␣␣␣␣select(SF1);
␣␣␣␣printf("square(%d)␣=␣%.4f;\n",␣$pidx,␣(2**$res));
␣␣␣␣select(SF2);
␣␣␣␣printf("%.15f\n",␣(2**$res)/(2**$res));
␣␣␣␣$pidx++;
}

select(STDOUT);
printf("I:␣%d,␣P:␣%.0f\n",␣$pidx,␣0)␣if␣$debug;
select(SF1);
printf("square(%d)␣=␣%.4f;\n",␣$pidx,␣0);
select(SF2);
printf("%.15f\n",␣0);
$pidx++;

#****************************************************************
#␣Generate␣Negative␣Pulse:
#****************************************************************
my␣($nidx)␣=␣$pidx;
for␣(my␣$n␣=␣0;␣$n␣<␣$negPoints;␣$n++)␣{
␣␣␣␣select(STDOUT);
␣␣␣␣printf("I:␣%d,␣N:␣%.0f\n",␣$nidx,␣$n)␣if␣$debug;
␣␣␣␣select(SF1);
␣␣␣␣printf("square(%d)␣=␣%.4f;\n",␣$nidx,␣-(2**$res));
␣␣␣␣select(SF2);
␣␣␣␣printf("%.15f\n",␣-(2**$res)/(2**$res));
␣␣␣␣$nidx++;
}

=head3␣File␣Footers

The␣B<square.pl>␣script␣will␣create␣a␣file␣header␣for␣both␣the␣Matlab␣file
```

```
and the waveform file. The Matlab file will contain code to
plot the generated square waveform. The waveform file will
contain the tag "#end", which indicates to the sig2hex.pl
script that the end of the data payload has been reached.

=cut

#*************************************************************
# Write Matlab Footer:
#*************************************************************
my ($matFoot) = <<"MATFOOT";



% Plot Waveforms: Time (us) vs Voltage (V)
figure(1);
set(gca,'FontSize',14);
plot(square,'-r','LineWidth',2);
xlabel('Time (us)','fontsize',14);
ylabel('Voltage (mV)','fontsize',14);
grid on;
axis([0 500 -1.1 1.1]);
if PrintOnEps
    png_file = sprintf('meas_sig_src_square_td.png');
    print('-dpng', png_file);
    eps_file = sprintf('meas_sig_src_square_td.eps');
    print('-depsc', eps_file);
end

MATFOOT

select(SF1);
printf("$matFoot\n");
select(STDOUT);

#*************************************************************
# Write Pattern Footer:
#*************************************************************
select(SF2);
printf("#end\n");
select(STDOUT);

exit;

=head1 SUBROUTINES

=cut
```

```
#***********************************************************
#␣Sub-routines
#***********************************************************

sub␣dienice␣{
␣␣␣␣my($errmsg)␣=␣@_;
␣␣␣␣print”$errmsg\n”;
␣␣␣␣exit;
}

sub␣print_usage␣{
␣␣␣␣my␣($usage);
␣␣␣␣$usage␣=␣”\nUsage:␣$0␣[-h]␣[-v]␣[-f␣<FILE>]␣[-s␣<FREQ>]\n”;
␣␣␣␣$usage␣.=␣”\n”;
␣␣␣␣$usage␣.=␣”\t-h\t\tPrint␣Help.\n”;
␣␣␣␣$usage␣.=␣”\t-v\t\tVerbose:␣Print␣Debug␣Information.\n”;
␣␣␣␣$usage␣.=␣”\t-f␣<FILE>\tNew␣Matlab␣filename.\n”;
␣␣␣␣$usage␣.=␣”\t-s␣<FREQ>\tRequested␣Frequency.\n”;
␣␣␣␣$usage␣.=␣”\n”;
␣␣␣␣$usage␣.=␣”\tExample:\n”;
␣␣␣␣$usage␣.=␣”\t\t$0␣-v␣-f␣sample.m␣-s␣1e6␣\n”;
␣␣␣␣$usage␣.=␣”\n”;
␣␣␣␣print($usage);
␣␣␣␣return;
}

sub␣dec2hex($)␣{

=head2␣Decimal␣to␣Hexadecimal␣Conversion

The␣B<dec2hex>␣sub-routine␣will␣convert␣a␣decimal␣value␣into␣its
hexadecimal␣equivalent.

=cut

␣␣␣␣#########################################################################
␣␣␣␣#␣Decimal␣to␣Hexadecimal␣Conversion:
␣␣␣␣#
␣␣␣␣#␣␣The␣sub-routine␣dec2hex()␣will␣convert␣a␣decimal␣value␣into␣its
␣␣␣␣#␣␣hexadecimal␣equivalent.
␣␣␣␣#
␣␣␣␣#␣␣Usage:␣$hout␣=␣dec2hex($din);
␣␣␣␣#
␣␣␣␣#########################################################################
␣␣␣␣my(␣$dec␣)␣=␣shift;
␣␣␣␣return␣sprintf(”%x”,␣$dec␣);
}
```

```
sub testhex {

=head2 Test Hexadecimal Values

The B<testhex> sub-routine will receive a single
hexadecimal value. If the hexadecimal value is less
than 4 digits, then B<testhex> will pre-pend the
appropriate number of zeros in order to provide a
complete 16-bit hexadecimal value. An error flag is
also provided to determine if the hex value is more
than 4 digits.

=cut

    ############################################################################
    # Test Hexadecimal Values:
    #
    #  The sub-routine testhex() will receive a single
    #  hexadecimal value. If the hexadecimal value is less
    #  than 4 digits, then testhex() will prepend the
    #  appropriate number of zeros in order to provide a
    #  complete 16-bit hexadecimal value. An error flag is
    #  also provided to determine if the hex value is more
    #  than 4 digits.
    #
    #   @hexOut = (orig, hex4, len, hflag);
    #
    #  Usage: my (@hexOut) = testhex($hexIn,$debug);
    #
    ############################################################################
    my ($hexIn) = shift;
    my ($debug) = shift;
    my (@hexInA) = split(//,$hexIn);
    my ($hexInALen) = scalar(@hexInA);
    print("* Hex In: $hexIn\n") if ($debug);
    print("* Hex Length In: $hexInALen\n") if ($debug);
    my (@hexOut);
    my ($i);
    my ($hexDiff) = 4-$hexInALen;
    my ($hFlag) = 0;

    if ($hexDiff == 0) {
        print("* hex length is 4\n") if ($debug);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
    } elsif ($hexDiff > 0) {
        for ($i=0; $i<$hexDiff; $i++) {
            unshift(@hexInA,0);
        }
        push(@hexOut, $hexIn);
        push(@hexOut, join("", @hexInA));
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
        print("* hex length is $hexInALen, add $hexDiff zeros to pad to 4\n") if ($debug);
        print("* New Hex: $hexOut[1]\n") if ($debug);
    } else {
        $hFlag = 1;
        push(@hexOut, $hexIn);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
        print("* hex length is: $hexInALen\n") if $debug;
    }

    print Dumper(@hexOut) if $debug;

    return(@hexOut);
}
```

Appendix D

Ramp Waveform Generation Perl Script

This chapter describes the ramp waveform generation Perl script, and how it is used to generate ramp waveforms.

D.1 NAME

ramp.pl - Ramp Waveform File Generation Script

D.2 SYNOPSIS

```
ramp.pl [-h] [-v] [-f <FILE>] [-s <FREQ>]

Help Options:
-h              Print Help.
-v              Verbose: Print Debug Information.
-f <FILE>       New Matlab filename.
-s <FREQ>       Requested Frequency.

Example:
    ./ramp.pl -v -f sample.m -s 1e6
```

D.3 OPTIONS

-h

Show the brief help information.

-v

Show debug information.

-f FILE

Ramp waveform Matlab filename. The pattern file will use the same filename, but with a *.pat* file extension.

-s FREQ

Ramp waveform frequency in the range of DC to 250 MHz. A sample rate of 500 MHz is assumed.

D.4 DESCRIPTION

ramp.pl is used to generate a ramp waveform file, which can be converted to the appropriate format for playback on the Measurement board by the sig2hex.pl script. The **ramp.pl** script will generate two files:

- Matlab waveform file for plotting and further analysis.
- Waveform file for conversion by the sig2hex.pl script.

ALGORITHM DESCRIPTION

Ramp Waveform Parameters

The **ramp.pl** script will use the FREQ parameter to calculate the following parameters:

- Ramp Waveform Period (`$reqPeriod`).
- Number of samples in the ramp waveform file (`$numPoints`).
- Number of samples in the positive incline of the ramp waveform (`$posPoints`).
- Number of samples in the negative decline of the ramp waveform (`$negPoints`).
- Sample spacing for the positive incline of the ramp waveform (`$posInc`).
- Sample spacing for the negative decline of the ramp waveform (`$negInc`).

In addition, the **ramp.pl** script will define the following parameters:

- Sample frequency (`$fs`).
- Sample period (`$sampPeriod`).
- High-speed DAC resolution (`$dacres`).
- Maximum signed 2's complement decimal value (`$res`).

File Headers

The **ramp.pl** script will then use the calculated variables to create the file headers for both the Matlab file and the waveform file. The header of the waveform file requires additional parameters to be calculated:

- Waveform length in bits (`$numBits`).
- Waveform Stop Address (`$stopAddr`).

The waveform length is calculated using the high-speed DAC resolution and the number of sample points required for the ramp waveform.

Positive and Negative Ramp Generation

Using the `$posInc` and `$negInc` parameters previously calculated, the **ramp.pl** script will generate the positive incline and negative decline of the ramp waveform using the following steps:

1. Starting at the minimum value of the high-speed DAC, increment in `$posInc` steps until the maximum high-speed DAC value is reached.
2. Starting at the maximum value of the high-speed DAC minus `$negInc`, decrement in `$negInc` steps until the minimum high-speed DAC value plus `$negInc` is reached.

File Footers

The **ramp.pl** script will create a file header for both the Matlab file and the waveform file. The Matlab file will contain code to plot the generated ramp waveform. The waveform file will contain the tag "#end", which indicates to the sig2hex.pl script that the end of the data payload has been reached.

D.5 SUBROUTINES

Decimal to Hexadecimal Conversion

The **dec2hex** sub-routine will convert a decimal value into its hexadecimal equivalent.

Test Hexadecimal Values

The **testhex** sub-routine will receive a single hexadecimal value. If the hexadecimal value is less than 4 digits, then **testhex** will pre-pend the appropriate number of zeros in order to provide a complete 16-bit hexadecimal value. An error flag is also provided to determine if the hex value is more than 4 digits.

D.6 CODE

Listing D.1: Ramp Waveform Generation Perl Script

```
#!/usr/bin/env perl
# vim:ts=4:sw=4:expandtab:cindent

#*****************************************************************
#
# ramp.pl module
#
#*****************************************************************
#
# VCL Confidential Copyright   2009 UC Davis, ECE Department
#
#*****************************************************************
#
# created on:   11/25/2009
# created by:   jwwebb
# last edit on: $DateTime: $
# last edit by: $Author: $
# revision: $Revision: $
# comments: Generated
#
#*****************************************************************
# Revision List:
#
#        1.0 11/25/2009  Initial release
#        1.1 01/01/2010  Add Perl POD documentation
#
#*****************************************************************
```

```
# Ramp Waveform File Generation
#
#  This utility is intended to generate a sawtooth waveform for
#  playback on the Measurement board.
#
#  Usage Information:
#
#          Usage: ./ramp.pl [-h] [-v] [-f <FILE>] [-s <FREQ>]
#
#             -h            Print Help.
#             -v            Verbose: Print Debug Information.
#             -f <FILE>     New Matlab filename.
#             -s <FREQ>     Requested Frequency.
#
#             Example:
#                  ./ramp.pl -v -f sample.m -s 1e6
#
#****************************************************************
use strict;

#****************************************************************
# CPAN Modules
#****************************************************************
use Getopt::Std;
use FileHandle;
use POSIX;
use Fcntl;                           # File control (lock, etc...)
use SDBM_File;                       # Simple database
use Carp;                            # Warnings/Errors for modules
use File::Basename;
use File::Path;
use Data::Dumper;

#****************************************************************
# Centellax Modules
#****************************************************************
use CentellaxATE;                   # System setup
use meas_utils;

#****************************************************************
# Constants and Variables:
#****************************************************************
my (%opts)=();
my ($file);
my ($freq);
my ($debug);

#****************************************************************
```

```
# Retrieve command line argument
#***************************************************************
getopts('hvf:s:',\%opts);

#check for valid combination command-line arguments
if ($opts{h} || !$opts{f}) {
    print_usage();
    exit;
}

# parse command-line arguments
$file = $opts{f};
$freq = $opts{s};
$debug = $opts{v};
=pod

=head1 NAME

B<ramp.pl> - Ramp Waveform File Generation Script

=head1 SYNOPSIS

  ramp.pl [-h] [-v] [-f <FILE>] [-s <FREQ>]

  Help Options:
  -h         Print Help.
  -v         Verbose: Print Debug Information.
  -f <FILE> New Matlab filename.
  -s <FREQ> Requested Frequency.

  Example:
      ./ramp.pl -v -f sample.m -s 1e6

=head1 OPTIONS

=over 8

=item B<-h>

Show the brief help information.

=item B<-v>

Show debug information.

=item B<-f FILE>
```

```
Ramp waveform Matlab filename. The pattern file will use the same filename, but with a I<.
    pat> file extension.

=item B<-s FREQ>

Ramp waveform frequency in the range of DC to 250 MHz. A sample rate of 500 MHz is assumed
    .

=back

=head1 DESCRIPTION

B<ramp.pl> is used to generate a ramp waveform file, which can be converted
to the appropriate format for playback on the Measurement board by the sig2hex.pl script.
The B<ramp.pl> script will generate two files:

=over 4

=item * Matlab waveform file for plotting and further analysis.

=item * Waveform file for conversion by the sig2hex.pl script.

=back

=cut

# check to make sure that the file doesn't exist.
die "Oops! A file called '$file' already exists.\n" if -e $file;
open (SF1,">$file") || die "Can't open $file!  $!\n";

# check to make sure that the file doesn't exist.
my ($patfile) = $file;
$patfile =~ s/\..*$/.pat/;
die "Oops! A file called '$patfile' already exists.\n" if -e $patfile;
open (SF2,">$patfile") || die "Can't open $patfile!  $!\n";

###########################################################################
# AutoFlush FileHandles:
###########################################################################
autoflush SF1 1;                # Immediate writes
autoflush SF2 1;                # Immediate writes
autoflush STDOUT 1;             # Immediate writes

=head2 ALGORITHM DESCRIPTION

=head3 Ramp Waveform Parameters

The B<ramp.pl> script will use the FREQ parameter to calculate
```

```
the following parameters:

=over 4

=item * Ramp Waveform Period (C<$reqPeriod>).

=item * Number of samples in the ramp waveform file (C<$numPoints>).

=item * Number of samples in the positive incline of the ramp waveform (C<$posPoints>).

=item * Number of samples in the negative decline of the ramp waveform (C<$negPoints>).

=item * Sample spacing for the positive incline of the ramp waveform (C<$posInc>).

=item * Sample spacing for the negative decline of the ramp waveform (C<$negInc>).

=back

In addition, the B<ramp.pl> script will define the following
parameters:

=over 4

=item * Sample frequency (C<$fs>).

=item * Sample period (C<$sampPeriod>).

=item * High-speed DAC resolution (C<$dacres>).

=item * Maximum signed 2's complement decimal value (C<$res>).

=back

=cut

#*****************************************************************
# Ramp Parameters:
#*****************************************************************
my ($res) = 15;
my ($dacres) = 16;
my ($fs) = 500e6;
$freq += 0;
my ($sampPeriod) = 1/$fs;
my ($reqPeriod) = 1/$freq;
my ($numPoints) = $reqPeriod/$sampPeriod;
my ($posPoints) = ceil($numPoints/2);
my ($negPoints) = ceil($numPoints/2);
```

```
my␣($posInc)␣=␣((2**$res)␣−␣(−(2**$res)))/$posPoints;
my␣($negInc)␣=␣((2**$res)␣−␣(−(2**$res)))/$negPoints;

#␣Print␣Parameters:
select(STDOUT);
printf("Sample␣Period:␣␣␣␣␣%␣.4e\n",␣$sampPeriod);
printf("Request␣Period:␣␣␣␣%␣.4e\n",␣$reqPeriod);
printf("Positive␣Ramp:␣␣␣␣␣%␣.4f\n",␣$posPoints);
printf("Negative␣Ramp:␣␣␣␣␣%␣.4f\n",␣$negPoints);
printf("Number␣Points:␣␣␣␣␣%␣.4f\n",␣$numPoints);
printf("Positive␣Inc:␣␣␣␣␣␣%␣.4f\n",␣$posInc);
printf("Negative␣Inc:␣␣␣␣␣␣%␣.4f\n",␣$negInc);

=head3␣File␣Headers

The␣B<ramp.pl>␣script␣will␣then␣use␣the␣calculated␣variables
to␣create␣the␣file␣headers␣for␣both␣the␣Matlab␣file␣and␣the
waveform␣file.␣The␣header␣of␣the␣waveform␣file␣requires␣additional
parameters␣to␣be␣calculated:

=over␣4

=item␣*␣Waveform␣length␣in␣bits␣(C<$numBits>).

=item␣*␣Waveform␣Stop␣Address␣(C<$stopAddr>).

=back

The␣waveform␣length␣is␣calculated␣using␣the␣high−speed␣DAC␣resolution
and␣the␣number␣of␣sample␣points␣required␣for␣the␣ramp␣waveform.

=cut

#****************************************************************
#␣Write␣Matlab␣Header:
#****************************************************************
my␣($matHead)␣=␣<<"MATHEAD";
%****************************************************************
%
%␣VCL␣Confidential␣Copyright␣ ␣2011␣UC␣Davis,␣ECE␣Department
%
%****************************************************************
%
%␣Module:␣␣␣␣␣␣␣␣$file
%␣Created␣on:␣␣␣␣␣␣␣␣Sun␣Jan␣01␣16:27:11␣2011
%\%\␣Executed␣by:␣␣␣␣␣␣␣␣jwwebb
%
%\%\␣board␣name:␣␣␣␣␣␣␣␣MSEE␣Thesis␣Measurement␣Board
```

```
%\%\␣board␣number:␣␣␣␣␣␣p342
%\%\␣board␣rev:␣␣␣␣␣001
%
%*************************************************************
%
%␣Column␣Headers␣for␣measurement
%␣␣␣\$1␣=␣time;␣Time␣(sec)
%␣␣␣\$2␣=␣volt;␣Voltage␣(V)
%
clear;clc;close␣all;
PrintOnEps␣=␣1;

MATHEAD

select(SF1);
printf("$matHead");
select(STDOUT);

#*************************************************************
#␣Write␣Pattern␣Header:
#*************************************************************
my␣($numBits)␣=␣$numPoints*$dacres;
my␣($mod128)␣=␣$numBits␣%␣128;
my␣($patLen)␣=␣($mod128␣>␣0)␣?␣$mod128*$numBits␣:␣$numBits;
my␣($stopAddr)␣=␣($patLen/128)␣-␣1;
my␣($stopAddrH)␣=␣dec2hex($stopAddr);
my␣($dispFs)␣=␣Suffix($fs,␣"Hz",␣1);
my␣($dispFc)␣=␣Suffix($freq,␣"Hz",␣1);
my␣($patHead)␣=␣<<"PATHEAD";
##############################################################
#patternname=$patfile
#patterntype=sram
#patternlength=$patLen
#readstartaddressa=0x0
#readstopaddressa=0x$stopAddrH
#readstartaddressb=0
#readstopaddressb=0
#triggerword=0
#patternstatistics=
#density=0.750
#bitshift=no
#bitshiftindex=0
#crc=???
#description=Ramp␣Waveform:␣Fc␣=␣$dispFc,␣Fs␣=␣$dispFs
##############################################################
#begin
PATHEAD
```

```
select(SF2);
printf("$patHead");
select(STDOUT);

=head3␣Positive␣and␣Negative␣Ramp␣Generation

Using␣the␣C<$posInc>␣and␣C<$negInc>␣parameters␣previously␣calculated,␣the
B<ramp.pl>␣script␣will␣generate␣the␣positive␣incline␣and␣negative␣decline␣of
the␣ramp␣waveform␣using␣the␣following␣steps:

=over␣4

=item␣1.␣Starting␣at␣the␣minimum␣value␣of␣the␣high-speed␣DAC,␣increment␣in␣C<$posInc>␣
    steps␣until␣the␣maximum␣high-speed␣DAC␣value␣is␣reached.

=item␣2.␣Starting␣at␣the␣maximum␣value␣of␣the␣high-speed␣DAC␣minus␣C<$negInc>,␣decrement␣
    in␣C<$negInc>␣steps␣until␣the␣minimum␣high-speed␣DAC␣value␣plus␣C<$negInc>␣is␣reached.

=back

=cut

#******************************************************************
#␣Generate␣Positive␣Ramp:
#******************************************************************
my␣($pidx)␣=␣1;
for␣(my␣$p␣=␣-(2**$res);␣$p␣<=␣(2**$res);␣$p␣+=␣$posInc)␣{
␣␣␣␣select(STDOUT);
␣␣␣␣printf("I:␣%d,␣P:␣%.4f\n",␣$pidx,␣$p)␣if␣$debug;
␣␣␣␣select(SF1);
␣␣␣␣printf("ramp(%d)␣=␣%.4f;\n",␣$pidx,␣$p/(2**$res));
␣␣␣␣select(SF2);
␣␣␣␣printf("%.15f\n",␣$p/(2**$res));
␣␣␣␣$pidx++;
}

#******************************************************************
#␣Generate␣Negative␣Ramp:
#******************************************************************
my␣($nidx)␣=␣$pidx;
my␣($j)=0;
for␣(my␣$n␣=␣(2**$res);␣$n␣>=␣(-(2**$res)␣+␣$negInc);␣$n␣-=␣$negInc)␣{
␣␣␣␣if␣($j␣>␣0)␣{␣#␣Skip␣maximum␣high-speed␣DAC␣value.
␣␣␣␣␣␣␣␣select(STDOUT);
␣␣␣␣␣␣␣␣printf("I:␣%d,␣N:␣%.4f\n",␣$nidx,␣$n)␣if␣$debug;
␣␣␣␣␣␣␣␣select(SF1);
␣␣␣␣␣␣␣␣printf("ramp(%d)␣=␣%.4f;\n",␣$nidx,␣$n/(2**$res));
␣␣␣␣␣␣␣␣select(SF2);
␣␣␣␣␣␣␣␣printf("%.15f\n",␣$n/(2**$res));
␣␣␣␣␣␣␣␣$nidx++;
␣␣␣␣}
␣␣␣␣$j++;
}

=head3␣File␣Footers

The␣B<ramp.pl>␣script␣will␣create␣a␣file␣header␣for␣both␣the␣Matlab␣file
and␣the␣waveform␣file.␣The␣Matlab␣file␣will␣contain␣code␣to
plot␣the␣generated␣ramp␣waveform.␣The␣waveform␣file␣will
contain␣the␣tag␣"#end",␣which␣indicates␣to␣the␣sig2hex.pl
script␣that␣the␣end␣of␣the␣data␣payload␣has␣been␣reached.

=cut

#***************************************************************
#␣Write␣Matlab␣Footer:
#***************************************************************
my␣($matFoot)␣=␣<<"MATFOOT";



%␣Plot␣Waveforms:␣Time␣(us)␣vs␣Voltage␣(V)
figure(1);
set(gca,'FontSize',14);
plot(ramp,'-r','LineWidth',2);
xlabel('Time (us)','fontsize',14);
ylabel('Voltage (mV)','fontsize',14);
grid␣on;
axis([0␣500␣-1.1␣1.1]);
if␣PrintOnEps
␣␣␣␣png_file␣=␣sprintf('meas_sig_src_ramp_td.png');
␣␣␣␣print('-dpng',␣png_file);
␣␣␣␣eps_file␣=␣sprintf('meas_sig_src_ramp_td.eps');
␣␣␣␣print('-depsc',␣eps_file);
end

MATFOOT

select(SF1);
printf("$matFoot\n");
select(STDOUT);

#***************************************************************
#␣Write␣Pattern␣Footer:
#***************************************************************
select(SF2);
```

```
printf("#end\n");
select(STDOUT);

exit;

=head1␣SUBROUTINES

=cut

#************************************************************
#␣Sub−routines
#************************************************************

sub␣dienice␣{
␣␣␣␣my($errmsg)␣=␣@_;
␣␣␣␣print"$errmsg\n";
␣␣␣␣exit;
}

sub␣print_usage␣{
␣␣␣␣my␣($usage);
␣␣␣␣$usage␣=␣"\nUsage:␣$0␣[−h]␣[−v]␣[−f␣<FILE>]␣[−s␣<FREQ>]\n";
␣␣␣␣$usage␣.=␣"\n";
␣␣␣␣$usage␣.=␣"\t−h\t\tPrint␣Help.\n";
␣␣␣␣$usage␣.=␣"\t−v\t\tVerbose:␣Print␣Debug␣Information.\n";
␣␣␣␣$usage␣.=␣"\t−f␣<FILE>\tNew␣Matlab␣filename.\n";
␣␣␣␣$usage␣.=␣"\t−s␣<FREQ>\tRequested␣Frequency.\n";
␣␣␣␣$usage␣.=␣"\n";
␣␣␣␣$usage␣.=␣"\tExample:\n";
␣␣␣␣$usage␣.=␣"\t\t$0␣−v␣−f␣sample.m␣−s␣1e6␣\n";
␣␣␣␣$usage␣.=␣"\n";
␣␣␣␣print($usage);
␣␣␣␣return;
}

sub␣dec2hex($)␣{

=head2␣Decimal␣to␣Hexadecimal␣Conversion

The␣B<dec2hex>␣sub−routine␣will␣convert␣a␣decimal␣value␣into␣its
hexadecimal␣equivalent.

=cut

␣␣␣␣##########################################################################
␣␣␣␣#␣Decimal␣to␣Hexadecimal␣Conversion:
␣␣␣␣#
␣␣␣␣#␣␣The␣sub−routine␣dec2hex()␣will␣convert␣a␣decimal␣value␣into␣its
```

```
␣␣␣␣#␣␣hexadecimal␣equivalent.
␣␣␣␣#
␣␣␣␣#␣␣Usage:␣$hout␣=␣dec2hex($din);
␣␣␣␣#
␣␣␣␣##########################################################################
␣␣␣␣my(␣$dec␣)␣=␣shift;
␣␣␣␣return␣sprintf(”%x”,␣$dec␣);
}

sub␣testhex␣{

=head2␣Test␣Hexadecimal␣Values

The␣B<testhex>␣sub−routine␣will␣receive␣a␣single
hexadecimal␣value.␣If␣the␣hexadecimal␣value␣is␣less
than␣4␣digits,␣then␣B<testhex>␣will␣pre−pend␣the
appropriate␣number␣of␣zeros␣in␣order␣to␣provide␣a
complete␣16−bit␣hexadecimal␣value.␣An␣error␣flag␣is
also␣provided␣to␣determine␣if␣the␣hex␣value␣is␣more
than␣4␣digits.

=cut

␣␣␣␣##########################################################################
␣␣␣␣#␣Test␣Hexadecimal␣Values:
␣␣␣␣#
␣␣␣␣#␣␣The␣sub−routine␣testhex()␣will␣receive␣a␣single
␣␣␣␣#␣␣hexadecimal␣value.␣If␣the␣hexadecimal␣value␣is␣less
␣␣␣␣#␣␣than␣4␣digits,␣then␣testhex()␣will␣prepend␣the
␣␣␣␣#␣␣appropriate␣number␣of␣zeros␣in␣order␣to␣provide␣a
␣␣␣␣#␣␣complete␣16−bit␣hexadecimal␣value.␣An␣error␣flag␣is
␣␣␣␣#␣␣also␣provided␣to␣determine␣if␣the␣hex␣value␣is␣more
␣␣␣␣#␣␣than␣4␣digits.
␣␣␣␣#
␣␣␣␣#␣␣␣@hexOut␣=␣(orig,␣hex4,␣len,␣hflag);
␣␣␣␣#
␣␣␣␣#␣␣Usage:␣my␣(@hexOut)␣=␣testhex($hexIn,$debug);
␣␣␣␣#
␣␣␣␣##########################################################################
␣␣␣␣my␣($hexIn)␣=␣shift;
␣␣␣␣my␣($debug)␣=␣shift;
␣␣␣␣my␣(@hexInA)␣=␣split(//,$hexIn);
␣␣␣␣my␣($hexInALen)␣=␣scalar(@hexInA);
␣␣␣␣print(”*␣Hex␣In:␣$hexIn\n”)␣if␣($debug);
␣␣␣␣print(”*␣Hex␣Length␣In:␣$hexInALen\n”)␣if␣($debug);
␣␣␣␣my␣(@hexOut);
␣␣␣␣my␣($i);
␣␣␣␣my␣($hexDiff)␣=␣4−$hexInALen;
```

```
    my ($hFlag) = 0;

    if ($hexDiff == 0) {
        print("* hex length is 4\n") if ($debug);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
    } elsif ($hexDiff > 0) {
        for ($i=0; $i<$hexDiff; $i++) {
            unshift(@hexInA,0);
        }
        push(@hexOut, $hexIn);
        push(@hexOut, join("", @hexInA));
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
        print("* hex length is $hexInALen, add $hexDiff zeros to pad to 4\n") if ($debug);
        print("* New Hex: $hexOut[1]\n") if ($debug);
    } else {
        $hFlag = 1;
        push(@hexOut, $hexIn);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
        print("* hex length is: $hexInALen\n") if $debug;
    }

    print Dumper(@hexOut) if $debug;

    return(@hexOut);
}
```

Appendix E

ADC Waveform Capture Conversion Perl Script

This chapter describes the wave2mat ADC waveform capture conversion Perl script, which is used to convert ADC data captured by the Data Path FPGA into a data file suitable for analysis in Matlab.

E.1 NAME

wave2mat.pl - ADC Capture Signed 2's Compliment Hexadecimal-to-Decimal Converter

E.2 SYNOPSIS

```
wave2mat.pl [-h] [-v] [-f <FILE>]

Help Options:
 -h          Print Help.
 -v          Verbose: Print Debug Information.
 -f <FILE>   Input ADC capture filename.

 Example:
     ./wave2mat.pl -v -f sample.dat
```

E.3 OPTIONS

-h

Show the brief help information.

-v

Show debug information.

-f FILE

ADC capture input filename containing ADC data in signed 2's complement hexadecimal format.

E.4 DESCRIPTION

wave2mat.pl is used to convert ADC capture files created by the Measurement Board and captured from the high-speed ADC into a data file suitable for analysis in Matlab. **wave2mat.pl** will read in an ADC capture file with 8 signed 2's complement hexadecimal values and a single over-range value per line. The values are comma separated. It will sign-extend and convert the signed 2's complement hexadecimal data to decimal. The decimal data will then be quantized based on the ADC resolution as follows:

- `y[n] = (x[n] / 2**11) * (2.2V / 2)`

The data will then be written out to a file for analysis in Matlab.

E.5 SUBROUTINES

Get Waveform File

The **getADCFile** sub-routine will open the input ADC capture file and store the data into a hash as an array.

Parse ADC Capture File

The **parseADCFile** sub-routine will extract the ADC capture data from the file array into its own array and store both the array and the number of ADC samples in a hash.

Convert 16-bit Hexadecimal to Signed 2's Complement Decimal

The **convHex2Dec** sub-routine will sign-extend, convert the hexadecimal value to a signed 2's complement decimal value, and quantize the data. It will then write the data to a data file suitable for analysis in Matlab.

ALGORITHM DESCRIPTION

1. Define the full scale ADC input voltage.

   a. The ADC full scale input voltage is 2.2V.

2. Define the maximum value of a hexadecimal sample using the RES value.

   a. `$maxValue = 2**$RES-1`

   b. The default RES value for the measurement board ADC is 12.

3. Grab a signed 2's complement hexadecimal value from the data array.

4. Sign-extend the signed 2's complement hexadecimal value to 16 bits.

5. Convert the sign-extended hexadecimal data to decimal.

   a. `$dec = signed_hex2dec($btc_hex)`

6. Quantize signed 2's complement decimal value to RES bits.

   a. `$decQuant = ($dec / $maxValue) * ($fs / 2)`

7. Store the signed 2's complement decimal value in an array in the order provided by the input ADC capture file.

8. Write the signed 2's complement decimal value to an output data file for analysis in Matlab.

Test Hexadecimal Values

The **testhex** sub-routine will receive a single hexadecimal value. If the hexadecimal value is less than 4 digits, then **testhex** will pre-pend the appropriate number of zeros in order to provide a complete 16-bit hexadecimal value. An error flag is also provided to determine if the hex value is more than 4 digits.

Signed Extension

The **signextend** sub-routine will sign-extend a hexadecimal value.

Signed Hexadecimal to Decimal Conversion

The **signed_hex2dec** sub-routine will convert a signed hexadecimal value into its decimal equivalent.

Hexadecimal to Decimal Conversion

The **hex2bin** sub-routine will convert a hexadecimal value into its decimal equivalent.

Decimal to Hexadecimal Conversion

The **dec2hex** sub-routine will convert a decimal value into its hexadecimal equivalent.

Hexadecimal to Binary Conversion

The **hex2bin** sub-routine will convert a hexadecimal value into its binary equivalent.

Binary to Hexadecimal Conversion

The **bin2hex** sub-routine will convert a binary value into its hexadecimal equivalent.

E.6 CODE

Listing E.1: ADC Waveform Capture Conversion Perl Script

```
#!/usr/bin/env perl
# vim:ts=4:sw=4:expandtab:cindent

#*****************************************************************
#
# wave2mat.pl module
#
#*****************************************************************
#
# VCL Confidential Copyright   2011 UC Davis, ECE Department
#
#*****************************************************************
#
# created on:    01/24/2011
# created by:    jwwebb
# last edit on: $DateTime: $
# last edit by: $Author: $
# revision:          $Revision: $
# comments:          Generated
#
#*****************************************************************
# Revision List:
#
#                  1.0      01/24/2011       Initial release
```

```
#
#        Please report bugs, errors, etc.
#***************************************************************
# ADC Capture Data to Matlab Script
#
#  This utility is intended to convert an ADC capture file
#  containing 8 ADC samples and an over-range indicator per line
#  into a Matlab array for further post-processing.
#
#  Usage Information:
#
#          Usage: ./wave2mat.pl [-h] [-v] [-f <FILE>]
#
#                  -h                Print Help.
#                  -v                Verbose: Print Debug Information.
#                  -f <FILE>         Input LOG File.
#
#                  Example:
#                             ./wave2mat.pl -v -f LOGFILE.txt
#
#***************************************************************


#***************************************************************
# CPAN Modules
#***************************************************************
use strict;
use Getopt::Std;
use Fcntl;                              # File control (lock, etc...)
use SDBM_File;                          # Simple database
use Carp;                               # Warnings/Errors for modules
use File::Basename;
use File::Path;
use FileHandle;
use POSIX;
use Data::Dumper;

#***************************************************************
# Constants and Variables:
#***************************************************************
my (%opts)=();
my ($file);
my ($debug);
my (%adcH, $adc_rH);

#***************************************************************
# Retrieve command line argument
#***************************************************************
```

```
getopts('hv:f:',\%opts);
my $optslen = scalar( keys %opts );
print("Number of Options on Command-Line: $optslen\n") if $opts{v};

#check for valid combination command-line arguments
if ($opts{h} || ($optslen eq "0")) {
    print_usage();
    exit;
}

# parse command-line arguments
$file = $opts{f};
$debug = $opts{v};

################################################################################
# Stuff input options into a Hash:
################################################################################
$adcH{ 'file' }        = $file;
$adcH{ 'debug' }       = $debug;

###########################################################################
# AutoFlush FileHandles:
###########################################################################
autoflush STDOUT 1;                          # Immediate writes
select(STDOUT);

################################################################################
# Convert ADC Sample Data from Log File to Matlab File:
################################################################################
if ($file) {
    ###########################################################################
    # Get Block RAM Patter File:
    ###########################################################################
    $adc_rH = getADCFile(\%adcH);
    ###########################################################################
    # Parse Block RAM Pattern File:
    ###########################################################################
    $adc_rH = parseADCFile($adc_rH);
    ###########################################################################
    # Convert from Hex 2 Dec:
    ###########################################################################
    $adc_rH = convHex2Dec($adc_rH);
}

exit;
=pod

=head1 NAME
```

```
B<wave2mat.pl> - ADC Capture Signed 2's␣Compliment␣Hexadecimal-to-Decimal␣Converter

=head1␣SYNOPSIS

␣␣wave2mat.pl␣[-h]␣[-v]␣[-f␣<FILE>]

␣␣Help␣Options:
␣␣␣-h␣␣␣␣␣␣␣␣␣␣Print␣Help.
␣␣␣-v␣␣␣␣␣␣␣␣␣␣Verbose:␣Print␣Debug␣Information.
␣␣␣-f␣<FILE>␣␣␣Input␣ADC␣capture␣filename.

␣␣␣Example:
␣␣␣␣␣␣␣./wave2mat.pl␣-v␣-f␣sample.dat

=head1␣OPTIONS

=over␣8

=item␣B<-h>

Show␣the␣brief␣help␣information.

=item␣B<-v>

Show␣debug␣information.

=item␣B<-f␣FILE>

ADC␣capture␣input␣filename␣containing␣ADC␣data␣in␣signed␣2's complement hexadecimal format
    .

=back

=head1 DESCRIPTION

B<wave2mat.pl> is used to convert ADC capture files created by the Measurement Board and
captured from the high-speed ADC into a data file suitable for analysis in Matlab.
B<wave2mat.pl> will read in an ADC capture file with 8 signed 2's␣complement␣hexadecimal␣
    values␣and
a␣single␣over-range␣value␣per␣line.␣The␣values␣are␣comma␣separated.␣It␣will␣sign-extend␣
    and
convert␣the␣signed␣2's complement hexadecimal data to decimal. The decimal data will
then be quantized based on the ADC resolution as follows:

=over 4

=item * C<y[n] = (x[n] / 2**11) * (2.2V / 2)>
```

```
=back

The data will then be written out to a file for analysis in Matlab.

=head1 SUBROUTINES

=cut

#*************************************************************
# Sub-routines
#*************************************************************

sub dienice {
        my($errmsg) = @_;
        print"$errmsg\n";
        exit;
}

sub print_usage {
        my ($usage);
        $usage = "\nUsage: $0 [-h] [-v] [-f <FILE>]\n";
        $usage .= "\n";
        $usage .= "\t-h\t\tPrint Help.\n";
        $usage .= "\t-v\t\tVerbose: Print Debug Information.\n";
        $usage .= "\t-f <FILE>\tInput LOG filename.\n";
        $usage .= "\n";
        $usage .= "\tExample:\n";
        $usage .= "\t\t$0 -v -f logfile.txt\n";
        $usage .= "\n";
        print($usage);
        return;
}

sub getADCFile {

=head2 Get Waveform File

The B<getADCFile> sub-routine will open the input ADC capture file and store
the data into a hash as an array.

=cut

    ##########################################################################
    # Get ADC Sample File:
    #
    #  The sub-routine getADCFile() will open the ADC Sample file
    #  and read its contents into an array. It will also determine
    #  the file length. The following parameters are created
    #
    #    * fileData:              @dataA
    #    * fileLen:               scalar(@dataA)
    #
    #  Usage: $adc_rH = getADCFile(\%adcH);
    #
    ###########################################################################
    my ($adc_rH)     = shift;             # Read in user's variable.
    my (%adcH)       = %{ $adc_rH };          # De-reference hash.
    my ($debug)      = $adcH{'debug'};    # Print out Debug Info.

    ###########################################################################
    # Open the ADC Sample file, and read the results into an array for
    # manipulating the data array. Strip new lines and carriage returns
    # from remove string array, and initialize for loop variables. Close file
    # when done.
    ###########################################################################
    my (@samples_AoH);
    my (@tmp);
    open(inF, "<", $adcH{ 'file' }) or dienice ("$adcH{ 'file' } open failed");
    while (<inF>) {
        chomp;
            if (!($_ =~ /#/)) {
            @tmp = split(/,/, $_);
                $tmp[2] =~ s/[\r|\n]//;
            push(@samples_AoH, {OVR     => $tmp[0],
                                             Slice7 => $tmp[1],
                                             Slice6 => $tmp[2],
                                             Slice5 => $tmp[3],
                                             Slice4 => $tmp[4],
                                             Slice3 => $tmp[5],
                                             Slice2 => $tmp[6],
                                             Slice1 => $tmp[7],
                                             Slice0 => $tmp[8]
                                               } );
            }
    }
    close(inF);

    print Dumper(@samples_AoH) if $debug;

    ###########################################################################
    # Push signals into Hash.
    ###########################################################################
    $adcH{'samples_AoH'} = \@samples_AoH;

    ###########################################################################
```

```
    # Return data to user
    ############################################################################
    return \%adcH;
}

sub parseADCFile {

=head2 Parse ADC Capture File

The B<parseADCFile> sub-routine will extract the ADC capture data from the
file array into its own array and store both the array and the number of
ADC samples in a hash.

=cut

    ############################################################################
    # Parse ADC Capture File
    #
    #  The sub-routine parseADCFile() will parse the input ADC Capture File
    #  and retrieve the following information:
    #
    #                              #begin
    #                              #end
    #
    #  This sub-routine will also extract the actual pattern data into an
    #  array for converting from hexadecimal to decimal.
    #
    #  Usage: $adc_rH = parseADCFile(\%adcH);
    #
    ############################################################################
    my ($adc_rH)        = shift;                     # Read in user's variable.
    my (%adcH)          = %{ $adc_rH };          # De-reference hash.
    my (@samples_AoH)   = @{ $adcH{ 'samples_AoH' } };
    my ($debug)         = $adcH{'debug'};        # Print out Debug Info.

    ############################################################################
    # Create an array with a single hexadecimal sample per element:
    ############################################################################
    my ($i) = 0;
    my (@samples_hexA);
    my ($tmp7);
    my ($tmp6);
    my ($tmp5);
    my ($tmp4);
    my ($tmp3);
    my ($tmp2);
    my ($tmp1);
    my ($tmp0);
    for $i ( 0 .. $#samples_AoH ) {
            # Grab 8 ADC Samples:
            $tmp7 = $samples_AoH[$i]{Slice7};
        $tmp6 = $samples_AoH[$i]{Slice6};
        $tmp5 = $samples_AoH[$i]{Slice5};
        $tmp4 = $samples_AoH[$i]{Slice4};
        $tmp3 = $samples_AoH[$i]{Slice3};
        $tmp2 = $samples_AoH[$i]{Slice2};
        $tmp1 = $samples_AoH[$i]{Slice1};
        $tmp0 = $samples_AoH[$i]{Slice0};

            push(@samples_hexA, $tmp7);
            push(@samples_hexA, $tmp6);
            push(@samples_hexA, $tmp5);
            push(@samples_hexA, $tmp4);
            push(@samples_hexA, $tmp3);
            push(@samples_hexA, $tmp2);
            push(@samples_hexA, $tmp1);
            push(@samples_hexA, $tmp0);
    }

    $adcH{'samples_hexA'} = \@samples_hexA;
    print Dumper(@samples_hexA) if $debug;

    ###########################################################################
    # Determine number of samples
    ###########################################################################
    $adcH{ 'NumberSamples' } = scalar(@{ $adcH{ 'samples_hexA' } });

    print("\n\n") if $debug;
    print("Total_number_of_lines:_$adcH{_'NumberSamples'_}\n") if $debug;
    print("\n\n") if $debug;

    ###########################################################################
    # Return data to user
    ###########################################################################
    return \%adcH;
}

sub convHex2Dec {

=head2 Convert 16-bit Hexadecimal to Signed 2's_Complement_Decimal

The_B<convHex2Dec>_sub-routine_will_sign-extend,_convert_the_hexadecimal
value_to_a_signed_2's complement decimal value, and quantize the data.
It will then write the data to a data file suitable for analysis in Matlab.

=head3 ALGORITHM DESCRIPTION
```

```
=over 4

=item 1. Define the full scale ADC input voltage.

=over 4

=item a. The ADC full scale input voltage is 2.2V.

=back

=item 2. Define the maximum value of a hexadecimal sample using the RES value.

=over 4

=item a. C<$maxValue = 2**$RES−1>

=item b. The default RES value for the measurment board ADC is 12.

=back

=item 3. Grab a signed 2's␣complement␣hexadecimal␣value␣from␣the␣data␣array.

=item␣4.␣Sign−extend␣the␣signed␣2's complement hexadecimal value to 16 bits.

=item 5. Convert the sign−extended hexadecimal data to decimal.

=over 4

=item a. C<$dec = signed_hex2dec($btc_hex)>

=back

=item 6. Quantize signed 2's␣complement␣decimal␣value␣to␣RES␣bits.

=over␣4

=item␣a.␣C<$decQuant␣=␣($dec␣/␣$maxValue)␣*␣($fs␣/␣2)>

=back

=item␣8.␣Store␣the␣signed␣2's complement decimal value in an array in the order provided
    by the input ADC capture file.

=item 9. Write the signed 2's␣complement␣decimal␣value␣to␣an␣output␣data␣file␣for␣analysis
    ␣in␣Matlab.

=back
```

```
=cut

    ##########################################################################
    # Convert Hexadecimal to Decimal:
    #
    #  The sub-routine convHex2Dec() will convert the hexadecimal data to
    #  decimal data.
    #
    #  Usage: $adc_rH = convHex2Dec($adcH);
    #
    ##########################################################################
    my ($adc_rH)        = shift;                    # Read in user's variable.
    my (%adcH)            = %{ $adc_rH };           # De-reference hash.
    my ($nsamps)              = $adcH{'NumberSamples'};
    my (@samples_hexA)    = @{ $adcH{ 'samples_hexA' } };
    my ($debug)           = $adcH{'debug'};             # Print out Debug Info.

    ##########################################################################
    # Convert Waveform Data from Signed 2's Complement Hexadecimal to Decimal
    ##########################################################################
    my ($fs) = 2.2; # 2.2V
    my ($res) = 12;
    my ($maxValue) = (2**($res-1)); #-1;
    print("Maximum Value: $maxValue\n") if $debug;

    my (@waveDecA);

    my ($j) = 0;
    for ($j=0; $j<$nsamps; $j++) {
        my $btc_hex = signextend($samples_hexA[$j]);
        my $dec = signed_hex2dec($btc_hex);
            my $decQuant = ($dec/$maxValue)*($fs/2);
            print("Hexadecimal: $samples_hexA[$j]; $btc_hex; Decimal: $dec, Decimal (
                Quantized): $decQuant\n") if $debug;
        my ($decOut) = $decQuant;
            push(@waveDecA, $decOut);

            print("\n\n") if $debug;
    }

    ##########################################################################
    # Write Waveform Data in Signed 2's Complement Decimal to Matlab File
    ##########################################################################
    my $newfile = $adcH{ 'file' };
    $newfile =~ s/\..*$//;
    $newfile .= "_dec.m";
    # check to make sure that the file doesn't exist.
    die "Oops! A file called '$newfile' already exists.\n" if -e $newfile;
    # Open Hex File:
    open(WF1, ">", $newfile);
    autoflush WF1 1;                              # Immediate writes
    select(WF1);

    my ($k) = 0;
    for ($k=0; $k<scalar(@waveDecA); $k++) {
            my ($index) = $k+1;
            printf("wave_p($index) = $waveDecA[$k];");
            printf("\n");
    }

        # Close the new waveform file:
    select(STDOUT);

    ##########################################################################
    # Return data to user
    ##########################################################################
    return \%adcH;
}

sub testhex {

=head2 Test Hexadecimal Values

The B<testhex> sub-routine will receive a single
hexadecimal value. If the hexadecimal value is less
than 4 digits, then B<testhex> will pre-pend the
appropriate number of zeros in order to provide a
complete 16-bit hexadecimal value. An error flag is
also provided to determine if the hex value is more
than 4 digits.

=cut

    ##########################################################################
    # Test Hexadecimal Values:
    #
    #  The sub-routine testhex() will receive a single
    #  hexadecimal value. If the hexadecimal value is less
    #  than 4 digits, then testhex() will prepend the
    #  appropriate number of zeros in order to provide a
    #  complete 16-bit hexadecimal value. An error flag is
    #  also provided to determine if the hex value is more
    #  than 4 digits.
    #
    #   @hexOut = (orig, hex4, len, hflag);
    #
    #  Usage: my (@hexOut) = testhex($hexIn,$debug);
    #
    ##########################################################################
    my ($hexIn) = shift;
    my ($debug) = shift;
    my (@hexInA) = split(//,$hexIn);
    my ($hexInALen) = scalar(@hexInA);
    print("Hex_In: $hexIn\n") if ($debug);
    print("Hex_Length_In: $hexInALen\n") if ($debug);
    my (@hexOut);
    my ($i);
    my ($hexDiff) = 4-$hexInALen;
    my ($hFlag) = 0;

    if ($hexDiff == 0) {
        print("hex_length_is_4\n") if ($debug);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
    } elsif ($hexDiff > 0) {
        for ($i=0; $i<$hexDiff; $i++) {
            unshift(@hexInA,0);
        }
        push(@hexOut, $hexIn);
        push(@hexOut, join("", @hexInA));
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
        print("hex_length_is_$hexInALen,_add_$hexDiff_zeros_to_pad_to_4\n") if ($debug);
        print("New_Hex: $hexOut[1]\n") if ($debug);
    } else {
        $hFlag = 1;
        push(@hexOut, $hexIn);
        push(@hexOut, $hexIn);
        push(@hexOut, $hexInALen);
        push(@hexOut, $hFlag);
        print("hex_length_is: $hexInALen\n") if $debug;
    }

    print Dumper(@hexOut) if $debug;

    return(@hexOut);
}

sub signextend {
```

```
=head2 Signed Extension

The B<signextend> sub-routine will sign-extend a hexadecimal value.

=cut

    ############################################################################
    # Signed Extension:
    #
    #  The sub-routine signextend() will sign-extend a hexadecimal value.
    #
    #  Usage: $hout = signextend($hin);
    #
    ############################################################################
    my ($hex) = shift;
    #--------------------------------------------------------------
    # Remove "0x" from hexadecimal input value:
    #--------------------------------------------------------------
    $hex =~ s/^0x//;
    #--------------------------------------------------------------
    # Split hex input value into multiple hex digits:
    #--------------------------------------------------------------
    my (@hexA) = split(//,$hex);
    #--------------------------------------------------------------
    # Convert each hex digit to binary:
    #--------------------------------------------------------------
    my (@tmpBinA);
    foreach my $i (@hexA) {
        push(@tmpBinA, split(//,hex2bin($i)));
    }
    #--------------------------------------------------------------
    # Check input value bit width and check MSB to determine
    # if sign-extension is a 0 or a 1.
    #--------------------------------------------------------------
    my (@tmpB);
    my ($size) = scalar(@tmpBinA);
    if ($size eq 12) {
        if ($tmpBinA[0] =~ m/0/) {
            push(@tmpB, 0);
            push(@tmpB, 0);
            push(@tmpB, 0);
            push(@tmpB, 0);
        } else {
            push(@tmpB, 1);
            push(@tmpB, 1);
            push(@tmpB, 1);
            push(@tmpB, 1);
        }
    }
    #---------------------------------------------------------------
    # Append binary input value to sign-extended MSB bits:
    #---------------------------------------------------------------
    push(@tmpB,@tmpBinA);
    #---------------------------------------------------------------
    # Join sign-extended binary bits:
    #---------------------------------------------------------------
    my ($binTmp) = join("",@tmpB);
    #---------------------------------------------------------------
    # Convert to Hex:
    #---------------------------------------------------------------
    my ($hexTmp) = bin2hex($binTmp);
    #---------------------------------------------------------------
    # Make sure value is padded to 16-bits:
    #---------------------------------------------------------------
    my (@hexOut) = testhex($hexTmp,0);
    my ($hexNew) = $hexOut[1];

    return ($hexNew);
}

sub signed_hex2dec {

=head2 Signed Hexadecimal to Decimal Conversion

The B<signed_hex2dec> sub-routine will convert a signed hexadecimal value into its
decimal equivalent.

=cut

    ###########################################################################
    # Signed Hexadecimal to Decimal Conversion:
    #
    #  The sub-routine signed_hex2dec() will convert a signed hexadecimal
    #  value into its decimal equivalent.
    #
    #  Usage: $dout = signed_hex2dec($hin);
    #
    ###########################################################################
    my ($hexIn) = shift;
    return unpack('s',pack 's', hex($hexIn));
}

sub hex2dec {

=head2 Hexadecimal to Decimal Conversion
```

```
The B<hex2bin> sub-routine will convert a hexadecimal value into its
decimal equivalent.

=cut

    ###########################################################################
    # Hexadecimal to Decimal Conversion:
    #
    #  The sub-routine hex2dec() will convert a hexadecimal value into its
    #  decimal equivalent.
    #
    #  Usage: $dout = hex2dec($hin);
    #
    ###########################################################################
    my $hex = shift;
    return hex($hex);
}

sub dec2hex($) {

=head2 Decimal to Hexadecimal Conversion

The B<dec2hex> sub-routine will convert a decimal value into its
hexadecimal equivalent.

=cut

    ###########################################################################
    # Decimal to Hexadecimal Conversion:
    #
    #  The sub-routine dec2hex() will convert a decimal value into its
    #  hexadecimal equivalent.
    #
    #  Usage: $hout = dec2hex($din);
    #
    ###########################################################################
    my( $dec ) = shift;
    return sprintf("%x", $dec );
}

sub hex2bin {

=head2 Hexadecimal to Binary Conversion

The B<hex2bin> sub-routine will convert a hexadecimal value into its
binary equivalent.

=cut
```

```
    ##########################################################################
    # Hexadecimal to Binary Conversion:
    #
    #  The sub-routine hex2bin() will convert a hexadecimal value into its
    #  binary equivalent.
    #
    #  Usage: $bout = hex2bin($hin);
    #
    ##########################################################################
    my $hex = shift;
    my $binary;
    my %h2b = (0 => "0000", 1 => "0001", 2 => "0010", 3 => "0011",
               4 => "0100", 5 => "0101", 6 => "0110", 7 => "0111",
               8 => "1000", 9 => "1001", a => "1010", b => "1011",
               c => "1100", d => "1101", e => "1110", f => "1111",
              );
    ($binary = $hex) =~ s/(.)/$h2b{lc $1}/g;
    return ($binary);
}

sub bin2hex {

=head2 Binary to Hexadecimal Conversion

The B<bin2hex> sub-routine will convert a binary value into its hexadecimal equivalent.

=cut

    ##########################################################################
    # Binary to Hexadecimal Conversion:
    #
    #  The sub-routine bin2hex() will convert a binary value into its
    #  hexadecimal equivalent.
    #
    #  Usage: $hout = bin2hex($bin);
    #
    ##########################################################################
    my $binary = shift;
    return sprintf("%X", oct("0b".$binary));
}
```

Appendix F

Signal Routing Groups

F.1 DDR SDRAM Routing Constraints

The MicroBlaze soft-core processor running on the Control FPGA of the measurement board uses a 512-Mbit, 125 MHz DDR SDRAM device for code storage and execution. The Control FPGA, which is a Xilinx Spartan-3A XC3S1400A, does not contain delay primitives that allow the timing of each I/O to be adjusted. Therefore the traces must be tightly matched in order for the memory to run at optimal speed. The DDR SDRAM control interface consists of the following signals:

- 1 differential clock
- 13 address bits
- 16 bidirectional data bits
- 2 bidirectional data strobe bits
- 1 clock enable bit
- 1 chip select bit
- 1 write enable bit
- 1 row-address strobe (RAS)
- 1 column-address strobe (CAS)
- 2 byte-address bits

- 2 data mask bits
- 1 feedback clock

The signals are routed as transmission lines. In the case of the unidirectional signals, the transmission lines are terminated using both series and parallel methods. The series terminations are located approximately halfway between the Control FPGA and the DDR SDRAM. The bidirectional signals have an additional parallel termination at the transmitter in order to minimize reflection during reads and writes. The use of series terminations effectively breaks the signals into two groups. The trace lengths of each signal group must be matched to within ± 70 mils or $\pm 10 - 15$ ps. The DDR SDRAM memory controller requires the feedback clock signal to have a length equal to the average length of the data strobe and differential clock. The equation used to calculate the trace length is shown in Equation F.1.

$$\begin{aligned} L_{CLKFB} \quad = \quad & \left(\frac{(DDR\_SDRAM\_LDQS + FPGA\_DDR\_SDRAM\_LDQS)}{2} \right) \\ & + \left(\frac{(DDR\_SDRAM\_UDQS + FPGA\_DDR\_SDRAM\_UDQS)}{2} \right) \\ & + \left(\frac{(FPGA\_DDR\_SDRAM\_CLK\_N + FPGA\_DDR\_SDRAM\_CLK\_P)}{2} \right) \end{aligned}$$

For the memory interface to operate correctly, the differential clock trace length must be equivalent to the average length of the pre- and post-series termination signal groups. The equation used to calculate the trace length is shown in Equation F.1.

$$L_{CLK\_LEN} = average(L_{DDR\_SDRAM1}) + average(L_{DDR\_SDRAM2}) \tag{F.1}$$

The signal groups and the associated length-matching tolerances are outlined in Table F.1.

Table F.1: DDR SDRAM Matched Length Sets

DDR SDRAM Matched Length Sets			
Matched Set	**Match From**	**Match To**	**Tolerance**
DDR_SDRAM1	FPGA_DDR_SDRAM_ADDR[12:0] FPGA_DDR_SDRAM_BA[1:0] FPGA_DDR_SDRAM_CASN FPGA_DDR_SDRAM_CKE FPGA_DDR_SDRAM_CSN FPGA_DDR_SDRAM_RASN FPGA_DDR_SDRAM_WEN	FPGA_DDR_SDRAM_CLK_N FPGA_DDR_SDRAM_CLK_P	± 25 ps
	FPGA_DDR_SDRAM_DATA[15:0] FPGA_DDR_SDRAM_LDM FPGA_DDR_SDRAM_UDM	FPGA_DDR_SDRAM_LDQS FPGA_DDR_SDRAM_UDQS	± 25 ps
	FPGA_DDR_SDRAM_LDQS FPGA_DDR_SDRAM_UDQS	FPGA_DDR_SDRAM_CLK_N FPGA_DDR_SDRAM_CLK_P	± 100 ps
DDR_SDRAM2	DDR_SDRAM_ADDR[12:0] DDR_SDRAM_BA[1:0] DDR_SDRAM_CASN DDR_SDRAM_CKE DDR_SDRAM_CSN DDR_SDRAM_RASN DDR_SDRAM_WEN	FPGA_DDR_SDRAM_CLK_N FPGA_DDR_SDRAM_CLK_P	± 25 ps
	DDR_SDRAM_DATA[15:0] DDR_SDRAM_LDM DDR_SDRAM_UDM	DDR_SDRAM_LDQS DDR_SDRAM_UDQS	± 25 ps
	DDR_SDRAM_LDQS DDR_SDRAM_UDQS	FPGA_DDR_SDRAM_CLK_N FPGA_DDR_SDRAM_CLK_P	± 100 ps
DDR_SDRAM3	FPGA_DDR_SDRAM_CLKFB		± 25 ps

F.2 DDR2 SDRAM SODIMM Routing Constraints

The DDR2 SDRAM SODIMM interface can be routed using a shortest length method. The signals in this group are shown in Table F.2.

Table F.2: DDR2 SDRAM SODIMM Matched Length Sets

DDR2 SDRAM SODIMM Matched Length Sets		
Matched Set	**Signal Name**	**Tolerance**
DDR2_SDRAM1	FPGA_DDR2_SDRAM_A[13:0]	NA
	FPGA_DDR2_SDRAM_CK_P0	
	FPGA_DDR2_SDRAM_CK_N0	
	FPGA_DDR2_SDRAM_CK_P1	
	FPGA_DDR2_SDRAM_CK_N1	
	FPGA_DDR2_SDRAM_BA[2:0]	
	FPGA_DDR2_SDRAM_WDATA[35:0]	
	FPGA_DDR2_SDRAM_CASN	
	FPGA_DDR2_SDRAM_RASN	
	FPGA_DDR2_SDRAM_CKE[1:0]	
	FPGA_DDR2_SDRAM_SN[1:0]	
	FPGA_DDR2_SDRAM_ODT[1:0]	
	FPGA_DDR2_SDRAM_WEN	
DDR2_SDRAM2	FPGA_DDR2_SDRAM_DM0	NA
	FPGA_DDR2_SDRAM_DQS0	
	FPGA_DDR2_SDRAM_DQSN_NC0	
	FPGA_DDR2_SDRAM_DQS[7:0]	
DDR2_SDRAM3	FPGA_DDR2_SDRAM_DM1	NA
	FPGA_DDR2_SDRAM_DQS1	
	FPGA_DDR2_SDRAM_DQSN_NC1	
	FPGA_DDR2_SDRAM_DQS[15:8]	
DDR2_SDRAM4	FPGA_DDR2_SDRAM_DM2	NA
	FPGA_DDR2_SDRAM_DQS2	
	FPGA_DDR2_SDRAM_DQSN_NC2	
	FPGA_DDR2_SDRAM_DQS[23:16]	
DDR2_SDRAM5	FPGA_DDR2_SDRAM_DM3	NA
	FPGA_DDR2_SDRAM_DQS3	
	FPGA_DDR2_SDRAM_DQSN_NC3	
	FPGA_DDR2_SDRAM_DQS[31:24]	

Continued on Next Page...

DDR2 SDRAM SODIMM Matched Length Sets		
Matched Set	**Signal Name**	**Tolerance**
DDR2_SDRAM6	FPGA_DDR2_SDRAM_DM4 FPGA_DDR2_SDRAM_DQS4 FPGA_DDR2_SDRAM_DQSN_NC4 FPGA_DDR2_SDRAM_DQS[39:32]	NA
DDR2_SDRAM7	FPGA_DDR2_SDRAM_DM5 FPGA_DDR2_SDRAM_DQS5 FPGA_DDR2_SDRAM_DQSN_NC5 FPGA_DDR2_SDRAM_DQS[47:40]	NA
DDR2_SDRAM8	FPGA_DDR2_SDRAM_DM6 FPGA_DDR2_SDRAM_DQS6 FPGA_DDR2_SDRAM_DQSN_NC6 FPGA_DDR2_SDRAM_DQS[55:48]	NA
DDR2_SDRAM9	FPGA_DDR2_SDRAM_DM7 FPGA_DDR2_SDRAM_DQS7 FPGA_DDR2_SDRAM_DQSN_NC7 FPGA_DDR2_SDRAM_DQS[63:56]	NA

F.3 QDR-II SRAM Routing Constraints

The trace lengths of the QDR-II memory device input signals (QDR_K, QDR_K_n, QDR_C, QDR_C_n, QDR_WR_n, QDR_RD_n, QDR_SA, QDR_BW_n, and QDR_D) must be well matched within ± 20ps to present the control, address, and data lines to the memory device with adequate setup and hold margins. The implementation of the physical interface ensures these signals are center aligned to the QDR_K and QDR_K_n clock edges when leaving the FPGA device outputs. The board traces must ensure that this relationship continues to the memory device inputs.

Similarly, the QDR-II memory device output signals (QDR_Q, QDR_CQ, and QDR_CQ_n) must have well-matched trace lengths within ±20 ps for the signals to all arrive edge aligned at the inputs to the Virtex-5 SX50T FPGA. This trace length matching is critical to the implementation of the direct-clocking Read data capture methodology. Any reasonable board design tool can match these traces within an acceptable tolerance with little effort.

Table F.3: QDR-II SRAM Matched Length Sets

QDR-II SRAM Matched Length Sets		
Matched Set	**Signal Name**	**Tolerance**
SRAM1	FPGA_SRAM_ADDR[20:0] FPGA_SRAM_K_CLK_P FPGA_SRAM_K_CLK_N FPGA_SRAM_C_CLK_P FPGA_SRAM_C_CLK_N FPGA_SRAM_BWN[3:0] FPGA_SRAM_WDATA[35:0] FPGA_SRAM_RDN FPGA_SRAM_WRN FPGA_SRAM_DLL_OFFN	NA
SRAM2	FPGA_SRAM_CQ_CLK_N FPGA_SRAM_RDATA[35:18]	NA
SRAM3	FPGA_SRAM_CQ_CLK_P FPGA_SRAM_RDATA[17:0]	NA

Parameter Definitions:

- w: trace width
- s: trace spacing between differential lines

Routing guidelines:

- Single-Ended Trace Spacing > 3·w.
- Differential Pair Spacing > 3·s.
- Minimum Bend Radius > 2·w (to inside edge of trace).
- Serpentine Spacing > 4·w.

F.4 High-Speed ADC Routing Constraints

The high-speed ADC interface can be routed using a shortest length method. The signals in this group are shown in Table F.4.

Table F.4: High-Speed ADC Matched Length Sets

High-Speed ADC Matched Length Sets		
Matched Set	**Signal Name**	**Tolerance**
ADC_DIFF	FPGA_ADC_OVR_P	NA
	FPGA_ADC_OVR_N	
	FPGA_ADC_DATA_RDY_P	
	FPGA_ADC_DATA_RDY_N	
	FPGA_ADC_DATA_P[15:0]	
	FPGA_ADC_DATA_N[15:0]	

F.5 High-Speed DAC Routing Constraints

The high-speed DAC interface can be routed using a shortest length method. The signals in this group are shown in Table F.5.

Table F.5: High-Speed DAC Matched Length Sets

High-Speed DAC Matched Length Sets		
Matched Set	**Signal Name**	**Tolerance**
DAC_DIFF	FPGA_DAC_CLK_P FPGA_DAC_CLK_N FPGA_DAC_SYNC_P FPGA_DAC_SYNC_N FPGA_DAC_DATA_P[15:0] FPGA_DAC_DATA_N[15:0]	NA

F.6 High-Speed AsAP Routing Constraints

The high-speed AsAP interfaces can be routed using a shortest length method. The signals in this group are shown in Table F.6.

Table F.6: AsAP Matched Length Sets

AsAP Matched Length Sets		
Matched Set	**Signal Name**	**Tolerance**
ASAP1_A	FPGA_ASAP1_REQ_OUT FPGA_ASAP1_VLD_OUT FPGA_ASAP1_CLK_OUT FPGA_ASAP1_DATA_OUT[15:0]	NA
ASAP1_B	FPGA_ASAP1_REQ_IN FPGA_ASAP1_VLD_IN FPGA_ASAP1_CLK_IN FPGA_ASAP1_DATA_IN[15:0]	NA
ASAP2_A	FPGA_ASAP2_REQ_OUT FPGA_ASAP2_VLD_OUT FPGA_ASAP2_CLK_OUT FPGA_ASAP2_DATA_OUT[15:0]	NA
		Continued on Next Page. . .

AsAP Matched Length Sets		
Matched Set	**Signal Name**	**Tolerance**
ASAP2_B	FPGA_ASAP2_REQ_IN	NA
	FPGA_ASAP2_VLD_IN	
	FPGA_ASAP2_CLK_IN	
	FPGA_ASAP2_DATA_IN[15:0]	

Appendix G

Printed Circuit Board Net Type Assignments

Table G.1: Nets assigned to Net Types

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
AD9516_1GHZ_VCO_OUT	SE_50	SE_50
AD9516_10MHZ_REF_N	DIFF_100	DIFF_100
AD9516_10MHZ_REF_P	DIFF_100	DIFF_100
AD9516_BYPASS	POWER_15MIL	POWER_15MIL
ADC_IN_N	DIFF_100_O	DIFF_100_O
ADC_IN_P	DIFF_100_O	DIFF_100_O
ADC_VREF	POWER_15MIL	POWER_15MIL
AIF_ADC_CLKIN_N	DIFF_100	DIFF_100
AIF_ADC_CLKIN_P	DIFF_100	DIFF_100
AIF_ADC_CLK_N	DIFF_100	DIFF_100
AIF_ADC_CLK_P	DIFF_100	DIFF_100
AMC6821_FAN1_A0	DEFAULT	DEFAULT
AMC6821_FAN1_A1	DEFAULT	DEFAULT
AMC6821_FAN1_FAULTN	DEFAULT	DEFAULT
AMC6821_FAN1_OVRN	DEFAULT	DEFAULT
AMC6821_FAN1_THERMN	DEFAULT	DEFAULT
AMC6821_FAN2_A0	DEFAULT	DEFAULT
AMC6821_FAN2_A1	DEFAULT	DEFAULT
AMC6821_FAN2_FAULTN	DEFAULT	DEFAULT
AMC6821_FAN2_OVRN	DEFAULT	DEFAULT

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
AMC6821_FAN2_THERMN	DEFAULT	DEFAULT
ANLG_INHIBIT	POWER_15MIL	POWER_15MIL
ASAP1_ANLG1	SE_50	SE_50
ASAP1_ANLG2	SE_50	SE_50
ASAP1_ANLG3	SE_50	SE_50
ASAP1_ANLG4	SE_50	SE_50
ASAP1_CFG_CLK	SE_50	SE_50
ASAP1_CFG_VALID	SE_50	SE_50
ASAP1_EXT_CLK_IN	SE_50	SE_50
ASAP1_RESET_COLD	SE_50	SE_50
ASAP1_RST_CNTCLK	SE_50	SE_50
ASAP1_SPI_CLK	SE_50	SE_50
ASAP1_SPI_CSN	SE_50	SE_50
ASAP1_SPI_LOAD	SE_50	SE_50
ASAP1_SPI_MISO	SE_50	SE_50
ASAP1_SPI_MOSI	SE_50	SE_50
ASAP1_TEST0	SE_50	SE_50
ASAP1_TEST1	SE_50	SE_50
ASAP1_TEST2	SE_50	SE_50
ASAP1_TEST3	SE_50	SE_50
ASAP1_TEST4	SE_50	SE_50
ASAP1_TEST5	SE_50	SE_50
ASAP1_TEST6	SE_50	SE_50
ASAP1_TEST7	SE_50	SE_50
ASAP1_TEST8	SE_50	SE_50
ASAP1_TEST_OUT0	SE_50	SE_50
ASAP1_TEST_OUT1	SE_50	SE_50
ASAP1_TEST_OUT2	SE_50	SE_50
ASAP1_TEST_OUT3	SE_50	SE_50
ASAP1_TEST_OUT4	SE_50	SE_50
ASAP1_TEST_OUT5	SE_50	SE_50
ASAP1_TEST_OUT6	SE_50	SE_50
ASAP1_TEST_OUT7	SE_50	SE_50
ASAP1_TEST_OUT8	SE_50	SE_50
ASAP2_ANLG1	SE_50	SE_50
ASAP2_ANLG2	SE_50	SE_50
ASAP2_ANLG3	SE_50	SE_50
ASAP2_ANLG4	SE_50	SE_50
ASAP2_CFG_CLK	SE_50	SE_50
ASAP2_CFG_VALID	SE_50	SE_50
		Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
ASAP2_EXT_CLK_IN	SE_50	SE_50
ASAP2_RESET_COLD	SE_50	SE_50
ASAP2_RST_CNTCLK	SE_50	SE_50
ASAP2_SPI_CLK	SE_50	SE_50
ASAP2_SPI_CSN	SE_50	SE_50
ASAP2_SPI_LOAD	SE_50	SE_50
ASAP2_SPI_MISO	SE_50	SE_50
ASAP2_SPI_MOSI	SE_50	SE_50
ASAP2_TEST0	SE_50	SE_50
ASAP2_TEST1	SE_50	SE_50
ASAP2_TEST2	SE_50	SE_50
ASAP2_TEST3	SE_50	SE_50
ASAP2_TEST4	SE_50	SE_50
ASAP2_TEST5	SE_50	SE_50
ASAP2_TEST6	SE_50	SE_50
ASAP2_TEST7	SE_50	SE_50
ASAP2_TEST8	SE_50	SE_50
ASAP2_TEST_OUT0	SE_50	SE_50
ASAP2_TEST_OUT1	SE_50	SE_50
ASAP2_TEST_OUT2	SE_50	SE_50
ASAP2_TEST_OUT3	SE_50	SE_50
ASAP2_TEST_OUT4	SE_50	SE_50
ASAP2_TEST_OUT5	SE_50	SE_50
ASAP2_TEST_OUT6	SE_50	SE_50
ASAP2_TEST_OUT7	SE_50	SE_50
ASAP2_TEST_OUT8	SE_50	SE_50
ASAP_INHIBIT	POWER_15MIL	POWER_15MIL
ASAP_TRACK_CTRL1	POWER_15MIL	POWER_15MIL
ASAP_TRACK_CTRL_OUT1	POWER_15MIL	POWER_15MIL
ASAP_TRACK_CTRL_OUT2	POWER_15MIL	POWER_15MIL
AUX_IN	SE_50_O	SE_50_O
CLK10MHZ_CPLD	SE_50	SE_50
CLK10MHZ_CPLD_OUT	SE_50	SE_50
CLK10MHZ_EXT_BUF_N	DIFF_100	DIFF_100
CLK10MHZ_EXT_BUF_P	DIFF_100	DIFF_100
CLK10MHZ_INT_BUF_N	DIFF_100	DIFF_100
CLK10MHZ_INT_BUF_P	DIFF_100	DIFF_100
CLK10MHZ_REFOUT_N	DIFF_100	DIFF_100
CLK10MHZ_REFOUT_P	DIFF_100	DIFF_100
CLK10MHZ_REF_AD9516_N	DIFF_100	DIFF_100

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
CLK10MHZ_REF_AD9516_P	DIFF_100	DIFF_100
CLK10MHZ_REF_INT_LVDS_N	DIFF_100	DIFF_100
CLK10MHZ_REF_INT_LVDS_P	DIFF_100	DIFF_100
CLK10MHZ_REF_N	DIFF_100	DIFF_100
CLK10MHZ_REF_P	DIFF_100	DIFF_100
CLK10MHZ_REF_OSC	SE_50	SE_50
CLK10MHZ_REF_OUT	SE_50	SE_50
CPLD_BOARD_RSTN	SE_FPGA	SE_FPGA
CPLD_BUTTON	SE_FPGA	SE_FPGA
CPLD_CLK10MHZ_LVDS_N	DIFF_100	DIFF_100
CPLD_CLK10MHZ_LVDS_P	DIFF_100	DIFF_100
CPLD_LED	SE_FPGA	SE_FPGA
CPLD_TCK	SE_FPGA	SE_FPGA
CPLD_TDI	SE_FPGA	SE_FPGA
CPLD_TDO	SE_FPGA	SE_FPGA
CPLD_TMS	SE_FPGA	SE_FPGA
CUST_CCLK	SE_50	SE_50
CUST_CFG_DONE	SE_FPGA	SE_FPGA
CUST_INIT_B	SE_FPGA	SE_FPGA
CUST_PROG_B	SE_FPGA	SE_FPGA
CUST_TCK	SE_FPGA	SE_FPGA
CUST_TDI	SE_FPGA	SE_FPGA
CUST_TDO	SE_FPGA	SE_FPGA
CUST_TMS	SE_FPGA	SE_FPGA
DAC5682_EXTLO	SE_50	SE_50
DAC5682_UNUSED_N	DIFF_100_O	DIFF_100_O
DAC5682_UNUSED_P	DIFF_100_O	DIFF_100_O
DAC_SS_N	DIFF_100_O	DIFF_100_O
DAC_SS_P	DIFF_100_O	DIFF_100_O
DDR_SDRAM_ADDR0	SE_50	SE_50
DDR_SDRAM_ADDR1	SE_50	SE_50
DDR_SDRAM_ADDR2	SE_50	SE_50
DDR_SDRAM_ADDR3	SE_50	SE_50
DDR_SDRAM_ADDR4	SE_50	SE_50
DDR_SDRAM_ADDR5	SE_50	SE_50
DDR_SDRAM_ADDR6	SE_50	SE_50
DDR_SDRAM_ADDR7	SE_50	SE_50
DDR_SDRAM_ADDR8	SE_50	SE_50
DDR_SDRAM_ADDR9	SE_50	SE_50
DDR_SDRAM_ADDR10	SE_50	SE_50

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
DDR_SDRAM_ADDR11	SE_50	SE_50
DDR_SDRAM_ADDR12	SE_50	SE_50
DDR_SDRAM_BA0	SE_50	SE_50
DDR_SDRAM_BA1	SE_50	SE_50
DDR_SDRAM_CASN	SE_50	SE_50
DDR_SDRAM_CKE	SE_50	SE_50
DDR_SDRAM_CLK_N	DIFF_100	DIFF_100
DDR_SDRAM_CLK_P	DIFF_100	DIFF_100
DDR_SDRAM_CSN	SE_50	SE_50
DDR_SDRAM_DATA0	SE_50	SE_50
DDR_SDRAM_DATA1	SE_50	SE_50
DDR_SDRAM_DATA2	SE_50	SE_50
DDR_SDRAM_DATA3	SE_50	SE_50
DDR_SDRAM_DATA4	SE_50	SE_50
DDR_SDRAM_DATA5	SE_50	SE_50
DDR_SDRAM_DATA6	SE_50	SE_50
DDR_SDRAM_DATA7	SE_50	SE_50
DDR_SDRAM_DATA8	SE_50	SE_50
DDR_SDRAM_DATA9	SE_50	SE_50
DDR_SDRAM_DATA10	SE_50	SE_50
DDR_SDRAM_DATA11	SE_50	SE_50
DDR_SDRAM_DATA12	SE_50	SE_50
DDR_SDRAM_DATA13	SE_50	SE_50
DDR_SDRAM_DATA14	SE_50	SE_50
DDR_SDRAM_DATA15	SE_50	SE_50
DDR_SDRAM_LDM	SE_50	SE_50
DDR_SDRAM_LDQS	SE_50	SE_50
DDR_SDRAM_RASN	SE_50	SE_50
DDR_SDRAM_UDM	SE_50	SE_50
DDR_SDRAM_UDQS	SE_50	SE_50
DDR_SDRAM_WEN	SE_50	SE_50
DIG2_INHIBIT	POWER_15MIL	POWER_15MIL
DIG_INHIBIT	POWER_15MIL	POWER_15MIL
EXT_CLK10MHZ_REF	SE_50_O	SE_50_O
EXT_CLK10MHZ_REFOUT	SE_50_O	SE_50_O
FAN1_TACH	POWER_15MIL	POWER_15MIL
FAN2_TACH	POWER_15MIL	POWER_15MIL
FIFO_USB_N	DIFF_100	DIFF_100
FIFO_USB_P	DIFF_100	DIFF_100
FOX_10MHZ_REF	SE_50_O	SE_50_O
		Continued on Next Page. . .

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_AD9516_CSB	SE_FPGA	SE_FPGA
FPGA_AD9516_LD	SE_FPGA	SE_FPGA
FPGA_AD9516_PD	SE_FPGA	SE_FPGA
FPGA_AD9516_REFMON	SE_FPGA	SE_FPGA
FPGA_AD9516_REF_SEL	SE_FPGA	SE_FPGA
FPGA_AD9516_RESET	SE_FPGA	SE_FPGA
FPGA_AD9516_SCLK	SE_FPGA	SE_FPGA
FPGA_AD9516_SDIO	SE_FPGA	SE_FPGA
FPGA_AD9516_SDO	SE_FPGA	SE_FPGA
FPGA_AD9516_STATUS	SE_FPGA	SE_FPGA
FPGA_AD9516_SYNC	SE_FPGA	SE_FPGA
FPGA_ADC_DATA_N0	DIFF_100	DIFF_100
FPGA_ADC_DATA_N1	DIFF_100	DIFF_100
FPGA_ADC_DATA_N2	DIFF_100	DIFF_100
FPGA_ADC_DATA_N3	DIFF_100	DIFF_100
FPGA_ADC_DATA_N4	DIFF_100	DIFF_100
FPGA_ADC_DATA_N5	DIFF_100	DIFF_100
FPGA_ADC_DATA_N6	DIFF_100	DIFF_100
FPGA_ADC_DATA_N7	DIFF_100	DIFF_100
FPGA_ADC_DATA_N8	DIFF_100	DIFF_100
FPGA_ADC_DATA_N9	DIFF_100	DIFF_100
FPGA_ADC_DATA_N10	DIFF_100	DIFF_100
FPGA_ADC_DATA_N11	DIFF_100	DIFF_100
FPGA_ADC_DATA_N12	DIFF_100	DIFF_100
FPGA_ADC_DATA_N13	DIFF_100	DIFF_100
FPGA_ADC_DATA_N14	DIFF_100	DIFF_100
FPGA_ADC_DATA_N15	DIFF_100	DIFF_100
FPGA_ADC_DATA_P0	DIFF_100	DIFF_100
FPGA_ADC_DATA_P1	DIFF_100	DIFF_100
FPGA_ADC_DATA_P2	DIFF_100	DIFF_100
FPGA_ADC_DATA_P3	DIFF_100	DIFF_100
FPGA_ADC_DATA_P4	DIFF_100	DIFF_100
FPGA_ADC_DATA_P5	DIFF_100	DIFF_100
FPGA_ADC_DATA_P6	DIFF_100	DIFF_100
FPGA_ADC_DATA_P7	DIFF_100	DIFF_100
FPGA_ADC_DATA_P8	DIFF_100	DIFF_100
FPGA_ADC_DATA_P9	DIFF_100	DIFF_100
FPGA_ADC_DATA_P10	DIFF_100	DIFF_100
FPGA_ADC_DATA_P11	DIFF_100	DIFF_100
FPGA_ADC_DATA_P12	DIFF_100	DIFF_100

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_ADC_DATA_P13	DIFF_100	DIFF_100
FPGA_ADC_DATA_P14	DIFF_100	DIFF_100
FPGA_ADC_DATA_P15	DIFF_100	DIFF_100
FPGA_ADC_DATA_RDY_N	DIFF_100	DIFF_100
FPGA_ADC_DATA_RDY_P	DIFF_100	DIFF_100
FPGA_ADC_OVR_N	DIFF_100	DIFF_100
FPGA_ADC_OVR_P	DIFF_100	DIFF_100
FPGA_AMC6821_FAN1_FAULTN	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN1_OVRN	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN1_SCK	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN1_SDA	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN1_SMBALERTN	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN1_THERMN	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN2_FAULTN	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN2_OVRN	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN2_SCK	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN2_SDA	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN2_SMBALERTN	SE_FPGA	SE_FPGA
FPGA_AMC6821_FAN2_THERMN	SE_FPGA	SE_FPGA
FPGA_ASAP1_CFG_CLK	SE_50	SE_50
FPGA_ASAP1_CFG_VALID	SE_50	SE_50
FPGA_ASAP1_CLK_IN	SE_50	SE_50
FPGA_ASAP1_CLK_OUT	SE_50	SE_50
FPGA_ASAP1_DATA_IN0	SE_50	SE_50
FPGA_ASAP1_DATA_IN1	SE_50	SE_50
FPGA_ASAP1_DATA_IN2	SE_50	SE_50
FPGA_ASAP1_DATA_IN3	SE_50	SE_50
FPGA_ASAP1_DATA_IN4	SE_50	SE_50
FPGA_ASAP1_DATA_IN5	SE_50	SE_50
FPGA_ASAP1_DATA_IN6	SE_50	SE_50
FPGA_ASAP1_DATA_IN7	SE_50	SE_50
FPGA_ASAP1_DATA_IN8	SE_50	SE_50
FPGA_ASAP1_DATA_IN9	SE_50	SE_50
FPGA_ASAP1_DATA_IN10	SE_50	SE_50
FPGA_ASAP1_DATA_IN11	SE_50	SE_50
FPGA_ASAP1_DATA_IN12	SE_50	SE_50
FPGA_ASAP1_DATA_IN13	SE_50	SE_50
FPGA_ASAP1_DATA_IN14	SE_50	SE_50
FPGA_ASAP1_DATA_IN15	SE_50	SE_50
FPGA_ASAP1_DATA_OUT0	SE_50	SE_50
		Continued on Next Page. . .

Net Name	Net Type (Inner)	Net Type (Outer)
FPGA_ASAP1_DATA_OUT1	SE_50	SE_50
FPGA_ASAP1_DATA_OUT2	SE_50	SE_50
FPGA_ASAP1_DATA_OUT3	SE_50	SE_50
FPGA_ASAP1_DATA_OUT4	SE_50	SE_50
FPGA_ASAP1_DATA_OUT5	SE_50	SE_50
FPGA_ASAP1_DATA_OUT6	SE_50	SE_50
FPGA_ASAP1_DATA_OUT7	SE_50	SE_50
FPGA_ASAP1_DATA_OUT8	SE_50	SE_50
FPGA_ASAP1_DATA_OUT9	SE_50	SE_50
FPGA_ASAP1_DATA_OUT10	SE_50	SE_50
FPGA_ASAP1_DATA_OUT11	SE_50	SE_50
FPGA_ASAP1_DATA_OUT12	SE_50	SE_50
FPGA_ASAP1_DATA_OUT13	SE_50	SE_50
FPGA_ASAP1_DATA_OUT14	SE_50	SE_50
FPGA_ASAP1_DATA_OUT15	SE_50	SE_50
FPGA_ASAP1_MISO	SE_50	SE_50
FPGA_ASAP1_MOSI	SE_50	SE_50
FPGA_ASAP1_REQ_IN	SE_50	SE_50
FPGA_ASAP1_REQ_OUT	SE_50	SE_50
FPGA_ASAP1_RESET_COLD	SE_50	SE_50
FPGA_ASAP1_RST_CNTCLK	SE_50	SE_50
FPGA_ASAP1_SPI_CLK	SE_50	SE_50
FPGA_ASAP1_SPI_CSN	SE_50	SE_50
FPGA_ASAP1_SPI_LOAD	SE_50	SE_50
FPGA_ASAP1_VLD_IN	SE_50	SE_50
FPGA_ASAP1_VLD_OUT	SE_50	SE_50
FPGA_ASAP2_CFG_CLK	SE_50	SE_50
FPGA_ASAP2_CFG_VALID	SE_50	SE_50
FPGA_ASAP2_CLK_IN	SE_50	SE_50
FPGA_ASAP2_CLK_OUT	SE_50	SE_50
FPGA_ASAP2_DATA_IN0	SE_50	SE_50
FPGA_ASAP2_DATA_IN1	SE_50	SE_50
FPGA_ASAP2_DATA_IN2	SE_50	SE_50
FPGA_ASAP2_DATA_IN3	SE_50	SE_50
FPGA_ASAP2_DATA_IN4	SE_50	SE_50
FPGA_ASAP2_DATA_IN5	SE_50	SE_50
FPGA_ASAP2_DATA_IN6	SE_50	SE_50
FPGA_ASAP2_DATA_IN7	SE_50	SE_50
FPGA_ASAP2_DATA_IN8	SE_50	SE_50
FPGA_ASAP2_DATA_IN9	SE_50	SE_50

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_ASAP2_DATA_IN10	SE_50	SE_50
FPGA_ASAP2_DATA_IN11	SE_50	SE_50
FPGA_ASAP2_DATA_IN12	SE_50	SE_50
FPGA_ASAP2_DATA_IN13	SE_50	SE_50
FPGA_ASAP2_DATA_IN14	SE_50	SE_50
FPGA_ASAP2_DATA_IN15	SE_50	SE_50
FPGA_ASAP2_DATA_OUT0	SE_50	SE_50
FPGA_ASAP2_DATA_OUT1	SE_50	SE_50
FPGA_ASAP2_DATA_OUT2	SE_50	SE_50
FPGA_ASAP2_DATA_OUT3	SE_50	SE_50
FPGA_ASAP2_DATA_OUT4	SE_50	SE_50
FPGA_ASAP2_DATA_OUT5	SE_50	SE_50
FPGA_ASAP2_DATA_OUT6	SE_50	SE_50
FPGA_ASAP2_DATA_OUT7	SE_50	SE_50
FPGA_ASAP2_DATA_OUT8	SE_50	SE_50
FPGA_ASAP2_DATA_OUT9	SE_50	SE_50
FPGA_ASAP2_DATA_OUT10	SE_50	SE_50
FPGA_ASAP2_DATA_OUT11	SE_50	SE_50
FPGA_ASAP2_DATA_OUT12	SE_50	SE_50
FPGA_ASAP2_DATA_OUT13	SE_50	SE_50
FPGA_ASAP2_DATA_OUT14	SE_50	SE_50
FPGA_ASAP2_DATA_OUT15	SE_50	SE_50
FPGA_ASAP2_MISO	SE_50	SE_50
FPGA_ASAP2_MOSI	SE_50	SE_50
FPGA_ASAP2_REQ_IN	SE_50	SE_50
FPGA_ASAP2_REQ_OUT	SE_50	SE_50
FPGA_ASAP2_RESET_COLD	SE_50	SE_50
FPGA_ASAP2_RST_CNTCLK	SE_50	SE_50
FPGA_ASAP2_SPI_CLK	SE_50	SE_50
FPGA_ASAP2_SPI_CSN	SE_50	SE_50
FPGA_ASAP2_SPI_LOAD	SE_50	SE_50
FPGA_ASAP2_VLD_IN	SE_50	SE_50
FPGA_ASAP2_VLD_OUT	SE_50	SE_50
FPGA_ASAP_CLKIN_N	DIFF_100	DIFF_100
FPGA_ASAP_CLKIN_P	DIFF_100	DIFF_100
FPGA_ASAP_CLK_N	DIFF_100	DIFF_100
FPGA_ASAP_CLK_P	DIFF_100	DIFF_100
FPGA_AUX_IN_N	DIFF_100	DIFF_100
FPGA_AUX_IN_P	DIFF_100	DIFF_100
FPGA_BANK3_VRN	POWER_15MIL	POWER_15MIL
		Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_BANK3_VRP	POWER_15MIL	POWER_15MIL
FPGA_BANK11_VRN	POWER_15MIL	POWER_15MIL
FPGA_BANK11_VRP	POWER_15MIL	POWER_15MIL
FPGA_BANK12_VRN	POWER_15MIL	POWER_15MIL
FPGA_BANK12_VRP	POWER_15MIL	POWER_15MIL
FPGA_BANK15_VRN	POWER_15MIL	POWER_15MIL
FPGA_BANK15_VRP	POWER_15MIL	POWER_15MIL
FPGA_BANK19_VRN	POWER_15MIL	POWER_15MIL
FPGA_BANK19_VRP	POWER_15MIL	POWER_15MIL
FPGA_BANK20_VRN	POWER_15MIL	POWER_15MIL
FPGA_BANK20_VRP	POWER_15MIL	POWER_15MIL
FPGA_BANK21_VRN	POWER_15MIL	POWER_15MIL
FPGA_BANK21_VRP	POWER_15MIL	POWER_15MIL
FPGA_BOARD_RSTN	SE_FPGA	SE_FPGA
FPGA_CCLK	SE_50	SE_50
FPGA_CLK10MHZ_REF_CTRL0	SE_FPGA	SE_FPGA
FPGA_CLK10MHZ_REF_CTRL1	SE_FPGA	SE_FPGA
FPGA_CLK10MHZ_REF_CTRL2	SE_FPGA	SE_FPGA
FPGA_CLK10MHZ_REF_CTRL3	SE_FPGA	SE_FPGA
FPGA_CLK100MHZ_N	DIFF_100	DIFF_100
FPGA_CLK100MHZ_P	DIFF_100	DIFF_100
FPGA_CONFIG_DONE	SE_FPGA	SE_FPGA
FPGA_CS_B	SE_FPGA	SE_FPGA
FPGA_DAC5682_RSTB	SE_FPGA	SE_FPGA
FPGA_DAC5682_SCLK	SE_FPGA	SE_FPGA
FPGA_DAC5682_SDENB	SE_FPGA	SE_FPGA
FPGA_DAC5682_SDIO	SE_FPGA	SE_FPGA
FPGA_DAC5682_SDO	SE_FPGA	SE_FPGA
FPGA_DAC_CLK_N	DIFF_100	DIFF_100
FPGA_DAC_CLK_P	DIFF_100	DIFF_100
FPGA_DAC_DATA_N0	DIFF_100	DIFF_100
FPGA_DAC_DATA_N1	DIFF_100	DIFF_100
FPGA_DAC_DATA_N2	DIFF_100	DIFF_100
FPGA_DAC_DATA_N3	DIFF_100	DIFF_100
FPGA_DAC_DATA_N4	DIFF_100	DIFF_100
FPGA_DAC_DATA_N5	DIFF_100	DIFF_100
FPGA_DAC_DATA_N6	DIFF_100	DIFF_100
FPGA_DAC_DATA_N7	DIFF_100	DIFF_100
FPGA_DAC_DATA_N8	DIFF_100	DIFF_100
FPGA_DAC_DATA_N9	DIFF_100	DIFF_100

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_DAC_DATA_N10	DIFF_100	DIFF_100
FPGA_DAC_DATA_N11	DIFF_100	DIFF_100
FPGA_DAC_DATA_N12	DIFF_100	DIFF_100
FPGA_DAC_DATA_N13	DIFF_100	DIFF_100
FPGA_DAC_DATA_N14	DIFF_100	DIFF_100
FPGA_DAC_DATA_N15	DIFF_100	DIFF_100
FPGA_DAC_DATA_P0	DIFF_100	DIFF_100
FPGA_DAC_DATA_P1	DIFF_100	DIFF_100
FPGA_DAC_DATA_P2	DIFF_100	DIFF_100
FPGA_DAC_DATA_P3	DIFF_100	DIFF_100
FPGA_DAC_DATA_P4	DIFF_100	DIFF_100
FPGA_DAC_DATA_P5	DIFF_100	DIFF_100
FPGA_DAC_DATA_P6	DIFF_100	DIFF_100
FPGA_DAC_DATA_P7	DIFF_100	DIFF_100
FPGA_DAC_DATA_P8	DIFF_100	DIFF_100
FPGA_DAC_DATA_P9	DIFF_100	DIFF_100
FPGA_DAC_DATA_P10	DIFF_100	DIFF_100
FPGA_DAC_DATA_P11	DIFF_100	DIFF_100
FPGA_DAC_DATA_P12	DIFF_100	DIFF_100
FPGA_DAC_DATA_P13	DIFF_100	DIFF_100
FPGA_DAC_DATA_P14	DIFF_100	DIFF_100
FPGA_DAC_DATA_P15	DIFF_100	DIFF_100
FPGA_DAC_SYNC_N	DIFF_100	DIFF_100
FPGA_DAC_SYNC_P	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_A0	SE_50	SE_50
FPGA_DDR2_SDRAM_A1	SE_50	SE_50
FPGA_DDR2_SDRAM_A2	SE_50	SE_50
FPGA_DDR2_SDRAM_A3	SE_50	SE_50
FPGA_DDR2_SDRAM_A4	SE_50	SE_50
FPGA_DDR2_SDRAM_A5	SE_50	SE_50
FPGA_DDR2_SDRAM_A6	SE_50	SE_50
FPGA_DDR2_SDRAM_A7	SE_50	SE_50
FPGA_DDR2_SDRAM_A8	SE_50	SE_50
FPGA_DDR2_SDRAM_A9	SE_50	SE_50
FPGA_DDR2_SDRAM_A10	SE_50	SE_50
FPGA_DDR2_SDRAM_A11	SE_50	SE_50
FPGA_DDR2_SDRAM_A12	SE_50	SE_50
FPGA_DDR2_SDRAM_A13	SE_50	SE_50
FPGA_DDR2_SDRAM_A14	SE_50	SE_50
FPGA_DDR2_SDRAM_A15	SE_50	SE_50

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_DDR2_SDRAM_BA0	SE_50	SE_50
FPGA_DDR2_SDRAM_BA1	SE_50	SE_50
FPGA_DDR2_SDRAM_BA2	SE_50	SE_50
FPGA_DDR2_SDRAM_CASN	SE_50	SE_50
FPGA_DDR2_SDRAM_CKE0	SE_50	SE_50
FPGA_DDR2_SDRAM_CKE1	SE_50	SE_50
FPGA_DDR2_SDRAM_CK_N0	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_CK_N1	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_CK_P0	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_CK_P1	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DM0	SE_50	SE_50
FPGA_DDR2_SDRAM_DM1	SE_50	SE_50
FPGA_DDR2_SDRAM_DM2	SE_50	SE_50
FPGA_DDR2_SDRAM_DM3	SE_50	SE_50
FPGA_DDR2_SDRAM_DM4	SE_50	SE_50
FPGA_DDR2_SDRAM_DM5	SE_50	SE_50
FPGA_DDR2_SDRAM_DM6	SE_50	SE_50
FPGA_DDR2_SDRAM_DM7	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ0	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ1	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ2	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ3	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ4	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ5	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ6	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ7	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ8	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ9	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ10	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ11	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ12	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ13	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ14	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ15	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ16	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ17	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ18	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ19	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ20	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ21	SE_50	SE_50

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_DDR2_SDRAM_DQ22	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ23	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ24	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ25	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ26	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ27	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ28	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ29	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ30	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ31	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ32	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ33	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ34	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ35	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ36	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ37	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ38	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ39	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ40	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ41	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ42	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ43	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ44	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ45	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ46	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ47	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ48	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ49	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ50	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ51	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ52	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ53	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ54	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ55	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ56	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ57	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ58	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ59	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ60	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ61	SE_50	SE_50

Continued on Next Page. . .

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_DDR2_SDRAM_DQ62	SE_50	SE_50
FPGA_DDR2_SDRAM_DQ63	SE_50	SE_50
FPGA_DDR2_SDRAM_DQS0	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQS1	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQS2	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQS3	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQS4	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQS5	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQS6	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQS7	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQSN_NC0	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQSN_NC1	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQSN_NC2	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQSN_NC3	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQSN_NC4	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQSN_NC5	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQSN_NC6	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_DQSN_NC7	DIFF_100	DIFF_100
FPGA_DDR2_SDRAM_ODT0	SE_50	SE_50
FPGA_DDR2_SDRAM_ODT1	SE_50	SE_50
FPGA_DDR2_SDRAM_RASN	SE_50	SE_50
FPGA_DDR2_SDRAM_SCL	SE_50	SE_50
FPGA_DDR2_SDRAM_SDA	SE_50	SE_50
FPGA_DDR2_SDRAM_SN0	SE_50	SE_50
FPGA_DDR2_SDRAM_SN1	SE_50	SE_50
FPGA_DDR2_SDRAM_WEN	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR0	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR1	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR2	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR3	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR4	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR5	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR6	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR7	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR8	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR9	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR10	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR11	SE_50	SE_50
FPGA_DDR_SDRAM_ADDR12	SE_50	SE_50
FPGA_DDR_SDRAM_BA0	SE_50	SE_50

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_DDR_SDRAM_BA1	SE_50	SE_50
FPGA_DDR_SDRAM_CASN	SE_50	SE_50
FPGA_DDR_SDRAM_CKE	SE_50	SE_50
FPGA_DDR_SDRAM_CLKFB	SE_50	SE_50
FPGA_DDR_SDRAM_CLK_N	DIFF_100	DIFF_100
FPGA_DDR_SDRAM_CLK_P	DIFF_100	DIFF_100
FPGA_DDR_SDRAM_CSN	SE_50	SE_50
FPGA_DDR_SDRAM_DATA0	SE_50	SE_50
FPGA_DDR_SDRAM_DATA1	SE_50	SE_50
FPGA_DDR_SDRAM_DATA2	SE_50	SE_50
FPGA_DDR_SDRAM_DATA3	SE_50	SE_50
FPGA_DDR_SDRAM_DATA4	SE_50	SE_50
FPGA_DDR_SDRAM_DATA5	SE_50	SE_50
FPGA_DDR_SDRAM_DATA6	SE_50	SE_50
FPGA_DDR_SDRAM_DATA7	SE_50	SE_50
FPGA_DDR_SDRAM_DATA8	SE_50	SE_50
FPGA_DDR_SDRAM_DATA9	SE_50	SE_50
FPGA_DDR_SDRAM_DATA10	SE_50	SE_50
FPGA_DDR_SDRAM_DATA11	SE_50	SE_50
FPGA_DDR_SDRAM_DATA12	SE_50	SE_50
FPGA_DDR_SDRAM_DATA13	SE_50	SE_50
FPGA_DDR_SDRAM_DATA14	SE_50	SE_50
FPGA_DDR_SDRAM_DATA15	SE_50	SE_50
FPGA_DDR_SDRAM_LDM	SE_50	SE_50
FPGA_DDR_SDRAM_LDQS	SE_50	SE_50
FPGA_DDR_SDRAM_RASN	SE_50	SE_50
FPGA_DDR_SDRAM_UDM	SE_50	SE_50
FPGA_DDR_SDRAM_UDQS	SE_50	SE_50
FPGA_DDR_SDRAM_WEN	SE_50	SE_50
FPGA_DP_CTRL_CSN	SE_FPGA	SE_FPGA
FPGA_DP_CTRL_GPIO0	SE_FPGA	SE_FPGA
FPGA_DP_CTRL_GPIO1	SE_FPGA	SE_FPGA
FPGA_DP_CTRL_GPIO2	SE_FPGA	SE_FPGA
FPGA_DP_CTRL_GPIO3	SE_FPGA	SE_FPGA
FPGA_DP_CTRL_GPIO4	SE_FPGA	SE_FPGA
FPGA_DP_CTRL_INTN	SE_FPGA	SE_FPGA
FPGA_DP_CTRL_MISO	SE_FPGA	SE_FPGA
FPGA_DP_CTRL_MOSI	SE_FPGA	SE_FPGA
FPGA_DP_CTRL_RSTN	SE_FPGA	SE_FPGA
FPGA_DP_CTRL_SCK	SE_FPGA	SE_FPGA

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_DSP_CLKIN_N	DIFF_100	DIFF_100
FPGA_DSP_CLKIN_P	DIFF_100	DIFF_100
FPGA_DSP_CLK_N	DIFF_100	DIFF_100
FPGA_DSP_CLK_P	DIFF_100	DIFF_100
FPGA_EXT_CLK10MHZ_LOS	SE_FPGA	SE_FPGA
FPGA_EXT_CLK10MHZ_REF_N	DIFF_100	DIFF_100
FPGA_EXT_CLK10MHZ_REF_P	DIFF_100	DIFF_100
FPGA_FTDI_DATA0	SE_FPGA	SE_FPGA
FPGA_FTDI_DATA1	SE_FPGA	SE_FPGA
FPGA_FTDI_DATA2	SE_FPGA	SE_FPGA
FPGA_FTDI_DATA3	SE_FPGA	SE_FPGA
FPGA_FTDI_DATA4	SE_FPGA	SE_FPGA
FPGA_FTDI_DATA5	SE_FPGA	SE_FPGA
FPGA_FTDI_DATA6	SE_FPGA	SE_FPGA
FPGA_FTDI_DATA7	SE_FPGA	SE_FPGA
FPGA_FTDI_PWRENN	SE_FPGA	SE_FPGA
FPGA_FTDI_RDN	SE_FPGA	SE_FPGA
FPGA_FTDI_RSTOUTN	SE_FPGA	SE_FPGA
FPGA_FTDI_RXFN	SE_FPGA	SE_FPGA
FPGA_FTDI_SI_WU	SE_FPGA	SE_FPGA
FPGA_FTDI_TXEN	SE_FPGA	SE_FPGA
FPGA_FTDI_WRN	SE_FPGA	SE_FPGA
FPGA_HWID0	SE_FPGA	SE_FPGA
FPGA_HWID1	SE_FPGA	SE_FPGA
FPGA_INIT_B	SE_FPGA	SE_FPGA
FPGA_INT_CLK10MHZ_REF_N	DIFF_100	DIFF_100
FPGA_INT_CLK10MHZ_REF_P	DIFF_100	DIFF_100
FPGA_IODELAY_CLK_N	DIFF_100	DIFF_100
FPGA_IODELAY_CLK_P	DIFF_100	DIFF_100
FPGA_LA_CLK	SE_FPGA	SE_FPGA
FPGA_LA_DATA0	SE_FPGA	SE_FPGA
FPGA_LA_DATA1	SE_FPGA	SE_FPGA
FPGA_LA_DATA2	SE_FPGA	SE_FPGA
FPGA_LA_DATA3	SE_FPGA	SE_FPGA
FPGA_LA_DATA4	SE_FPGA	SE_FPGA
FPGA_LA_DATA5	SE_FPGA	SE_FPGA
FPGA_LA_DATA6	SE_FPGA	SE_FPGA
FPGA_LA_DATA7	SE_FPGA	SE_FPGA
FPGA_LEDS0	SE_FPGA	SE_FPGA
FPGA_LEDS1	SE_FPGA	SE_FPGA

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_LEDS2	SE_FPGA	SE_FPGA
FPGA_LEDS3	SE_FPGA	SE_FPGA
FPGA_LOCAL_RSTN	SE_FPGA	SE_FPGA
FPGA_PROG_B	SE_FPGA	SE_FPGA
FPGA_PUSHBUTTON0	SE_FPGA	SE_FPGA
FPGA_PUSHBUTTON1	SE_FPGA	SE_FPGA
FPGA_PUSHBUTTON2	SE_FPGA	SE_FPGA
FPGA_RDWR_B	SE_FPGA	SE_FPGA
FPGA_REACH_RX	SE_FPGA	SE_FPGA
FPGA_REACH_TX	SE_FPGA	SE_FPGA
FPGA_RS232_CTS	SE_FPGA	SE_FPGA
FPGA_RS232_RTS	SE_FPGA	SE_FPGA
FPGA_RS232_RX	SE_FPGA	SE_FPGA
FPGA_RS232_TX	SE_FPGA	SE_FPGA
FPGA_S3A_BOARD_RSTN	SE_FPGA	SE_FPGA
FPGA_SDRAM_CLK_N	DIFF_100	DIFF_100
FPGA_SDRAM_CLK_P	DIFF_100	DIFF_100
FPGA_SD_BUSY_LED	SE_FPGA	SE_FPGA
FPGA_SD_CARD_DETECT	SE_FPGA	SE_FPGA
FPGA_SD_CLK	SE_FPGA	SE_FPGA
FPGA_SD_RXD	SE_FPGA	SE_FPGA
FPGA_SD_TXD	SE_FPGA	SE_FPGA
FPGA_SLOTID0	SE_FPGA	SE_FPGA
FPGA_SLOTID1	SE_FPGA	SE_FPGA
FPGA_SLOTID2	SE_FPGA	SE_FPGA
FPGA_SRAM_ADDR0	SE_50	SE_50
FPGA_SRAM_ADDR1	SE_50	SE_50
FPGA_SRAM_ADDR2	SE_50	SE_50
FPGA_SRAM_ADDR3	SE_50	SE_50
FPGA_SRAM_ADDR4	SE_50	SE_50
FPGA_SRAM_ADDR5	SE_50	SE_50
FPGA_SRAM_ADDR6	SE_50	SE_50
FPGA_SRAM_ADDR7	SE_50	SE_50
FPGA_SRAM_ADDR8	SE_50	SE_50
FPGA_SRAM_ADDR9	SE_50	SE_50
FPGA_SRAM_ADDR10	SE_50	SE_50
FPGA_SRAM_ADDR11	SE_50	SE_50
FPGA_SRAM_ADDR12	SE_50	SE_50
FPGA_SRAM_ADDR13	SE_50	SE_50
FPGA_SRAM_ADDR14	SE_50	SE_50
		Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_SRAM_ADDR15	SE_50	SE_50
FPGA_SRAM_ADDR16	SE_50	SE_50
FPGA_SRAM_ADDR17	SE_50	SE_50
FPGA_SRAM_BWN0	SE_50	SE_50
FPGA_SRAM_BWN1	SE_50	SE_50
FPGA_SRAM_BWN2	SE_50	SE_50
FPGA_SRAM_BWN3	SE_50	SE_50
FPGA_SRAM_CLK_N	DIFF_100	DIFF_100
FPGA_SRAM_CLK_P	DIFF_100	DIFF_100
FPGA_SRAM_CQ_CLK_N	DIFF_100	DIFF_100
FPGA_SRAM_CQ_CLK_P	DIFF_100	DIFF_100
FPGA_SRAM_C_CLK_N	DIFF_100	DIFF_100
FPGA_SRAM_C_CLK_P	DIFF_100	DIFF_100
FPGA_SRAM_K_CLK_N	DIFF_100	DIFF_100
FPGA_SRAM_K_CLK_P	DIFF_100	DIFF_100
FPGA_SRAM_RDATA0	SE_50	SE_50
FPGA_SRAM_RDATA1	SE_50	SE_50
FPGA_SRAM_RDATA2	SE_50	SE_50
FPGA_SRAM_RDATA3	SE_50	SE_50
FPGA_SRAM_RDATA4	SE_50	SE_50
FPGA_SRAM_RDATA5	SE_50	SE_50
FPGA_SRAM_RDATA6	SE_50	SE_50
FPGA_SRAM_RDATA7	SE_50	SE_50
FPGA_SRAM_RDATA8	SE_50	SE_50
FPGA_SRAM_RDATA9	SE_50	SE_50
FPGA_SRAM_RDATA10	SE_50	SE_50
FPGA_SRAM_RDATA11	SE_50	SE_50
FPGA_SRAM_RDATA12	SE_50	SE_50
FPGA_SRAM_RDATA13	SE_50	SE_50
FPGA_SRAM_RDATA14	SE_50	SE_50
FPGA_SRAM_RDATA15	SE_50	SE_50
FPGA_SRAM_RDATA16	SE_50	SE_50
FPGA_SRAM_RDATA17	SE_50	SE_50
FPGA_SRAM_RDATA18	SE_50	SE_50
FPGA_SRAM_RDATA19	SE_50	SE_50
FPGA_SRAM_RDATA20	SE_50	SE_50
FPGA_SRAM_RDATA21	SE_50	SE_50
FPGA_SRAM_RDATA22	SE_50	SE_50
FPGA_SRAM_RDATA23	SE_50	SE_50
FPGA_SRAM_RDATA24	SE_50	SE_50

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_SRAM_RDATA25	SE_50	SE_50
FPGA_SRAM_RDATA26	SE_50	SE_50
FPGA_SRAM_RDATA27	SE_50	SE_50
FPGA_SRAM_RDATA28	SE_50	SE_50
FPGA_SRAM_RDATA29	SE_50	SE_50
FPGA_SRAM_RDATA30	SE_50	SE_50
FPGA_SRAM_RDATA31	SE_50	SE_50
FPGA_SRAM_RDATA32	SE_50	SE_50
FPGA_SRAM_RDATA33	SE_50	SE_50
FPGA_SRAM_RDATA34	SE_50	SE_50
FPGA_SRAM_RDATA35	SE_50	SE_50
FPGA_SRAM_RDN	SE_50	SE_50
FPGA_SRAM_WDATA0	SE_50	SE_50
FPGA_SRAM_WDATA1	SE_50	SE_50
FPGA_SRAM_WDATA2	SE_50	SE_50
FPGA_SRAM_WDATA3	SE_50	SE_50
FPGA_SRAM_WDATA4	SE_50	SE_50
FPGA_SRAM_WDATA5	SE_50	SE_50
FPGA_SRAM_WDATA6	SE_50	SE_50
FPGA_SRAM_WDATA7	SE_50	SE_50
FPGA_SRAM_WDATA8	SE_50	SE_50
FPGA_SRAM_WDATA9	SE_50	SE_50
FPGA_SRAM_WDATA10	SE_50	SE_50
FPGA_SRAM_WDATA11	SE_50	SE_50
FPGA_SRAM_WDATA12	SE_50	SE_50
FPGA_SRAM_WDATA13	SE_50	SE_50
FPGA_SRAM_WDATA14	SE_50	SE_50
FPGA_SRAM_WDATA15	SE_50	SE_50
FPGA_SRAM_WDATA16	SE_50	SE_50
FPGA_SRAM_WDATA17	SE_50	SE_50
FPGA_SRAM_WDATA18	SE_50	SE_50
FPGA_SRAM_WDATA19	SE_50	SE_50
FPGA_SRAM_WDATA20	SE_50	SE_50
FPGA_SRAM_WDATA21	SE_50	SE_50
FPGA_SRAM_WDATA22	SE_50	SE_50
FPGA_SRAM_WDATA23	SE_50	SE_50
FPGA_SRAM_WDATA24	SE_50	SE_50
FPGA_SRAM_WDATA25	SE_50	SE_50
FPGA_SRAM_WDATA26	SE_50	SE_50
FPGA_SRAM_WDATA27	SE_50	SE_50

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_SRAM_WDATA28	SE_50	SE_50
FPGA_SRAM_WDATA29	SE_50	SE_50
FPGA_SRAM_WDATA30	SE_50	SE_50
FPGA_SRAM_WDATA31	SE_50	SE_50
FPGA_SRAM_WDATA32	SE_50	SE_50
FPGA_SRAM_WDATA33	SE_50	SE_50
FPGA_SRAM_WDATA34	SE_50	SE_50
FPGA_SRAM_WDATA35	SE_50	SE_50
FPGA_SRAM_WRN	SE_50	SE_50
FPGA_TMPSENS_1A_CSN	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1A_MISO	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1A_MOSI	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1A_SCK	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1B_CSN	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1B_MISO	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1B_MOSI	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1B_SCK	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1C_CSN	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1C_MISO	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1C_MOSI	SE_FPGA	SE_FPGA
FPGA_TMPSENS_1C_SCK	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2A_CSN	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2A_MISO	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2A_MOSI	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2A_SCK	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2B_CSN	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2B_MISO	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2B_MOSI	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2B_SCK	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2C_CSN	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2C_MISO	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2C_MOSI	SE_FPGA	SE_FPGA
FPGA_TMPSENS_2C_SCK	SE_FPGA	SE_FPGA
FPGA_TRIG_IN_N	DIFF_100	DIFF_100
FPGA_TRIG_IN_P	DIFF_100	DIFF_100
FPGA_TRIG_OUT_N	DIFF_100	DIFF_100
FPGA_TRIG_OUT_P	DIFF_100	DIFF_100
FPGA_UI_CSN	SE_FPGA	SE_FPGA
FPGA_UI_INTN	SE_FPGA	SE_FPGA
FPGA_UI_MISO	SE_FPGA	SE_FPGA

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_UI_MOSI	SE_FPGA	SE_FPGA
FPGA_UI_PROG0	SE_FPGA	SE_FPGA
FPGA_UI_PROG1	SE_FPGA	SE_FPGA
FPGA_UI_PROG2	SE_FPGA	SE_FPGA
FPGA_UI_RDY	SE_FPGA	SE_FPGA
FPGA_UI_RSTN	SE_FPGA	SE_FPGA
FPGA_UI_SCK	SE_FPGA	SE_FPGA
FPGA_V5_BOARD_RSTN	SE_FPGA	SE_FPGA
FPGA_V5_CLK100MHZ_N	DIFF_100	DIFF_100
FPGA_V5_CLK100MHZ_P	DIFF_100	DIFF_100
FPGA_V5_FTDI_DATA0	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_DATA1	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_DATA2	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_DATA3	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_DATA4	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_DATA5	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_DATA6	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_DATA7	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_PWRENN	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_RDN	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_RSTOUTN	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_RXFN	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_SI_WU	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_TXEN	SE_FPGA	SE_FPGA
FPGA_V5_FTDI_WRN	SE_FPGA	SE_FPGA
FPGA_V5_LA_CLK	SE_FPGA	SE_FPGA
FPGA_V5_LA_DATA0	SE_FPGA	SE_FPGA
FPGA_V5_LA_DATA1	SE_FPGA	SE_FPGA
FPGA_V5_LA_DATA2	SE_FPGA	SE_FPGA
FPGA_V5_LA_DATA3	SE_FPGA	SE_FPGA
FPGA_V5_LA_DATA4	SE_FPGA	SE_FPGA
FPGA_V5_LA_DATA5	SE_FPGA	SE_FPGA
FPGA_V5_LA_DATA6	SE_FPGA	SE_FPGA
FPGA_V5_LA_DATA7	SE_FPGA	SE_FPGA
FPGA_V5_LEDS0	SE_FPGA	SE_FPGA
FPGA_V5_LEDS1	SE_FPGA	SE_FPGA
FPGA_V5_LEDS2	SE_FPGA	SE_FPGA
FPGA_V5_LEDS3	SE_FPGA	SE_FPGA
FPGA_V5_PUSHBUTTON0	SE_FPGA	SE_FPGA
FPGA_V5_PUSHBUTTON1	SE_FPGA	SE_FPGA

Continued on Next Page. . .

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
FPGA_V5_RS232_CTS	SE_FPGA	SE_FPGA
FPGA_V5_RS232_RTS	SE_FPGA	SE_FPGA
FPGA_V5_RS232_RX	SE_FPGA	SE_FPGA
FPGA_V5_RS232_TX	SE_FPGA	SE_FPGA
FPGA_V5_SD_BUSY_LED	SE_FPGA	SE_FPGA
FPGA_V5_SD_CARD_DETECT	SE_FPGA	SE_FPGA
FPGA_V5_SD_CLK	SE_FPGA	SE_FPGA
FPGA_V5_SD_RXD	SE_FPGA	SE_FPGA
FPGA_V5_SD_TXD	SE_FPGA	SE_FPGA
FP_AUX_IN	SE_50	SE_50
FP_TRIG_IN	SE_50	SE_50
FTDI_EECS	SE_FPGA	SE_FPGA
FTDI_EESCK	SE_FPGA	SE_FPGA
FTDI_EESDIO	SE_FPGA	SE_FPGA
FTDI_POWER	POWER_25MIL	POWER_25MIL
FTDI_RESETN	SE_FPGA	SE_FPGA
FTDI_USBDM	DIFF_100	DIFF_100
FTDI_USBDP	DIFF_100	DIFF_100
FTDI_XTAL_IN	SE_50	SE_50
FTDI_XTAL_OUT	SE_50	SE_50
GND	POWER_25MIL	POWER_25MIL
GNDA_FPGA	POWER_15MIL	POWER_15MIL
GND_MAIN	POWER_50MIL	POWER_50MIL
IF_AAF_IN	SE_50_O	SE_50_O
IF_AAF_OUT	SE_50_O	SE_50_O
IF_IN	SE_50_O	SE_50_O
IF_LNA_IN	SE_50_O	SE_50_O
IF_LNA_OUT	SE_50_O	SE_50_O
IF_PREAMP_CM	SE_50	SE_50
IF_PREAMP_IN	SE_50_O	SE_50_O
IF_PREAMP_OUT_N	DIFF_100_O	DIFF_100_O
IF_PREAMP_OUT_P	DIFF_100_O	DIFF_100_O
JTAG_CCN	SE_FPGA	SE_FPGA
N2V5A_AIF	POWER_25MIL	POWER_25MIL
N2V5A_IFLNA	POWER_25MIL	POWER_25MIL
N5V2AF_AIF	POWER_25MIL	POWER_25MIL
N5V2AF_AIF_IN	POWER_25MIL	POWER_25MIL
N5V2A_AIF	POWER_25MIL	POWER_25MIL
N5V2A_SS	POWER_25MIL	POWER_25MIL
N5V2A_SS_AMPF	POWER_25MIL	POWER_25MIL

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
N6VA	POWER_25MIL	POWER_25MIL
N6VAF_AIF	POWER_25MIL	POWER_25MIL
N6VAF_AIF_IN	POWER_25MIL	POWER_25MIL
N6VAF_SS	POWER_25MIL	POWER_25MIL
N6VAF_SS_IN	POWER_25MIL	POWER_25MIL
N6VA_OUT	POWER_25MIL	POWER_25MIL
P0V9D_MEM_VREF	POWER_15MIL	POWER_15MIL
P0V9D_SDRAM_VREF	POWER_15MIL	POWER_15MIL
P0V9D_SDRAM_VTT	POWER_15MIL	POWER_15MIL
P0V9D_SRAM_VREF	POWER_15MIL	POWER_15MIL
P0V9D_SRAM_VTT	POWER_15MIL	POWER_15MIL
P0V9D_VREF	POWER_15MIL	POWER_15MIL
P0V9D_VTT	POWER_15MIL	POWER_15MIL
P1V0D	POWER_25MIL	POWER_25MIL
P1V0D_ASAP	POWER_25MIL	POWER_25MIL
P1V0D_V5	POWER_25MIL	POWER_25MIL
P1V2D	POWER_25MIL	POWER_25MIL
P1V2D_S3A	POWER_15MIL	POWER_15MIL
P1V3D_ASAP	POWER_25MIL	POWER_25MIL
P1V8A_SS	POWER_25MIL	POWER_25MIL
P1V8A_SSF	POWER_25MIL	POWER_25MIL
P1V8D	POWER_25MIL	POWER_25MIL
P1V8D_SDRAM	POWER_25MIL	POWER_25MIL
P1V8D_SRAM	POWER_25MIL	POWER_25MIL
P1V8D_V5	POWER_25MIL	POWER_25MIL
P1V25D_DDR_VREF	POWER_15MIL	POWER_15MIL
P1V25D_DDR_VTT	POWER_15MIL	POWER_15MIL
P2V5A	POWER_25MIL	POWER_25MIL
P2V5A_AIF	POWER_25MIL	POWER_25MIL
P2V5A_IFLNA	POWER_25MIL	POWER_25MIL
P2V5A_TRIG	POWER_25MIL	POWER_25MIL
P2V5D	POWER_25MIL	POWER_25MIL
P2V5D_10MHZREF	POWER_25MIL	POWER_25MIL
P2V5D_S3A	POWER_25MIL	POWER_25MIL
P2V5D_V5	POWER_25MIL	POWER_25MIL
P2V5F_AUX_IN	POWER_15MIL	POWER_15MIL
P2V5F_TRIG_IN	POWER_15MIL	POWER_15MIL
P2V5F_TRIG_OUT	POWER_15MIL	POWER_15MIL
P2V5REF	POWER_15MIL	POWER_15MIL
P2V5REF_FPGA	POWER_15MIL	POWER_15MIL

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
P3V3A_10MHZREF	POWER_25MIL	POWER_25MIL
P3V3A_AD9516	POWER_25MIL	POWER_25MIL
P3V3A_ADC	POWER_25MIL	POWER_25MIL
P3V3A_AIF	POWER_25MIL	POWER_25MIL
P3V3A_CLK	POWER_25MIL	POWER_25MIL
P3V3A_CLKDIV	POWER_25MIL	POWER_25MIL
P3V3A_CLKDIVREF	POWER_25MIL	POWER_25MIL
P3V3A_SS	POWER_25MIL	POWER_25MIL
P3V3A_SSF	POWER_25MIL	POWER_25MIL
P3V3D	POWER_25MIL	POWER_25MIL
P3V3D_ADC	POWER_25MIL	POWER_25MIL
P3V3D_AIF	POWER_25MIL	POWER_25MIL
P3V3D_CP2102	POWER_25MIL	POWER_25MIL
P3V3D_CPLD	POWER_25MIL	POWER_25MIL
P3V3D_FAN1	POWER_25MIL	POWER_25MIL
P3V3D_FAN2	POWER_25MIL	POWER_25MIL
P3V3D_REACH	POWER_25MIL	POWER_25MIL
P3V3D_RST	POWER_25MIL	POWER_25MIL
P3V3D_S3A	POWER_25MIL	POWER_25MIL
P3V3D_SD	POWER_25MIL	POWER_25MIL
P3V3D_SDRAM	POWER_25MIL	POWER_25MIL
P3V3D_TPS74201	POWER_25MIL	POWER_25MIL
P3V3D_TRACK_CTRL	POWER_15MIL	POWER_15MIL
P3V3D_TRACK_CTRL_OUT	POWER_15MIL	POWER_15MIL
P3V3D_USB	POWER_25MIL	POWER_25MIL
P3V3D_V5	POWER_15MIL	POWER_15MIL
P3V3D_V5_CP2102	POWER_15MIL	POWER_15MIL
P3V3D_V5_SD	POWER_25MIL	POWER_25MIL
P3V3D_V5_USB	POWER_25MIL	POWER_25MIL
P5V2A	POWER_25MIL	POWER_25MIL
P5V2A_ADC	POWER_25MIL	POWER_25MIL
P5V2A_PREAMP	POWER_25MIL	POWER_25MIL
P5V2A_SS_AMPF	POWER_25MIL	POWER_25MIL
P5V2A_SS_FILT	POWER_25MIL	POWER_25MIL
P5V2A_VCO	POWER_25MIL	POWER_25MIL
P5V5A	POWER_25MIL	POWER_25MIL
P5V5AF	POWER_25MIL	POWER_25MIL
P5V5AF_AIF	POWER_25MIL	POWER_25MIL
P5V5AF_CLK	POWER_25MIL	POWER_25MIL
P5V5AF_CLKDIV	POWER_25MIL	POWER_25MIL

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
P5V5AF_SS	POWER_25MIL	POWER_25MIL
P5V5AF_THS4302	POWER_25MIL	POWER_25MIL
P5V5AF_TRIG	POWER_25MIL	POWER_25MIL
P5V5DF_AIF	POWER_25MIL	POWER_25MIL
P5VA_USB	POWER_25MIL	POWER_25MIL
P5VA_V5_USB	POWER_25MIL	POWER_25MIL
P5VD	POWER_25MIL	POWER_25MIL
P5VD_UI	POWER_25MIL	POWER_25MIL
P5VD_USB	POWER_25MIL	POWER_25MIL
P5VD_V5_USB	POWER_25MIL	POWER_25MIL
P8VAF_VCO	POWER_25MIL	POWER_25MIL
P8VA_CLKDIV	POWER_25MIL	POWER_25MIL
P12VA_TRACK_CTRL1	POWER_15MIL	POWER_15MIL
P12VA_TRACK_CTRL2	POWER_15MIL	POWER_15MIL
P12VD_TRACK_CTRL1	POWER_15MIL	POWER_15MIL
P12VFA	POWER_25MIL	POWER_25MIL
P12VFD	POWER_25MIL	POWER_25MIL
P12VFD_FAN	POWER_25MIL	POWER_25MIL
P12VFD_REACH	POWER_25MIL	POWER_25MIL
P12VF_FAN1	POWER_25MIL	POWER_25MIL
P12VF_FAN2	POWER_25MIL	POWER_25MIL
P12VF_IN	POWER_50MIL	POWER_50MIL
P12V_IN	POWER_50MIL	POWER_50MIL
P12V_MAIN	POWER_50MIL	POWER_50MIL
PTH08T220_SOUT1	SE_50	SE_50
PTH08T220_SOUT2	SE_50	SE_50
PTH08T220_SOUT3	SE_50	SE_50
PTH08T220_SOUT4	SE_50	SE_50
PTH08T220_SOUT5	SE_50	SE_50
PTH08T220_SOUT6	SE_50	SE_50
PTH08T220_SOUT7	SE_50	SE_50
PTH08T220_SYNC1	SE_50	SE_50
PTH08T220_SYNC2	SE_50	SE_50
PTH08T220_SYNC3	SE_50	SE_50
PTH08T220_SYNC4	SE_50	SE_50
PTH08T220_SYNC5	SE_50	SE_50
PTH08T220_SYNC6	SE_50	SE_50
PTH08T220_SYNC7	SE_50	SE_50
PTH08T260_SOUT8	SE_50	SE_50
PTH08T260_SOUT9	SE_50	SE_50

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
PTH08T260_SYNC8	SE_50	SE_50
PTH08T260_SYNC9	SE_50	SE_50
REACH_RX	SE_FPGA	SE_FPGA
REACH_TX	SE_FPGA	SE_FPGA
S3A_CCLK	SE_50	SE_50
S3A_CONFIG_DONE	SE_FPGA	SE_FPGA
S3A_INIT_B	SE_FPGA	SE_FPGA
S3A_M0	SE_FPGA	SE_FPGA
S3A_M1	SE_FPGA	SE_FPGA
S3A_M2	SE_FPGA	SE_FPGA
S3A_PROG_B	SE_FPGA	SE_FPGA
S3A_SPI_CSO	SE_FPGA	SE_FPGA
S3A_SPI_DATA_TO_V5	SE_FPGA	SE_FPGA
S3A_SPI_HOLDN	SE_FPGA	SE_FPGA
S3A_SPI_MISO	SE_FPGA	SE_FPGA
S3A_SPI_MOSI	SE_FPGA	SE_FPGA
S3A_SPI_WPN	SE_FPGA	SE_FPGA
S3A_V0	SE_FPGA	SE_FPGA
S3A_V1	SE_FPGA	SE_FPGA
S3A_V2	SE_FPGA	SE_FPGA
SRAM_DLL_OFFN	SE_FPGA	SE_FPGA
SS_AIF_IN_N	DIFF_100_O	DIFF_100_O
SS_AIF_IN_P	DIFF_100_O	DIFF_100_O
SS_AIF_OUT_N	DIFF_100_O	DIFF_100_O
SS_AIF_OUT_P	DIFF_100_O	DIFF_100_O
SS_AMP_OUT	SE_50_O	SE_50_O
SS_DAC_CLKIN_N	DIFF_100	DIFF_100
SS_DAC_CLKIN_P	DIFF_100	DIFF_100
SS_DAC_CLK_N	DIFF_100	DIFF_100
SS_DAC_CLK_P	DIFF_100	DIFF_100
SS_OUT	SE_50_O	SE_50_O
TCK	SE_FPGA	SE_FPGA
TCK_SRAM	SE_FPGA	SE_FPGA
TDI_SRAM	SE_FPGA	SE_FPGA
TDI_TO_S3A	SE_FPGA	SE_FPGA
TDI_TO_V5	SE_FPGA	SE_FPGA
TDO_SRAM	SE_FPGA	SE_FPGA
TDO_TO_JTAG	SE_FPGA	SE_FPGA
TEST_CLK_N	DIFF_100	DIFF_100
TEST_CLK_P	DIFF_100	DIFF_100

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
TMS	SE_FPGA	SE_FPGA
TMS_SRAM	SE_FPGA	SE_FPGA
TOUTN	SE_50_O	SE_50_O
TOUTP	SE_50_O	SE_50_O
TRIG_IN	SE_50_O	SE_50_O
TRIG_OUT	SE_50_O	SE_50_O
UI_CSN	SE_FPGA	SE_FPGA
UI_INTN	SE_FPGA	SE_FPGA
UI_MISO	SE_FPGA	SE_FPGA
UI_MOSI	SE_FPGA	SE_FPGA
UI_PROG0	SE_FPGA	SE_FPGA
UI_PROG1	SE_FPGA	SE_FPGA
UI_PROG2	SE_FPGA	SE_FPGA
UI_RDY	SE_FPGA	SE_FPGA
UI_RSTN	SE_FPGA	SE_FPGA
UI_SCK	SE_FPGA	SE_FPGA
USB_D_N	DIFF_100	DIFF_100
USB_D_P	DIFF_100	DIFF_100
USB_VBUS	POWER_25MIL	POWER_25MIL
V5_CCLK	SE_50	SE_50
V5_CONFIG_DONE	SE_FPGA	SE_FPGA
V5_FIFO_USB_N	DIFF_100	DIFF_100
V5_FIFO_USB_P	DIFF_100	DIFF_100
V5_FS0	SE_FPGA	SE_FPGA
V5_FS1	SE_FPGA	SE_FPGA
V5_FS2	SE_FPGA	SE_FPGA
V5_FTDI_EECS	SE_FPGA	SE_FPGA
V5_FTDI_EESCK	SE_FPGA	SE_FPGA
V5_FTDI_EESDIO	SE_FPGA	SE_FPGA
V5_FTDI_POWER	POWER_25MIL	POWER_25MIL
V5_FTDI_RESETN	SE_FPGA	SE_FPGA
V5_FTDI_USBDM	DIFF_100	DIFF_100
V5_FTDI_USBDP	DIFF_100	DIFF_100
V5_FTDI_XTAL_IN	SE_50	SE_50
V5_FTDI_XTAL_OUT	SE_50	SE_50
V5_HSWAPEN	SE_FPGA	SE_FPGA
V5_INIT_B	SE_FPGA	SE_FPGA
V5_M0	SE_FPGA	SE_FPGA
V5_M1	SE_FPGA	SE_FPGA
V5_M2	SE_FPGA	SE_FPGA

Continued on Next Page...

Net Name	**Net Type (Inner)**	**Net Type (Outer)**
V5_PROG_B	SE_FPGA	SE_FPGA
V5_USB_D_N	DIFF_100	DIFF_100
V5_USB_D_P	DIFF_100	DIFF_100
V5_USB_VBUS	POWER_25MIL	POWER_25MIL
VDDH_ASAP1	POWER_25MIL	POWER_25MIL
VDDH_ASAP2	POWER_25MIL	POWER_25MIL
VDDIO_ASAP1	POWER_25MIL	POWER_25MIL
VDDIO_ASAP2	POWER_25MIL	POWER_25MIL
VDDL_ASAP1	POWER_25MIL	POWER_25MIL
VDDL_ASAP2	POWER_25MIL	POWER_25MIL
VDDON_ASAP1	POWER_25MIL	POWER_25MIL
VDDON_ASAP2	POWER_25MIL	POWER_25MIL
VDDOSC_ASAP1	POWER_25MIL	POWER_25MIL
VDDOSC_ASAP2	POWER_25MIL	POWER_25MIL
VECTRON_10MHZ_REF	SE_50_O	SE_50_O
VP_VN_SM	SE_FPGA	SE_FPGA

Appendix H

Data Path FPGA IODELAY Results

H.1 IODELAY Script for High-Speed ADC Interface

Listing H.1: IODELAY Tap Calculation Perl Script

```
#!/usr/bin/perl

#*****************************************************************
#
# adc module
#
#*****************************************************************
#
# VCL Confidential Copyright   2010, UC Davis, ECE Department
#
#*****************************************************************
#
# created on:    06/09/2009
# created by:    jwwebb
# last edit on: $DateTime: $
# last edit by: $Author: $
# revision:      $Revision: $
# comments:      Generated
#
#*****************************************************************
# Revision List:
#
#                1.0      06/09/2009      Initial release
```

```
#
#*************************************************************
# Xilinx FPGA IODELAY Calculation Script
#
#  This utility is intended to make calculating Xilinx FPGA IODELAY
#  tap values for high-speed device interfaces.
#
#  The user will supply a comma separated value file containing
#  the signal name, length, and matched-length group. For example,
#
#          FPGA_CLK_P,3353.41,SIG_DML2_CON
#          FPGA_DATA0,2388.68,SIG_DML2_CON
#          FPGA_DATA1,1773.26,SIG_DML2_CON
#          FPGA_DATA2,1921.38,SIG_DML2_CON
#          FPGA_DATA3,2018.79,SIG_DML2_CON
#          FPGA_DATA4,1899.73,SIG_DML2_CON
#
#  This script will calculate the number of IODELAY taps, and
#  provide the results in the following file:
#
#          adc_delays.csv:  Comma Separated Value File for Excel Viewing
#
#  The script can be used by typing the following command at the
#  command line prompt:
#
#          [jwwebb@stormtater ~/bin/perldev/adc]
#          $ ./adc -f adc_lengths.txt
#          Delay file is ready for use.
#
#*************************************************************

#*************************************************************
# CPAN Modules
#*************************************************************
use strict;
use Getopt::Std;
use POSIX;

#*************************************************************
# Constants and Variables:
#*************************************************************
my (%opts)=();
my ($file);
my ($csv);
my ($debug);
my (%xilinxH, $xilinx_rH);

#*************************************************************
```

```
# Retrieve command line argument
#*************************************************************
getopts('hvf:c',\%opts);

#check for valid combination command-line arguments
if ($opts{h} || !$opts{f} || ($opts{f} && !$opts{c})) ) {
    print_usage();
    exit;
}

# parse command-line arguments
$file = $opts{f};
$csv = $opts{c};
$debug = $opts{v};

#*************************************************************
# Initialize Xilinx Hash:
#*************************************************************
$xilinxH{ 'file' } = $file;
$xilinxH{ 'debug' } = $debug;

#*************************************************************
# Print Module Declaration:
#*************************************************************
$xilinx_rH = getFile(\%xilinxH);
$xilinx_rH = calcDelay($xilinx_rH);
if ($csv)  {$xilinx_rH = writeDelayCSV($xilinx_rH);}
exit;

#-------------------------------------------------------------------------------
# Generic Error and Exit routine
#-------------------------------------------------------------------------------

sub dienice {
        my($errmsg) = @_;
        print"$errmsg\n";
        exit;
}

sub print_usage {
        my ($usage);    $usage = "\nUsage: $0 [-h] [-v] [-f <FILE>] [-c]\n";
        $usage .= "\n";
        $usage .= "\t-h\t\tPrint Help.\n";
        $usage .= "\t-v\t\tVerbose: Print Debug Information.\n";
        $usage .= "\t-f <FILE>\tInput CSV Signal File.\n";
        $usage .= "\t-c\t\tCSV: Generate CSV File with Estimated IODELAY Taps.\n";
        $usage .= "\n";
        $usage .= "\tExample:\n";
        $usage .= "\t\t$0 -v -f sample.txt \n";
        $usage .= "\t\t$0 -v -f sample.csv \n";
        $usage .= "\n";
        print($usage);  return;
}

sub getFile {
    #------------------------------------------------------------------------
    # Get Input File:
    #
    #  The sub-routine getFile() will open the input file, which is either a
    #  binary or text file and read its contents into an array. It will also
    #  determine the file length. The following parameters are created
    #
    #   * filedata:              @vdataA
    #   * fileLen:               scalar(@vdataA)
    #
    #  Usage: $xilinx_rH = getFile(\%xilinxH);
    #
    #------------------------------------------------------------------------
    my ($xilinx_rH) = shift;            # Read in user's variable.

    my (%xilinxH) = %{ $xilinx_rH };    # De-reference Xilinx hash.

    my ($file)  = $xilinxH{'file'};     # File Name
    my ($debug) = $xilinxH{'debug'};    # Print out Debug Info.

    #--------------------------------------------------------------------
    # Open the text file, and read the results into an array of hashes for
    # manipulating the data array. Close file when done.
    #
    # The Hash elements are:
    #
    #           adc_sigs_AoH
    #           {Signal => "x",
    #            LenMils => "x",
    #            MLGroup => "x",
    #            IsRef => "x",
    #            AvgLenMils => "x",
    #            LenPS => "x",
    #            AddedDelay => "x",
    #            NumTaps => "x",
    #           }
    #
    #--------------------------------------------------------------------
    my (@tmp);
    my (@adc_sigs_AoH);
    open(inF, "<", $file) or dienice ("$file open failed");
    while (<inF>) {
        chomp;
        @tmp = split(/,/, $_);
        $tmp[2] =~ s/[\r|\n]//;
        push(@adc_sigs_AoH, {Signal => $tmp[0],
                             LenMils => $tmp[1],
                             MLGroup => $tmp[2],
                             IsRef => "0",
                             AvgLenMils => "x",
                             LenPS => "x",
                             AddedDelay => "x",
                             NumTaps => "x"
                            }
             );
    }
    close(inF);

    #--------------------------------------------------------------------------
    # Push signals into Hash.
    #--------------------------------------------------------------------------
    push (@{ $xilinxH{ 'adc_sigs_AoH' } }, @adc_sigs_AoH);

    #--------------------------------------------------------------------------
    # Determine number of lines, and set beginning for loop index.
    #--------------------------------------------------------------------------
    $xilinxH{ 'adc_sigs_Num' } = scalar(@{ $xilinxH{ 'adc_sigs_AoH' } });

    #--------------------------------------------------------------------------
    # Create an Array with members only from SIG_DML1_CON.
    #--------------------------------------------------------------------------
    print("\n\n") if $debug;
    print("Total number of lines: $xilinxH{ 'adc_sigs_Num' }\n") if $debug;
    print("\n\n") if $debug;

    #------------------------------------------------------------------------------
    # Return data to user
    #------------------------------------------------------------------------------
    return \%xilinxH;
}

sub calcDelay {
    #------------------------------------------------------------------------------
    # Calculat Delays For Device Signals:
    #
    #  The sub-routine calcDelay() will calculated necessary delays for each signal.
    #
    #  Usage: $xilinx_rH = calcDelay(\%xilinxH);
    #
```

```
#-----------------------------------------------------------------------------------
my ($xilinx_rH) = shift;            # Read in user's variable.

my (%xilinxH) = %{ $xilinx_rH };    # De-reference Xilinx hash.

my ($file)  = $xilinxH{'file'};     # File Name
my (@adc_sigs_AoH);
push (@adc_sigs_AoH, @{ $xilinxH{ 'adc_sigs_AoH' } });
my ($adc_sigs_Num) = $xilinxH{ 'adc_sigs_Num' };
my ($debug) = $xilinxH{'debug'};    # Print out Debug Info.

#-------------------------------------------------------------------------------
# Prepare important variables.
#-------------------------------------------------------------------------------
my ($ML1_name) = "ADC_DIFF";
$xilinxH{ 'ML1_name' } = $ML1_name;

my ($Ref_name) = "FPGA_ADC_DATA_RDY_P";
$xilinxH{ 'Ref_name' } = $Ref_name;

my ($Ref_len) = 0;

# [(167 ps)/(1 inch)] * [(1 inch)/(1000 mils)];
my ($ps_per_in) = 167;
$xilinxH{ 'ps_per_in' } = $ps_per_in;
my ($ps_per_mil) = $ps_per_in*(1/1000);
$xilinxH{ 'ps_per_mil' } = $ps_per_mil;
my ($IODELAY_TAP_PS) = 78.125;
$xilinxH{ 'IODELAY_TAP_PS' } = $IODELAY_TAP_PS;

#-------------------------------------------------------------------------------
# Push each signal length into an array based on the appropriate
# Matched Length Group (i.e., SIG_DML1_CON). Also determine reference signal
# indexes and store refrence signal lengths in mils from the Array of Hashes.
#-------------------------------------------------------------------------------
my ($i);
my ($k);
my (@ML1_Lengths);
for $i ( 0 .. $#adc_sigs_AoH ) {
    print("ML_Group: $adc_sigs_AoH[$i]{MLGroup}.\n") if $debug;
    if ( $adc_sigs_AoH[$i]{MLGroup} eq $ML1_name ) {
        push(@ML1_Lengths, $adc_sigs_AoH[$i]{LenMils});
    }

    if ( $adc_sigs_AoH[$i]{Signal} eq $Ref_name ) {
        $adc_sigs_AoH[$i]{IsRef} = "1";
        $Ref_len = $adc_sigs_AoH[$i]{LenMils};
    }
```

```
}

#------------------------------------------------------------------------------
# Determine Average Length in Mils of each Matched Length Set using the
# reference signal lengths in Mils. Subtract the reference signal length
# from the overall length.
#------------------------------------------------------------------------------
my ($ML1_Len) = scalar(@ML1_Lengths);
my ($ML1_Total) = 0;
($ML1_Total+=$_) for @ML1_Lengths;
my ($ML1_Avg) = $ML1_Total/$ML1_Len;

#------------------------------------------------------------------------------
# Determine length in picoseconds of the reference signals.
#------------------------------------------------------------------------------
my ($Ref_lenPS) = $Ref_len*$ps_per_mil;

#------------------------------------------------------------------------------
# Calculate the following:
#
#   * Length in picoseconds (PS).
#   * Difference between signal length in PS and reference signal in PS.
#   * Number of IODELAY Taps Required to equalize length.
#
# The Hash elements are:
#
#          adc_sigs_AoH
#          {Signal => "x",
#           LenMils => "x",
#           MLGroup => "x",
#           IsRef => "x",
#           AvgLenMils => "x",
#           LenPS => "x",
#           AddedDelay => "x",
#           NumTaps => "x",
#          }
#
#------------------------------------------------------------------------------
my ($tmp_lenMils);
my ($tmp_lenPS);
my ($tmp_diffPS);
my ($tmp_numTaps);
my ($tmp_numTapsF);
my (@ML1_Taps);
for $i ( 0 .. $#adc_sigs_AoH ) {
    if ( $adc_sigs_AoH[$i]{MLGroup} eq $ML1_name ) {
        $adc_sigs_AoH[$i]{AvgLenMils} = $ML1_Avg;
```

```
            $tmp_lenMils = $adc_sigs_AoH[$i]{LenMils};
            $tmp_lenPS = $tmp_lenMils * $ps_per_mil;
            $tmp_diffPS = $Ref_lenPS - $tmp_lenPS;
            $tmp_numTaps = $tmp_diffPS/$IODELAY_TAP_PS;
            $tmp_numTapsF = floor(abs($tmp_numTaps));
            if ($tmp_diffPS<0) {$tmp_numTapsF *= -1;}
            push(@ML1_Taps, $tmp_numTapsF);
            print("Signal: $adc_sigs_AoH[$i]{Signal}, ");
            print("LenMils: $tmp_lenMils, ");
            print("LenPS: $tmp_lenPS, ");
            print("DiffPS: $tmp_diffPS, ");
            print("NumTaps: $tmp_numTaps, ");
            print("NumTaps(Floored): $tmp_numTapsF\n") if $debug;
            $adc_sigs_AoH[$i]{LenPS} = $tmp_lenPS;
            $adc_sigs_AoH[$i]{AddedDelay} = $tmp_diffPS;
            $adc_sigs_AoH[$i]{NumTaps} = $tmp_numTapsF;
        }

    }

    # Calculate Minimum Tap Delay of each Matched Length Group:
    my (@ML1_Taps_Sorted) = sort {$a <=> $b} @ML1_Taps;

    print("ML1 (minimum): $ML1_Taps_Sorted[0]\n") if $debug;

    my ($minTaps1) = $ML1_Taps_Sorted[0];
    my ($ML1_NewTap);
    my ($ML1_MinTap);
    my ($ML1_NumTaps);

    for $i ( 0 .. $#adc_sigs_AoH ) {
        if ( $adc_sigs_AoH[$i]{MLGroup} eq $ML1_name ) {
            $adc_sigs_AoH[$i]{MinTaps} = $minTaps1;
            $ML1_MinTap = $adc_sigs_AoH[$i]{MinTaps};
            $ML1_NumTaps = $adc_sigs_AoH[$i]{NumTaps};
            if ($ML1_MinTap < 0) {
                $ML1_NewTap = $ML1_NumTaps + ($ML1_MinTap*-1);
            } else {
                $ML1_NewTap = $ML1_NumTaps;
            }
            $adc_sigs_AoH[$i]{NumTapsNorm} = $ML1_NewTap;
        }

    }

    #------------------------------------------------------------------------
    # Push signals into Array of Hashes.
    #------------------------------------------------------------------------
    push (@{ $xilinxH{ 'adc_sigs_new_AoH' } }, @adc_sigs_AoH);

    #-------------------------------------------------------------------------------
    # Return data to user
    #-------------------------------------------------------------------------------
    return \%xilinxH;
}

sub writeDelayCSV {
    #-------------------------------------------------------------------------------
    # Write Out the Calculated Delays to a CSV File:
    #
    #  The sub-routine writeDelayCSV() will write calculated delays to a file.
    #
    #   * filedata:            @adcDelaysA
    #   * fileLen:             scalar(@adcDelaysA)
    #
    #  Usage: $xilinx_rH = writeDelayCSV(\%xilinxH);
    #
    #-------------------------------------------------------------------------------
    my ($xilinx_rH) = shift;           # Read in user's variable.

    my (%xilinxH) = %{ $xilinx_rH };   # De-reference Xilinx hash.

    my ($file)  = $xilinxH{'file'};    # File Name
    my (@adc_sigs_AoH);
    push (@adc_sigs_AoH, @{ $xilinxH{ 'adc_sigs_new_AoH' } });
    my ($adc_sigs_Num) = $xilinxH{ 'adc_sigs_Num' };
    my ($debug) = $xilinxH{'debug'};   # Print out Debug Info.

    #---------------------------------------------------------------------------
    # Setup Delay File Name:
    #---------------------------------------------------------------------------
    my ($delayfile) = "adc_delays.csv";
    $xilinxH{ 'delayfile' } = $delayfile;

    #---------------------------------------------------------------------------
    # Write out the contents of the Array of Hashes into a CSV file.
    # The Hash elements are:
    #
    #           adc_sigs_AoH
    #           {Signal => "x",
    #            LenMils => "x",
    #            MLGroup => "x",
    #            IsRef => "x",
    #            AvgLenMils => "x",
    #            LenPS => "x",
    #            AddedDelay => "x",
    #             NumTaps => "x",
    #           }
    #------------------------------------------------------------------------------
    open(outF, '>', $delayfile) or die "Couldn't open file for writing: $!n";
    my ($i);
    printf(outF "Signal,");
    printf(outF "LenMils,");
    printf(outF "MLGroup,");
    printf(outF "IsRef,");
    printf(outF "AvgLenMils,");
    printf(outF "LenPS,");
    printf(outF "AddedDelay,");
    printf(outF "NumTaps,");
    printf(outF "NumTapsNorm\n");
    for $i ( 0 .. $#adc_sigs_AoH ) {
        printf(outF "$adc_sigs_AoH[$i]{Signal},");
        printf(outF "$adc_sigs_AoH[$i]{LenMils},");
        printf(outF "$adc_sigs_AoH[$i]{MLGroup},");
        printf(outF "$adc_sigs_AoH[$i]{IsRef},");
        printf(outF "$adc_sigs_AoH[$i]{AvgLenMils},");
        printf(outF "$adc_sigs_AoH[$i]{LenPS},");
        printf(outF "$adc_sigs_AoH[$i]{AddedDelay},");
        printf(outF "$adc_sigs_AoH[$i]{NumTaps},");
        printf(outF "$adc_sigs_AoH[$i]{NumTapsNorm}\n");
    }
    close outF;

    print("Delay CSV file is ready for use.\n");

    #-------------------------------------------------------------------------------
    # Return data to user
    #-------------------------------------------------------------------------------
    return \%xilinxH;
}
```

H.2 IODELAY Table for High-Speed DAC Interface

Table H.1: DAC Signal Delay Values

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps$_{Sig}$**	**Taps$_{Norm}$**
FPGA_DAC_CLK_P	5016.47	837.750490	1	0	0	3
FPGA_DAC_CLK_N	5016.88	837.818960	0	-0.0684700	0	3
FPGA_DAC_SYNC_P	6315.62	1054.70854	0	-216.95805	-2	1
FPGA_DAC_SYNC_N	6361.72	1062.40724	0	-224.65675	-2	1
FPGA_DAC_DATA_P15	5762.20	962.287400	0	-124.53691	-1	2
FPGA_DAC_DATA_N15	5810.88	970.416960	0	-132.66647	-1	2
FPGA_DAC_DATA_P14	5834.68	974.391560	0	-136.64107	-1	2
FPGA_DAC_DATA_N14	5852.89	977.432630	0	-139.68214	-1	2
FPGA_DAC_DATA_P13	5537.71	924.797570	0	-87.047080	-1	2
FPGA_DAC_DATA_N13	5491.10	917.013700	0	-79.263210	-1	2
FPGA_DAC_DATA_P12	6196.47	1034.81049	0	-197.06000	-2	1
FPGA_DAC_DATA_N12	6163.08	1029.23436	0	-191.48387	-2	1
FPGA_DAC_DATA_P11	5641.90	942.197300	0	-104.44681	-1	2
FPGA_DAC_DATA_N11	5668.62	946.659540	0	-108.90905	-1	2
FPGA_DAC_DATA_P10	5670.41	946.958470	0	-109.20798	-1	2
FPGA_DAC_DATA_N10	5647.87	943.194290	0	-105.44380	-1	2
FPGA_DAC_DATA_P9	5608.93	936.691310	0	-98.940820	-1	2
FPGA_DAC_DATA_N9	5682.29	948.942430	0	-111.19194	-1	2
FPGA_DAC_DATA_P8	5530.95	923.668650	0	-85.918160	-1	2
FPGA_DAC_DATA_N8	5542.17	925.542390	0	-87.791900	-1	2
FPGA_DAC_DATA_P7	5735.98	957.908660	0	-120.15817	-1	2
FPGA_DAC_DATA_N7	5735.61	957.846870	0	-120.09638	-1	2
FPGA_DAC_DATA_P6	5582.68	932.307560	0	-94.557070	-1	2
FPGA_DAC_DATA_N6	5503.96	919.161320	0	-81.410830	-1	2
FPGA_DAC_DATA_P5	5505.65	919.443550	0	-81.693060	-1	2
FPGA_DAC_DATA_N5	5485.51	916.080170	0	-78.329680	-1	2
FPGA_DAC_DATA_P4	5829.65	973.551550	0	-135.80106	-1	2
FPGA_DAC_DATA_N4	5860.13	978.641710	0	-140.89122	-1	2
FPGA_DAC_DATA_P3	5696.69	951.347230	0	-113.59674	-1	2
FPGA_DAC_DATA_N3	5818.83	971.744610	0	-133.99412	-1	2
FPGA_DAC_DATA_P2	6465.15	1079.68005	0	-241.92956	-3	0
FPGA_DAC_DATA_N2	6486.70	1083.27890	0	-245.52841	-3	0
FPGA_DAC_DATA_P1	5715.30	954.455100	0	-116.70461	-1	2
FPGA_DAC_DATA_N1	5796.85	968.073950	0	-130.32346	-1	2
FPGA_DAC_DATA_P0	6597.08	1101.71236	0	-263.96187	-3	0
FPGA_DAC_DATA_N0	6580.01	1098.86167	0	-261.11118	-3	0

H.3 IODELAY Table for DDR2 SDRAM Interface

Table H.2: DDR2 SDRAM Signal Delay Values

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps$_{Sig}$**	**Taps$_{Norm}$**
Matched Length Group: #1						
FPGA_DDR2_SDRAM_CK_P0	4579.13	764.71471	1	0	0	0
FPGA_DDR2_SDRAM_CK_N0	4415.00	737.30500	0	27.4097100	0	0
FPGA_DDR2_SDRAM_CK_P1	4047.45	675.92415	0	88.7905600	1	1
FPGA_DDR2_SDRAM_CK_N1	3897.80	650.93260	0	113.782110	1	1
FPGA_DDR2_SDRAM_A0	2998.88	500.81296	0	263.901750	3	3
FPGA_DDR2_SDRAM_A1	2650.25	442.59175	0	322.122960	4	4
FPGA_DDR2_SDRAM_A2	2817.97	470.60099	0	294.113720	3	3
FPGA_DDR2_SDRAM_A3	2935.95	490.30365	0	274.411060	3	3
FPGA_DDR2_SDRAM_A4	2578.31	430.57777	0	334.136940	4	4
FPGA_DDR2_SDRAM_A5	3084.73	515.14991	0	249.564800	3	3
FPGA_DDR2_SDRAM_A6	2689.36	449.12312	0	315.591590	4	4
FPGA_DDR2_SDRAM_A7	2859.02	477.45634	0	287.258370	3	3
FPGA_DDR2_SDRAM_A8	3039.45	507.58815	0	257.126560	3	3
FPGA_DDR2_SDRAM_A9	3025.96	505.33532	0	259.379390	3	3
FPGA_DDR2_SDRAM_A10	3821.08	638.12036	0	126.594350	1	1
FPGA_DDR2_SDRAM_A11	2753.06	459.76102	0	304.953690	3	3
FPGA_DDR2_SDRAM_A12	3039.53	507.60151	0	257.113200	3	3
FPGA_DDR2_SDRAM_A13	4166.21	695.75707	0	68.9576400	0	0
FPGA_DDR2_SDRAM_BA0	4085.54	682.28518	0	82.4295300	1	1
FPGA_DDR2_SDRAM_BA1	3795.98	633.92866	0	130.786050	1	1
FPGA_DDR2_SDRAM_BA2	3092.63	516.46921	0	248.245500	3	3
FPGA_DDR2_SDRAM_CASN	4206.48	702.48216	0	62.2325500	0	0
FPGA_DDR2_SDRAM_CKE0	3972.22	663.36074	0	101.353970	1	1
FPGA_DDR2_SDRAM_CKE1	4002.67	668.44589	0	96.2688200	1	1
FPGA_DDR2_SDRAM_ODT0	3864.74	645.41158	0	119.303130	1	1
FPGA_DDR2_SDRAM_ODT1	4270.34	713.14678	0	51.5679300	0	0
FPGA_DDR2_SDRAM_RASN	4007.72	669.28924	0	95.4254700	1	1
FPGA_DDR2_SDRAM_SN0	4098.98	684.52966	0	80.1850500	1	1
FPGA_DDR2_SDRAM_SN1	4272.83	713.56261	0	51.1521000	0	0
FPGA_DDR2_SDRAM_WEN	4242.59	708.51253	0	56.2021800	0	0
Continued on Next Page. . .						

Signal Name	Length (Mils)	Length (ps)	Is Ref	Added Delay (ps)	Taps$_{Sig}$	Taps$_{Norm}$
Matched Length Group: #2						
FPGA_DDR2_SDRAM_DQS0	5013.03	837.17601	1	0	0	2
FPGA_DDR2_SDRAM_DQSN0	5120.19	855.07173	0	-17.895720	0	2
FPGA_DDR2_SDRAM_DM0	4957.16	827.84572	0	9.33029000	0	2
FPGA_DDR2_SDRAM_DQ0	5212.59	870.50253	0	-33.326520	0	2
FPGA_DDR2_SDRAM_DQ1	5287.59	883.02753	0	-45.851520	0	2
FPGA_DDR2_SDRAM_DQ2	5971.22	997.19374	0	-160.01773	-2	0
FPGA_DDR2_SDRAM_DQ3	5918.91	988.45797	0	-151.28196	-1	1
FPGA_DDR2_SDRAM_DQ4	4849.32	809.83644	0	27.3395700	0	2
FPGA_DDR2_SDRAM_DQ5	4963.70	828.93790	0	8.23811000	0	2
FPGA_DDR2_SDRAM_DQ6	5252.29	877.13243	0	-39.956420	0	2
FPGA_DDR2_SDRAM_DQ7	5525.77	922.80359	0	-85.627580	-1	1
Matched Length Group: #3						
FPGA_DDR2_SDRAM_DQS1	5087.82	849.66594	1	0	0	1
FPGA_DDR2_SDRAM_DQSN1	5041.40	841.91380	0	7.75214000	0	1
FPGA_DDR2_SDRAM_DM1	4747.95	792.90765	0	56.7582900	0	1
FPGA_DDR2_SDRAM_DQ8	5025.85	839.31695	0	10.3489900	0	1
FPGA_DDR2_SDRAM_DQ9	5105.20	852.56840	0	-2.9024600	0	1
FPGA_DDR2_SDRAM_DQ10	5191.29	866.94543	0	-17.279490	0	1
FPGA_DDR2_SDRAM_DQ11	5591.27	933.74209	0	-84.076150	-1	0
FPGA_DDR2_SDRAM_DQ12	4985.72	832.61524	0	17.0507000	0	1
FPGA_DDR2_SDRAM_DQ13	4870.79	813.42193	0	36.2440100	0	1
FPGA_DDR2_SDRAM_DQ14	5183.55	865.65285	0	-15.986910	0	1
FPGA_DDR2_SDRAM_DQ15	5180.95	865.21865	0	-15.552710	0	1
Matched Length Group: #4						
FPGA_DDR2_SDRAM_DQS2	4096.24	684.07208	1	0	0	1
FPGA_DDR2_SDRAM_DQSN2	4053.50	676.93450	0	7.13758000	0	1
FPGA_DDR2_SDRAM_DM2	3097.62	517.30254	0	166.769540	2	3
FPGA_DDR2_SDRAM_DQ16	3213.96	536.73132	0	147.340760	1	2
FPGA_DDR2_SDRAM_DQ17	4673.29	780.43943	0	-96.367350	-1	0
FPGA_DDR2_SDRAM_DQ18	3385.15	565.32005	0	118.752030	1	2
FPGA_DDR2_SDRAM_DQ19	3360.45	561.19515	0	122.876930	1	2
FPGA_DDR2_SDRAM_DQ20	3098.27	517.41109	0	166.660990	2	3
FPGA_DDR2_SDRAM_DQ21	3005.61	501.93687	0	182.135210	2	3
FPGA_DDR2_SDRAM_DQ22	2996.25	500.37375	0	183.698330	2	3
FPGA_DDR2_SDRAM_DQ23	2933.04	489.81768	0	194.254400	2	3
Continued on Next Page. . .						

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps$_{Sig}$**	**Taps$_{Norm}$**
Matched Length Group: #5						
FPGA_DDR2_SDRAM_DQS3	2983.24	498.20108	1	0	0	1
FPGA_DDR2_SDRAM_DQSN3	3023.39	504.90613	0	-6.7050500	0	1
FPGA_DDR2_SDRAM_DM3	3152.38	526.44746	0	-28.246380	0	1
FPGA_DDR2_SDRAM_DQ24	3390.75	566.25525	0	-68.054170	0	1
FPGA_DDR2_SDRAM_DQ25	3052.47	509.76249	0	-11.561410	0	1
FPGA_DDR2_SDRAM_DQ26	3622.58	604.97086	0	-106.76978	-1	0
FPGA_DDR2_SDRAM_DQ27	3219.61	537.67487	0	-39.473790	0	1
FPGA_DDR2_SDRAM_DQ28	2853.83	476.58961	0	21.6114700	0	1
FPGA_DDR2_SDRAM_DQ29	3098.43	517.43781	0	-19.236730	0	1
FPGA_DDR2_SDRAM_DQ30	2847.60	475.54920	0	22.6518800	0	1
FPGA_DDR2_SDRAM_DQ31	2916.43	487.04381	0	11.1572700	0	1
Matched Length Group: #6						
FPGA_DDR2_SDRAM_DQS4	3222.22	538.11074	1	0	0	2
FPGA_DDR2_SDRAM_DQSN4	3270.92	546.24364	0	-8.1329000	0	2
FPGA_DDR2_SDRAM_DM4	3182.03	531.39901	0	6.71173000	0	2
FPGA_DDR2_SDRAM_DQ32	3229.13	539.26471	0	-1.1539700	0	2
FPGA_DDR2_SDRAM_DQ33	3155.01	526.88667	0	11.2240700	0	2
FPGA_DDR2_SDRAM_DQ34	3607.15	602.39405	0	-64.283310	0	2
FPGA_DDR2_SDRAM_DQ35	4247.09	709.26403	0	-171.15329	-2	0
FPGA_DDR2_SDRAM_DQ36	2940.51	491.06517	0	47.0455700	0	2
FPGA_DDR2_SDRAM_DQ37	2957.14	493.84238	0	44.2683600	0	2
FPGA_DDR2_SDRAM_DQ38	3501.94	584.82398	0	-46.713240	0	2
FPGA_DDR2_SDRAM_DQ39	3857.18	644.14906	0	-106.03832	-1	1
Matched Length Group: #7						
FPGA_DDR2_SDRAM_DQS5	3522.42	588.24414	1	0	0	1
FPGA_DDR2_SDRAM_DQSN5	3405.06	568.64502	0	19.599120	0	1
FPGA_DDR2_SDRAM_DM5	3549.72	592.80324	0	-4.559100	0	1
FPGA_DDR2_SDRAM_DQ40	3996.72	667.45224	0	-79.20810	-1	0
FPGA_DDR2_SDRAM_DQ41	3555.78	593.81526	0	-5.571120	0	1
FPGA_DDR2_SDRAM_DQ42	3842.33	641.66911	0	-53.42497	0	1
FPGA_DDR2_SDRAM_DQ43	4029.84	672.98328	0	-84.73914	-1	0
FPGA_DDR2_SDRAM_DQ44	3311.40	553.00380	0	35.240340	0	1
FPGA_DDR2_SDRAM_DQ45	3135.08	523.55836	0	64.685780	0	1
FPGA_DDR2_SDRAM_DQ46	3605.04	602.04168	0	-13.79754	0	1
FPGA_DDR2_SDRAM_DQ47	3325.56	555.36852	0	32.875620	0	1

Continued on Next Page. . .

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps$_{Sig}$**	**Taps$_{Norm}$**
Matched Length Group: #8						
FPGA_DDR2_SDRAM_DQS6	3270.90	546.24030	1	0	0	0
FPGA_DDR2_SDRAM_DQSN6	3124.10	521.72470	0	24.515600	0	0
FPGA_DDR2_SDRAM_DM6	2854.88	476.76496	0	69.475340	0	0
FPGA_DDR2_SDRAM_DQ48	3316.24	553.81208	0	-7.571780	0	0
FPGA_DDR2_SDRAM_DQ49	3187.81	532.36427	0	13.876030	0	0
FPGA_DDR2_SDRAM_DQ50	3301.99	551.43233	0	-5.192030	0	0
FPGA_DDR2_SDRAM_DQ51	3414.35	570.19645	0	-23.95615	0	0
FPGA_DDR2_SDRAM_DQ52	2930.49	489.39183	0	56.848470	0	0
FPGA_DDR2_SDRAM_DQ53	3105.69	518.65023	0	27.590070	0	0
FPGA_DDR2_SDRAM_DQ54	2867.57	478.88419	0	67.356110	0	0
FPGA_DDR2_SDRAM_DQ55	2980.24	497.70008	0	48.540220	0	0
Matched Length Group: #9						
FPGA_DDR2_SDRAM_DQS7	3814.75	637.06325	1	0	0	0
FPGA_DDR2_SDRAM_DQSN7	3706.51	618.98717	0	18.076080	0	0
FPGA_DDR2_SDRAM_DM7	4052.38	676.74746	0	-39.68421	0	0
FPGA_DDR2_SDRAM_DQ56	3984.63	665.43321	0	-28.36996	0	0
FPGA_DDR2_SDRAM_DQ57	4208.12	702.75604	0	-65.69279	0	0
FPGA_DDR2_SDRAM_DQ58	4026.51	672.42717	0	-35.36392	0	0
FPGA_DDR2_SDRAM_DQ59	3832.92	640.09764	0	-3.034390	0	0
FPGA_DDR2_SDRAM_DQ60	3930.77	656.43859	0	-19.37534	0	0
FPGA_DDR2_SDRAM_DQ61	3748.47	625.99449	0	11.068760	0	0
FPGA_DDR2_SDRAM_DQ62	3768.33	629.31111	0	7.7521400	0	0
FPGA_DDR2_SDRAM_DQ63	3708.73	619.35791	0	17.705340	0	0

H.4 IODELAY Table for QDR-II SRAM Interface

Table H.3: QDR-II SRAM Signal Delay Values

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps$_{Sig}$**	**Taps$_{Norm}$**
Matched Length Group: #1						
FPGA_SRAM_K_CLK_P	2628.50	438.95950	1	0	0	4
FPGA_SRAM_K_CLK_N	2585.79	431.82693	0	7.13257000	0	4
FPGA_SRAM_WDATA35	3150.58	526.14686	0	-87.187360	-1	3
FPGA_SRAM_WDATA34	3050.75	509.47525	0	-70.515750	0	4
FPGA_SRAM_WDATA33	2802.70	468.05090	0	-29.091400	0	4
FPGA_SRAM_WDATA32	2562.85	427.99595	0	10.9635500	0	4
FPGA_SRAM_WDATA31	2276.92	380.24564	0	58.7138600	0	4
FPGA_SRAM_WDATA30	2199.48	367.31316	0	71.6463400	0	4
FPGA_SRAM_WDATA29	1792.90	299.41430	0	139.545200	1	5
FPGA_SRAM_WDATA28	1647.73	275.17091	0	163.788590	2	6
FPGA_SRAM_WDATA27	1730.36	288.97012	0	149.989380	1	5
FPGA_SRAM_WDATA26	3218.44	537.47948	0	-98.519980	-1	3
FPGA_SRAM_WDATA25	2953.25	493.19275	0	-54.233250	0	4
FPGA_SRAM_WDATA24	2792.77	466.39259	0	-27.433090	0	4
FPGA_SRAM_WDATA23	2563.12	428.04104	0	10.9184600	0	4
FPGA_SRAM_WDATA22	1994.36	333.05812	0	105.901380	1	5
FPGA_SRAM_WDATA21	1963.23	327.85941	0	111.100090	1	5
FPGA_SRAM_WDATA20	1709.22	285.43974	0	153.519760	1	5
FPGA_SRAM_WDATA19	1825.79	304.90693	0	134.052570	1	5
FPGA_SRAM_WDATA18	2008.51	335.42117	0	103.538330	1	5
FPGA_SRAM_WDATA17	2557.84	427.15928	0	11.8002200	0	4
FPGA_SRAM_WDATA16	2591.71	432.81557	0	6.14393000	0	4
FPGA_SRAM_WDATA15	2650.50	442.63350	0	-3.6740000	0	4
FPGA_SRAM_WDATA14	2979.66	497.60322	0	-58.643720	0	4
FPGA_SRAM_WDATA13	3128.26	522.41942	0	-83.459920	-1	3
FPGA_SRAM_WDATA12	3299.14	550.95638	0	-111.99688	-1	3
FPGA_SRAM_WDATA11	3631.04	606.38368	0	-167.42418	-2	2
FPGA_SRAM_WDATA10	3844.59	642.04653	0	-203.08703	-2	2
FPGA_SRAM_WDATA9	3883.33	648.51611	0	-209.55661	-2	2
FPGA_SRAM_WDATA8	2600.65	434.30855	0	4.65095000	0	4
FPGA_SRAM_WDATA7	2737.99	457.24433	0	-18.284830	0	4

Continued on Next Page...

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps$_{Sig}$**	**Taps$_{Norm}$**
FPGA_SRAM_WDATA6	2777.23	463.79741	0	-24.837910	0	4
FPGA_SRAM_WDATA5	3063.23	511.55941	0	-72.599910	0	4
FPGA_SRAM_WDATA4	3154.47	526.79649	0	-87.836990	-1	3
FPGA_SRAM_WDATA3	3295.01	550.26667	0	-111.30717	-1	3
FPGA_SRAM_WDATA2	3659.74	611.17658	0	-172.21708	-2	2
FPGA_SRAM_WDATA1	3626.80	605.67560	0	-166.71610	-2	2
FPGA_SRAM_WDATA0	3958.94	661.14298	0	-222.18348	-2	2
FPGA_SRAM_ADDR17	4604.33	768.92311	0	-329.96361	-4	0
FPGA_SRAM_ADDR16	3819.12	637.79304	0	-198.83354	-2	2
FPGA_SRAM_ADDR15	3373.71	563.40957	0	-124.45007	-1	3
FPGA_SRAM_ADDR14	2694.44	449.97148	0	-11.011980	0	4
FPGA_SRAM_ADDR13	2530.12	422.53004	0	16.4294600	0	4
FPGA_SRAM_ADDR12	2476.10	413.50870	0	25.4508000	0	4
FPGA_SRAM_ADDR11	3864.65	645.39655	0	-206.43705	-2	2
FPGA_SRAM_ADDR10	3629.69	606.15823	0	-167.19873	-2	2
FPGA_SRAM_ADDR9	3037.20	507.21240	0	-68.252900	0	4
FPGA_SRAM_ADDR8	2574.04	429.86468	0	9.09482000	0	4
FPGA_SRAM_ADDR7	3324.62	555.21154	0	-116.25204	-1	3
FPGA_SRAM_ADDR6	3145.99	525.38033	0	-86.420830	-1	3
FPGA_SRAM_ADDR5	2671.76	446.18392	0	-7.2244200	0	4
FPGA_SRAM_ADDR4	3057.69	510.63423	0	-71.674730	0	4
FPGA_SRAM_ADDR3	3591.46	599.77382	0	-160.81432	-2	2
FPGA_SRAM_ADDR2	2905.65	485.24355	0	-46.284050	0	4
FPGA_SRAM_ADDR1	2879.98	480.95666	0	-41.997160	0	4
FPGA_SRAM_ADDR0	2616.20	436.90540	0	2.05410000	0	4
FPGA_SRAM_RDN	2715.00	453.40500	0	-14.445500	0	4
FPGA_SRAM_WRN	2279.91	380.74497	0	58.2145300	0	4
FPGA_SRAM_BWN3	2604.82	435.00494	0	3.95456000	0	4
FPGA_SRAM_BWN2	2445.94	408.47198	0	30.4875200	0	4
FPGA_SRAM_BWN1	2293.23	382.96941	0	55.9900900	0	4
FPGA_SRAM_BWN0	3213.01	536.57267	0	-97.613170	-1	3
FPGA_SRAM_DLL_OFFN	3008.41	502.40447	0	-63.444970	0	4

Continued on Next Page...

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps$_{Sig}$**	**Taps$_{Norm}$**
Matched Length Group: #2						
FPGA_SRAM_CQ_CLK_P	1976.35	330.05045	1	0	0	4
FPGA_SRAM_RDATA17	2060.67	344.13189	0	-14.081440	0	4
FPGA_SRAM_RDATA16	2361.19	394.31873	0	-64.268280	0	4
FPGA_SRAM_RDATA15	2555.24	426.72508	0	-96.674630	-1	3
FPGA_SRAM_RDATA14	2643.55	441.47285	0	-111.42240	-1	3
FPGA_SRAM_RDATA13	2783.77	464.88959	0	-134.83914	-1	3
FPGA_SRAM_RDATA12	3204.88	535.21496	0	-205.16451	-2	2
FPGA_SRAM_RDATA11	3450.88	576.29696	0	-246.24651	-3	1
FPGA_SRAM_RDATA10	3748.97	626.07799	0	-296.02754	-3	1
FPGA_SRAM_RDATA9	4128.87	689.52129	0	-359.47084	-4	0
FPGA_SRAM_RDATA8	2206.29	368.45043	0	-38.399980	0	4
FPGA_SRAM_RDATA7	2387.63	398.73421	0	-68.683760	0	4
FPGA_SRAM_RDATA6	2458.86	410.62962	0	-80.579170	-1	3
FPGA_SRAM_RDATA5	2562.97	428.01599	0	-97.965540	-1	3
FPGA_SRAM_RDATA4	2813.14	469.79438	0	-139.74393	-1	3
FPGA_SRAM_RDATA3	3162.41	528.12247	0	-198.07202	-2	2
FPGA_SRAM_RDATA2	3312.99	553.26933	0	-223.21888	-2	2
FPGA_SRAM_RDATA1	3670.76	613.01692	0	-282.96647	-3	1
FPGA_SRAM_RDATA0	3793.17	633.45939	0	-303.40894	-3	1
Matched Length Group: #3						
FPGA_SRAM_CQ_CLK_N	1647.00	275.04900	1	0	0	4
FPGA_SRAM_RDATA35	3261.71	544.70557	0	-269.65657	-3	1
FPGA_SRAM_RDATA34	3086.63	515.46721	0	-240.41821	-3	1
FPGA_SRAM_RDATA33	2746.72	458.70224	0	-183.65324	-2	2
FPGA_SRAM_RDATA32	2473.22	413.02774	0	-137.97874	-1	3
FPGA_SRAM_RDATA31	2417.51	403.72417	0	-128.67517	-1	3
FPGA_SRAM_RDATA30	1839.46	307.18982	0	-32.140820	0	4
FPGA_SRAM_RDATA29	1790.92	299.08364	0	-24.034640	0	4
FPGA_SRAM_RDATA28	1675.18	279.75506	0	-4.7060600	0	4
FPGA_SRAM_RDATA27	1635.68	273.15856	0	1.89044000	0	4
FPGA_SRAM_RDATA26	3518.30	587.55610	0	-312.50710	-4	0
FPGA_SRAM_RDATA25	3180.23	531.09841	0	-256.04941	-3	1
FPGA_SRAM_RDATA24	2973.35	496.54945	0	-221.50045	-2	2
FPGA_SRAM_RDATA23	2635.30	440.09510	0	-165.04610	-2	2
FPGA_SRAM_RDATA22	2221.83	371.04561	0	-95.996610	-1	3

Continued on Next Page...

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps$_{Sig}$**	**Taps$_{Norm}$**
FPGA_SRAM_RDATA21	2003.05	334.50935	0	-59.460350	0	4
FPGA_SRAM_RDATA20	1848.85	308.75795	0	-33.708950	0	4
FPGA_SRAM_RDATA19	1675.54	279.81518	0	-4.7661800	0	4
FPGA_SRAM_RDATA18	1688.03	281.90101	0	-6.8520100	0	4

H.5 IODELAY Table for AsAP #1 Interface

Table H.4: AsAP #1 Signal Delay Values

Signal Name	Length (Mils)	Length (ps)	Is Ref	Added Delay (ps)	Taps$_{Sig}$	Taps$_{Norm}$
Matched Length Group: #1						
FPGA_ASAP1_CLK_OUT	3888.03	649.30101	1	0	0	1
FPGA_ASAP1_VLD_OUT	4049.70	676.29990	0	-26.998890	0	1
FPGA_ASAP1_REQ_OUT	4052.61	676.78587	0	-27.484860	0	1
FPGA_ASAP1_DOUT15	4144.88	692.19496	0	-42.893950	0	1
FPGA_ASAP1_DOUT14	4209.27	702.94809	0	-53.647080	0	1
FPGA_ASAP1_DOUT13	4166.49	695.80383	0	-46.502820	0	1
FPGA_ASAP1_DOUT12	3939.19	657.84473	0	-8.5437200	0	1
FPGA_ASAP1_DOUT11	4805.99	802.60033	0	-153.29932	-1	0
FPGA_ASAP1_DOUT10	4815.56	804.19852	0	-154.89751	-1	0
FPGA_ASAP1_DOUT9	3641.68	608.16056	0	41.1404500	0	1
FPGA_ASAP1_DOUT8	3701.51	618.15217	0	31.1488400	0	1
FPGA_ASAP1_DOUT7	4163.47	695.29949	0	-45.998480	0	1
FPGA_ASAP1_DOUT6	4227.27	705.95409	0	-56.653080	0	1
FPGA_ASAP1_DOUT5	3559.15	594.37805	0	54.9229600	0	1
FPGA_ASAP1_DOUT4	3785.44	632.16848	0	17.1325300	0	1
FPGA_ASAP1_DOUT3	3856.54	644.04218	0	5.25882999	0	1
FPGA_ASAP1_DOUT2	4078.00	681.02600	0	-31.724990	0	1
FPGA_ASAP1_DOUT1	3278.86	547.56962	0	101.731390	1	2
FPGA_ASAP1_DOUT0	3288.72	549.21624	0	100.084770	1	2
Matched Length Group: #2						
FPGA_ASAP1_CLK_IN	2656.68	443.66556	1	0	0	2
FPGA_ASAP1_VLD_IN	2137.68	356.99256	0	86.6730000	1	3
FPGA_ASAP1_REQ_IN	2767.24	462.12908	0	-18.463520	0	2
FPGA_ASAP1_DIN15	2695.18	450.09506	0	-6.4295000	0	2
FPGA_ASAP1_DIN14	2657.36	443.77912	0	-0.1135600	0	2
FPGA_ASAP1_DIN13	2342.88	391.26096	0	52.4045999	0	2
FPGA_ASAP1_DIN12	2361.73	394.40891	0	49.2566499	0	2
FPGA_ASAP1_DIN11	2188.27	365.44109	0	78.2244699	1	3
FPGA_ASAP1_DIN10	2183.37	364.62279	0	79.0427700	1	3
FPGA_ASAP1_DIN9	2551.92	426.17064	0	17.4949199	0	2
FPGA_ASAP1_DIN8	2961.36	494.54712	0	-50.881560	0	2
FPGA_ASAP1_DIN7	2636.86	440.35562	0	3.30993999	0	2
FPGA_ASAP1_DIN6	2601.93	434.52231	0	9.14324999	0	2
FPGA_ASAP1_DIN5	3023.82	504.97794	0	-61.312380	0	2

Continued on Next Page...

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps**$_{Sig}$	**Taps**$_{Norm}$
FPGA_ASAP1_DIN4	3355.69	560.40023	0	-116.73467	-1	1
FPGA_ASAP1_DIN3	3219.09	537.58803	0	-93.922470	-1	1
FPGA_ASAP1_DIN2	3142.93	524.86931	0	-81.203750	-1	1
FPGA_ASAP1_DIN1	3586.95	599.02065	0	-155.35509	-1	1
FPGA_ASAP1_DIN0	3717.09	620.75403	0	-177.08847	-2	0

H.6 IODELAY Table for AsAP #2 Interface

Table H.5: AsAP #2 Signal Delay Values

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps$_{Sig}$**	**Taps$_{Norm}$**
Matched Length Group: #1						
FPGA_ASAP2_CLK_OUT	5402.93	902.289310	1	0	0	1
FPGA_ASAP2_VLD_OUT	5298.40	884.832800	0	17.4565100	0	1
FPGA_ASAP2_REQ_OUT	5138.41	858.114470	0	44.1748400	0	1
FPGA_ASAP2_DOUT15	5198.95	868.224650	0	34.0646600	0	1
FPGA_ASAP2_DOUT14	5282.13	882.115710	0	20.1736000	0	1
FPGA_ASAP2_DOUT13	5487.81	916.464270	0	-14.174960	0	1
FPGA_ASAP2_DOUT12	5680.78	948.690260	0	-46.400950	0	1
FPGA_ASAP2_DOUT11	5874.91	981.109970	0	-78.820660	-1	0
FPGA_ASAP2_DOUT10	5837.32	974.832440	0	-72.543130	0	1
FPGA_ASAP2_DOUT9	4944.92	825.801640	0	76.4876700	0	1
FPGA_ASAP2_DOUT8	4669.31	779.774770	0	122.514540	1	2
FPGA_ASAP2_DOUT7	5679.63	948.498210	0	-46.208900	0	1
FPGA_ASAP2_DOUT6	5282.01	882.095670	0	20.1936400	0	1
FPGA_ASAP2_DOUT5	4691.11	783.415370	0	118.873940	1	2
FPGA_ASAP2_DOUT4	4599.08	768.046360	0	134.242950	1	2
FPGA_ASAP2_DOUT3	5928.96	990.136320	0	-87.847010	-1	0
FPGA_ASAP2_DOUT2	5494.71	917.616570	0	-15.327260	0	1
FPGA_ASAP2_DOUT1	4679.77	781.521590	0	120.767720	1	2
FPGA_ASAP2_DOUT0	4556.70	760.968900	0	141.320410	1	2
Matched Length Group: #2						
FPGA_ASAP2_CLK_IN	5126.14	856.065380	1	0	0	2
FPGA_ASAP2_VLD_IN	4551.79	760.148930	0	95.9164500	1	3
FPGA_ASAP2_REQ_IN	4173.06	696.901020	0	159.164360	2	4
FPGA_ASAP2_DIN15	4780.52	798.346840	0	57.7185400	0	2
FPGA_ASAP2_DIN14	4717.12	787.759040	0	68.3063400	0	2
FPGA_ASAP2_DIN13	4786.03	799.267010	0	56.7983700	0	2
FPGA_ASAP2_DIN12	4730.52	789.996840	0	66.0685400	0	2
FPGA_ASAP2_DIN11	4767.18	796.119060	0	59.9463200	0	2
FPGA_ASAP2_DIN10	4904.30	819.018100	0	37.0472800	0	2
FPGA_ASAP2_DIN9	5335.07	890.956690	0	-34.891310	0	2
FPGA_ASAP2_DIN8	5698.97	951.727990	0	-95.662610	-1	1
FPGA_ASAP2_DIN7	5019.04	838.179680	0	17.8857000	0	2
FPGA_ASAP2_DIN6	4888.85	816.437950	0	39.6274300	0	2
FPGA_ASAP2_DIN5	5732.70	957.360900	0	-101.29552	-1	1

Continued on Next Page...

Signal Name	**Length (Mils)**	**Length (ps)**	**Is Ref**	**Added Delay (ps)**	**Taps$_{Sig}$**	**Taps$_{Norm}$**
FPGA_ASAP2_DIN4	5651.11	943.735370	0	-87.669990	-1	1
FPGA_ASAP2_DIN3	5832.44	974.017480	0	-117.95210	-1	1
FPGA_ASAP2_DIN2	5564.81	929.323270	0	-73.257890	0	2
FPGA_ASAP2_DIN1	6143.62	1025.98454	0	-169.91916	-2	0
FPGA_ASAP2_DIN0	6081.24	1015.56708	0	-159.50170	-2	0

Appendix I

Data Path FPGA Register Definitions

The Data Path FPGA on the measurement board is controlled by the Control FPGA via a SPI interface. The Data Path FPGA is essentially a large register file made up of seven base address ranges, shown in Table I.1. The following sections describe the registers in the Data Path FPGA, including what data they are storing and how the data is used.

Table I.1: Hardware Register Addresses

Hardware Register Addresses			
Section	**Sub-System**	**Hardware Registers**	**Dev Address**
I.1	Measurement Board	System Registers	0x0000
I.2	Measurement Board	IODELAY Control Registers	0x0100
I.3	Measurement Board	RAM/User Pattern Control Registers	0x0300
I.4	Measurement Board	Auxiliary Input Control Registers	0x0400
I.5	Measurement Board	Trigger Control Registers	0x0500
I.6	Measurement Board	ADC Control Registers	0x0700
I.7	Measurement Board	AsAP Control Registers	0x0800

Hardware Function:	0x0
Device Address:	0x0000
Read/Write:	Write Only

I.1 Base Address: 0x0000, System Registers

I.1.1 Safe State, Address: 0x0000

Writing anything to this register causes all Data Path FPGA registers to go into their reset state. This is the same state that the registers are in when the measurement board assembly first powers up.

I.1.2 Instrument Reset Register, Address: 0x0002

Hardware Function:	0x0
Device Address:	0x0002
Read/Write:	Read and Write

This is the Instrument Reset register. It is used to reset portions of the Data Path FPGA that may need to be reset during the operation of the instrument. Table I.2 shows the register bit assignments and the default value. If a bit defaults to a 1, then that bit is active low. If a bit defaults to a 0, then that bit is active high.

Table I.2: Instrument Reset Register

BIT	**NAME**	**RESET**
31:8	RSVD	0
7	WAVE_RST	1
6	ADC_FIFO_RST	0
5	RSVD	0
4	RAM_DAC_RST	1
3	SYS_DCM	0
2	SRAM_CTRL	1
1	XIL_SRAM_CTRL	0
0	SRAM_PLL	0

Table I.3: Instrument Reset State

Bit	Reset	Description
7	WAVE_RST	Active Low
6	ADC_FIFO_RST	Active High
5	RSVD	Active High
4	RAM_DAC_RST	Active Low
3	SYS_DCM	Active High
2	SRAM_CTRL	Active Low
1	XIL_SRAM_CTRL	Active High
0	SRAM_PLL	Active High

I.1.3 Interrupt Register, Address: 0x0004

Hardware Function:	0x0
Device Address:	0x0004
Read/Write:	Read Only

The Interrupt Register is used to tell the status of the interrupts of the assembly currently being addressed. It is a read only register.

The bits are assembly specific. A pending interrupt is indicated by the appropriate bit in the Interrupt Register being set. All interrupt bits are latched except for Bit 0. Bit 0 of the Interrupt Register is reserved to tell the status of the Interrupt Service Request (SRQn) output of the assembly. When Bit 0 is enabled, Bit 0 is the OR of the interrupt bits 1 through 31.

Table I.4: Interrupt Register

BIT	NAME	RESET
31:7	RSVD	0
6	SRAM_CAL_DONE_ON	0
5	SRAM_CAL_DONE_OFF	0
4	SYS_DCM_LOCKED_ON	0
3	SYS_DCM_LOCKED_OFF	0
2	SRAM_DCM_LOCKED_ON	0
1	SRAM_DCM_LOCKED_OFF	0
0	SRQn	1

SRQn - Service Request Bit

0 = SRQn is NOT being driven low.

1 = SRQn is being driven low.

SRAM_DCM_LOCKED_OFF

0 = FALSE

1 = TRUE

SRAM_DCM_LOCKED_ON

0 = FALSE

1 = TRUE

SYS_DCM_LOCKED_OFF

0 = FALSE

1 = TRUE

SYS_DCM_LOCKED_ON

0 = FALSE

1 = TRUE

SRAM_CAL_DONE_OFF

0 = FALSE

1 = TRUE

SRAM_CAL_DONE_ON

0 = FALSE

1 = TRUE

I.1.4 Interrupt Enable Register, Address: 0x0006

Hardware Function:	0x0
Device Address:	0x0006
Read/Write:	Read and Write

The Interrupt Enable Register is used to enable/disable the individual interrupts of the assembly currently being addressed. It is a write only register.

Each bit of the Interrupt Enable Register has a one to one correspondence with the bits in the Interrupt Register. Bit 0 of the Interrupt Enable Register is reserved to enable/disable the SRQn output of the assembly.

An interrupt is enabled by setting its bit in the Interrupt Enable Register. The enabled interrupts will always show their status in the Interrupt Register. Disabled interrupts do not cause interrupts but do show the latched status of the associated bit in the Interrupt Register. The latched interrupt status bits will still show their status even if the SRQ_EN bit is disabled. This allows the interrupts to be polled without causing the system controller to be interrupted.

Table I.5: Interrupt Enable Register

BIT	NAME	RESET
31:7	RSVD	0
6	SRAM_CAL_DONE_ON	0
5	SRAM_CAL_DONE_OFF	0
4	SYS_DCM_LOCKED_ON	0
3	SYS_DCM_LOCKED_OFF	0
2	SRAM_DCM_LOCKED_ON	0
1	SRAM_DCM_LOCKED_OFF	0
0	SRQ_EN	1

SRQn - Service Request Bit

0 = SRQn output is disabled.

1 = SRQn output is enabled.

SRAM_DCM_LOCKED_OFF

0 = disabled

1 = enabled

SRAM_DCM_LOCKED_ON

0 = disabled

1 = enabled

SYS_DCM_LOCKED_OFF

0 = disabled

1 = enabled

SYS_DCM_LOCKED_ON

0 = disabled

1 = enabled

SRAM_CAL_DONE_OFF

0 = disabled

1 = enabled

SRAM_CAL_DONE_ON

0 = disabled

1 = enabled

I.1.5 Interrupt Clear Register, Address: 0x0008

Hardware Function:	0x0
Device Address:	0x0008
Read/Write:	Write Only

The Interrupt Clear Register is used to clear the individual interrupts of the assembly currently being addressed. It is a write only register.

Each bit of the Interrupt Clear Register has a one to one correspondence with the bits in the Interrupt Register. Bit 0 of the Interrupt Clear Register does nothing since the SRQ interrupt is not latched. When the condition causing the interrupt is fixed, the CPU should clear the latched interrupt bit by writing a one to the corresponding bit in the Interrupt Clear Register.

Table I.6: Interrupt Clear Register

BIT	NAME	RESET
31:7	RSVD	0
6	SRAM_CAL_DONE_ON	0
5	SRAM_CAL_DONE_OFF	0
4	SYS_DCM_LOCKED_ON	0
3	SYS_DCM_LOCKED_OFF	0
2	SRAM_DCM_LOCKED_ON	0
1	SRAM_DCM_LOCKED_OFF	0
0	No Effect	1

SRAM_DCM_LOCKED_OFF

0 = no change

1 = clear interrupt

SRAM_DCM_LOCKED_ON

0 = no change

1 = clear interrupt

SYS_DCM_LOCKED_OFF

0 = no change

1 = clear interrupt

SYS_DCM_LOCKED_ON

0 = no change

1 = clear interrupt

SRAM_CAL_DONE_OFF

0 = no change

1 = clear interrupt

SRAM_CAL_DONE_ON

0 = no change

1 = clear interrupt

I.1.6 Status Register, Address: 0x000A

Hardware Function:	0x0
Device Address:	0x000A
Read/Write:	Read Only

Table I.7: Status Register

BIT	NAME	RESET
31:7	RSVD	0
6	SRAM_CAL_DONE_ON	0
5	SRAM_CAL_DONE_OFF	0
4	SYS_DCM_LOCKED_ON	0
3	SYS_DCM_LOCKED_OFF	0
2	SRAM_DCM_LOCKED_ON	0
1	SRAM_DCM_LOCKED_OFF	0
0	No Effect	1

SRAM_DCM_LOCKED_OFF

0 = no change

1 = interrupt

SRAM_DCM_LOCKED_ON

0 = no change

1 = interrupt

SYS_DCM_LOCKED_OFF

0 = no change

1 = clear interrupt

SYS_DCM_LOCKED_ON

0 = no change

1 = clear interrupt

SRAM_CAL_DONE_OFF

0 = no change

1 = clear interrupt

SRAM_CAL_DONE_ON

0 = no change

1 = clear interrupt

I.1.7 Debug LED Register, Address: 0x000C

Hardware Function:	0x0
Device Address:	0x000C
Read/Write:	Read and Write

This register is used to control the debug LEDs. Writing a 0 to this register will turn an LED on, and writing a 1 will turn an LED off. This register is used only for debugging purposes.

Table I.8: Debug LED Register

BIT	NAME	RESET
31:4	RSVD	0
3:0	DEBUG_LEDS[3:0]	0x99

I.1.8 FPGA Date Code Register, Address: 0x000E

Hardware Function:	0x0
Device Address:	0x000E
Read/Write:	Read Only

This is the FPGA Date Code register. It is used to determine what date and time the FPGA was built. The date code is essentially the UNIX time format.

Table I.9: FPGA Date Code Register

BIT	**NAME**	**RESET**
31:0	DATECODE[31:0]	0x00000000

I.1.9 FPGA Version Register, Address: 0x0010

Hardware Function:	0x0
Device Address:	0x0010
Read/Write:	Read Only

This register is used to determine the version of the FPGA being built. This register will be updated when major or minor changes have been applied to the FPGA. An example version for a first release in integer is 1.0.1.0. and hexadecimal is 0x01.0x01.0x0000.

Table I.10: FPGA Version Register

BIT	**NAME**	**RESET**
31:24	REV_MAJOR[31:24]	0x0100
23:16	REV_MINOR[23:16]	0x0001
15:0	REV_DEV[15:0]	0x0001

I.2 Base Address: 0x0100, IODELAY Control Registers

I.2.1 IDELAY_CTRL Reset Register, Address: 0x0100

Hardware Function:	0x0
Device Address:	0x0100
Read/Write:	Read and Write

The IDELAY_CTRL Reset register is used to reset the IDELAY_CTRL blocks used in the Data Path FPGA. The IODELAY blocks will not function until the IDELAY_CTRL blocks are reset. A reset is performed by setting all bits in this register high-then-low.

Table I.11: IDELAY_CTRL Reset Register

BIT	NAME	RESET
31:6	RSVD	0
5	RESERVED	0
4	RESERVED	0
3	RESERVED	0
2	RESERVED	0
1	ADC_IDLY_RST_CTRL	0
0	DAC_IDLY_RST_CTRL	0

I.2.2 IDELAY_CTRL Status Register, Address: 0x0102

Hardware Function:	0x0
Device Address:	0x0102
Read/Write:	Read Only

The IDELAY_CTRL Status register is used to report the readiness of the IDELAY_CTRL block. The ready signals are active high.

Table I.12: IDELAY_CTRL Status Register

BIT	NAME	RESET
31:4	RSVD	0
3	ADC_IDELAY_RDY	0
2	SRAM_Q_IDELAY_RDY	0
1	SRAM_D_IDELAY_RDY	0
0	SRAM_CTRL_IDELAY_RDY	0

I.3 Base Address: 0x0300, RAM/User Pattern Control Registers

I.3.1 Playback RAM Write Control Register, Address: 0x0300

Hardware Function:	0x0
Device Address:	0x0300
Read/Write:	Read and Write

This is the Playback RAM Write Control register. This register initiates an SRAM 4-word burst write. The write data resides in registers 0x0306 to 0x030C, and the write address resides in register 0x0304. When performing an SRAM 4-word burst write, the CPU must toggle this bit high then low. The time between the SRAM Write Enable bit being set high then low is inconsequential, because a one-shot circuit inside the FPGA creates a single pulse one clock period wide when it detects a 0→1 transition on its input.

Table I.13: Playback RAM Write Control Register

BIT	NAME	RESET
31:1	RSVD	0
0	PLAY_SRAM_WR_EN	0

I.3.2 Playback RAM Write Status Register, Address: 0x0302

Hardware Function:	0x0
Device Address:	0x0302
Read/Write:	Read Only

This is the Playback RAM Write Status register. The SRAM_CAL_DONE bit indicates when the Xilinx QDR-II SRAM Controller has completed its calibration routine. The Control FPGA must wait until the SRAM_CAL_DONE bit goes high before it performs any SRAM related operations, especially if the PPG is currently in SRAM Pattern mode. The SRAM_WR_DONE bit indicates when a single 128-bit packet has been successfully written to the QDR-II SRAM. The Control FPGA must wait until the SRAM_WR_DONE bit goes high before initiating a new QDR-II SRAM Write Sequence.

Table I.14: Playback RAM Write Control Register

BIT	NAME	RESET
31:2	RSVD	0
1	PLAY_SRAM_WR_DONE	0
0	SRAM_CAL_DONE	0

I.3.3 Playback RAM Write Address Register, Address: 0x0304

Hardware Function:	0x0
Device Address:	0x0304
Read/Write:	Read and Write

These are the Playback RAM Write Address registers. These registers comprise a register that is 18-bits wide, and addresses the 4-word burst write transaction.

Table I.15: Playback RAM Write Address Register

BIT	NAME	RESET
31:18	RSVD	0
17:0	PLAY_WR_ADDR[17:0]	0x00000

I.3.4 Playback RAM Write Data 0 Register, Address: 0x0306

Hardware Function:	0x0
Device Address:	0x0306
Read/Write:	Read and Write

These are the Playback RAM Write Data 0 registers. These registers comprise a register that is 32-bits wide, and make up one word of the 4-word burst write transaction. This 32-bit packet is the LSB Word in the 128-bit packet.

Table I.16: Playback RAM Write Data 0 Register

BIT	NAME	RESET
31:0	PLAY_WR0_DATA[31:0]	0x00000000

I.3.5 Playback RAM Write Data 1 Register, Address: 0x0308

Hardware Function:	0x0
Device Address:	0x0308
Read/Write:	Read and Write

These are the Playback RAM Write Data 1 registers. These registers comprise a register that is 32-bits wide, and make up one word of the 4-word burst write transaction.

Table I.17: Playback RAM Write Data 1 Register

BIT	**NAME**	**RESET**
31:0	PLAY_WR1_DATA[31:0]	0x00000000

I.3.6 Playback RAM Write Data 2 Register, Address: 0x030A

Hardware Function:	0x0
Device Address:	0x030A
Read/Write:	Read and Write

These are the Playback RAM Write Data 2 registers. These registers comprise a register that is 32-bits wide, and make up one word of the 4-word burst write transaction.

Table I.18: Playback RAM Write Data 2 Register

BIT	**NAME**	**RESET**
31:0	PLAY_WR2_DATA[31:0]	0x00000000

I.3.7 Playback RAM Write Data 3 Register, Address: 0x030C

Hardware Function:	0x0
Device Address:	0x030C
Read/Write:	Read and Write

These are the Playback RAM Write Data 3 registers. These registers comprise a register that is 32-bits wide, and make up one word of the 4-word burst write transaction. This 32-bit packet is the MSB Word in the 128-bit packet.

Table I.19: Playback RAM Write Data 3 Register

BIT	NAME	RESET
31:0	PLAY_WR3_DATA[31:0]	0x00000000

I.3.8 Playback RAM Read Control Register, Address: 0x030E

Hardware Function:	0x0
Device Address:	0x030E
Read/Write:	Read and Write

This is the Playback RAM Read Control register. This register initiates an SRAM read. The read address resides in registers 0x0312 to 0x0318. When performing an SRAM Read initiation, the CPU must toggle this bit high then low. The time between the SRAM Read Enable bit being set high then low is inconsequential, because a one-shot circuit inside the FPGA creates a single pulse one clock period wide when it detects a $0 \rightarrow 1$ transition on its input.

Table I.20: Playback RAM Read Control Register

BIT	NAME	RESET
31:1	RSVD	0
0	PLAY_SRAM_RD_EN	0

I.3.9 Playback RAM Read Status Register, Address: 0x0310

Hardware Function:	0x0
Device Address:	0x0310
Read/Write:	Read Only

This is the Playback RAM Read Status register. This register is a place holder for any important information related to SRAM read operations.

Table I.21: Playback RAM Read Control Register

BIT	NAME	RESET
31:2	RSVD	0
0	PLAY_SRAM_RD_DONE	0
0	SRAM_CAL_DONE	0

I.3.10 Playback RAM Read Start Address Register, Address: 0x0312

Hardware Function:	0x0
Device Address:	0x0312
Read/Write:	Read and Write

This is the Playback RAM Read Start Address register. This register indicates the start address for the pattern. It is 18-bits wide, and addresses the 4-word burst read transaction.

Table I.22: Playback RAM Read Start Address Register

BIT	NAME	RESET
31:18	RSVD	0
17:0	PLAY_RD_START_ADDR[17:0]	0

I.3.11 Playback RAM Read Stop Address Register, Address: 0x0314

Hardware Function:	0x0
Device Address:	0x0314
Read/Write:	Read and Write

This is the Playback RAM Read Stop Address register. This register indicates the stop address for the pattern. It is 18-bits wide, and addresses the 4-word burst read transaction.

Table I.23: RAM Read Stop Address (Pattern A) Register

BIT	**NAME**	**RESET**
31:18	RSVD	0
17:0	PLAY_RD_STOP_ADDR[17:0]	0

I.3.12 Playback RAM Read Increment Address Register, Address: 0x0316

Hardware Function:	0x0
Device Address:	0x0316
Read/Write:	Read and Write

This is the Playback RAM Read Increment Address register. This register indicates the value that the SRAM address is incremented when streaming data from SRAM. It is 18-bits wide.

Table I.24: Playback RAM Read Increment Address Register

BIT	**NAME**	**RESET**
31:18	RSVD	0
17:0	PLAY_RD_INC_ADDR[17:0]	0x00001

I.3.13 Playback RAM Read Maximum Address Register, Address: 0x0318

Hardware Function:	0x0
Device Address:	0x0318
Read/Write:	Read and Write

This is the Playback RAM Read Maximum Address register. This register indicates the maximum initial address, which is used to pre-load the SRAM stream address counter. It is 18-bits wide.

Table I.25: Playback RAM Read Maximum Address Register

BIT	NAME	RESET
31:18	RSVD	0
17:0	PLAY_RD_MAX_ADDR[17:0]	0x3FFFF

I.3.14 Xilinx SRAM Controller Done Status Register, Address: 0x031A

Hardware Function:	0x0
Device Address:	0x031A
Read/Write:	Read Only

This is the Xilinx SRAM Controller Done Status register. This register is used to troubleshoot the calibration routine in the Xilinx SRAM Controller. The calibration routine is made up of several stages. Each stage of the routine is complete when its status bit goes high.

Table I.26: Xilinx SRAM Controller Done Status Register

BIT	NAME	RESET
31:8	RSVD	0
7	INIT_COUNT_DONE	0
6	Q_CQp_INIT_DELAY_DONE	0
5	Q_CQn_INIT_DELAY_DONE	0
4	CQp_CAL_DONE	0
3	CQn_CAL_DONE	0
2	WE_CAL_DONE_CQp	0
1	WE_CAL_DONE_CQn	0
0	CAL_DONE	0

I.3.15 Xilinx SRAM Controller Count Status Register, Address: 0x031C

Hardware Function:	0x0
Device Address:	0x031C
Read/Write:	Read Only

This is the Xilinx SRAM Controller Count Status register. This register is used to troubleshoot the calibration routine in the Xilinx SRAM Controller. The calibration routine is made up of several stages. Each stage of the routine calculates a tap count that is used to center the CQ_p clock in the lower 18-bits of read data and the CQ_n clock in the upper 18-bits of read data.

Table I.27: Xilinx SRAM Controller Count Status Register

BIT	NAME	RESET
31:26	RSVD	0
25:20	Q_CQp_INIT_DELAY_DONE_TAP_CNT[5:0]	0
19:14	Q_CQn_INIT_DELAY_DONE_TAP_CNT[5:0]	0
13:8	CQp_CAL_TAP_CNT[5:0]	0
7:2	CQn_CAL_TAP_CNT[5:0]	0
1	CQp_Q_DATA_VALID	0
0	CQn_Q_DATA_VALID	0

I.3.16 Xilinx SRAM Controller Debug Control Register, Address: 0x031E

Hardware Function:	0x0
Device Address:	0x031E
Read/Write:	Read Only

This is the Xilinx SRAM Controller Debug Control register. This register is used to troubleshoot the calibration routine in the Xilinx SRAM Controller. The signals below allow for manual control of the calibration routines. For a detailed description of the signals below, see the appendix of the Xilinx MIG User's Guide (ug086.pdf) [27].

Table I.28: Xilinx SRAM Controller Debug Control Register

BIT	NAME	RESET
31:18	RSVD	0
17	DBG_IDEL_UP_ALL	0
16	DBG_IDEL_DOWN_ALL	0
15	DBG_SEL_ALL_IDEL_CQ	0
14	DBG_SEL_IDEL_CQ	0
13	DBG_IDEL_UP_CQ	0
12	DBG_IDEL_DOWN_CQ	0
11	DBG_SEL_ALL_IDEL_CQ_n	0
10	DBG_SEL_IDEL_CQ_n	0
9	DBG_IDEL_UP_CQ_n	0
8	DBG_IDEL_DOWN_CQ_n	0
7	DBG_SEL_ALL_IDEL_Q_CQ	0
6	DBG_SEL_IDEL_Q_CQ	0
5	DBG_SEL_ALL_IDEL_Q_CQ_n	0
4	DBG_IDEL_UP_Q_CQ	0
3	DBG_IDEL_DOWN_Q_CQ	0
2	DBG_SEL_IDEL_Q_CQ_n	0
1	DBG_IDEL_UP_Q_CQ_n	0
0	DBG_IDEL_DOWN_Q_CQ_n	0

I.3.17 Custom SRAM Controller FIFO Status Register, Address: 0x0320

Hardware Function:	0x0
Device Address:	0x0320
Read/Write:	Read Only

This is the custom SRAM Controller FIFO Status register. This register is used to troubleshoot the read data path of the SRAM Controllers. These bits should always be 0.

Table I.29: Custom SRAM Controller FIFO Status Register

BIT	NAME	RESET
31:2	RSVD	0
1	SRAM_FIFO_FULL	0
0	SRAM_FIFO_EMPTY	0

I.3.18 Playback RAM Write Trigger Data Register, Address: 0x0322

Hardware Function:	0x0
Device Address:	0x0322
Read/Write:	Read and Write

This is the Playback RAM Write Trigger Data registers. This register contains trigger location bits that are stored in the MSB 4-bits of each of the 4x 36-bit SRAM words. This trigger data is all ones at the first location in the SRAM, and all zeros at all remaining SRAM locations. This allows the trigger pulse to be synchronous to the SRAM data.

Table I.30: Playback RAM Write Trigger Data Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	PLAY_TRIG_DATA[15:0]	0x0000

I.3.19 Capture RAM Read Control Register, Address: 0x0324

Hardware Function:	0x0
Device Address:	0x0324
Read/Write:	Read and Read

This is the Capture RAM Read Control register. This register initiates an SRAM 4-word burst read. The read data resides in registers 0x0328 to 0x032E, and the read address resides in register 0x0326. When performing an SRAM 4-word burst read, the CPU must toggle the CAP_SRAM_RD_EN bit high then low. The time between the Capture SRAM Read Enable bit being set high then low is inconsequential, because a one-shot circuit inside the FPGA creates a single pulse one clock period wide when it detects a 0→1 transition on its input. To enable Capture mode set CAP_NPLAY to a logic high. To enable Playback mode set CAP_NPLAY to a logic low.

Table I.31: Capture RAM Read Control Register

BIT	NAME	RESET
31:3	RSVD	0
2	CAP_NPLAY	0
1	CAP_SRAM_RD_EN	0
0	RSVD	0

I.3.20 Capture RAM Read Address Register, Address: 0x0326

Hardware Function:	0x0
Device Address:	0x0326
Read/Write:	Read and Write

These are the Capture RAM Read Address registers. This register is 18-bits wide, and addresses the 4-word burst read transaction.

Table I.32: Capture RAM Read Address Register

BIT	NAME	RESET
31:18	RSVD	0
17:0	CAP_RD_ADDR[17:0]	0x00000

I.3.21 Capture RAM Write Start Address Register, Address: 0x0328

Hardware Function:	0x0
Device Address:	0x0328
Read/Write:	Read and Write

This is the Capture RAM Write Start Address register. This register indicates the start address that the data is streamed into the SRAM. It is 18-bits wide.

Table I.33: Capture RAM Write Start Address Register

BIT	NAME	RESET
31:18	RSVD	0
17:0	CAP_WR_START_ADDR[17:0]	0x00000

I.3.22 Capture RAM Write Stop Address Register, Address: 0x032A

Hardware Function:	0x0
Device Address:	0x032A
Read/Write:	Read and Write

This is the Capture RAM Write Stop Address register. This register indicates the stop address that the data is streamed into the SRAM. It is 18-bits wide.

Table I.34: Capture RAM Write Stop Address Register

BIT	NAME	RESET
31:18	RSVD	0
17:0	CAP_WR_STOP_ADDR[17:0]	0x3FFFF

I.3.23 Capture RAM Write Increment Address Register, Address: 0x032C

Hardware Function:	0x0
Device Address:	0x032C
Read/Write:	Read and Write

This is the Capture RAM Write Increment Address register. This register indicates the value that the SRAM address is incremented when streaming data into the SRAM. It is 18-bits wide.

Table I.35: Capture RAM Write Increment Address Register

BIT	NAME	RESET
31:18	RSVD	0
17:0	CAP_WR_INC_ADDR[17:0]	0x00001

I.3.24 Capture RAM Write Maximum Address Register, Address: 0x032E

Hardware Function:	0x0
Device Address:	0x032E
Read/Write:	Read and Write

This is the Capture RAM Write Maximum Address register. This register indicates the maximum initial address, which is used to pre-load the SRAM stream address counter. It is 18-bits wide.

Table I.36: Capture RAM Read Maximum Address Register

BIT	NAME	RESET
31:18	RSVD	0
17:0	CAP_WR_MAX_ADDR[17:0]	0x3FFFE

I.3.25 Capture RAM Read Data 0 Register, Address: 0x0330

Hardware Function:	0x0
Device Address:	0x0330
Read/Write:	Read Only

These are the Capture RAM Read Data 0 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 4-word burst read transaction. This 16-bit packet is the LSB Word in the 128-bit packet.

Table I.37: Capture RAM Read Data 0 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD0_DATA[15:0]	0x0000

I.3.26 Capture RAM Read Data 1 Register, Address: 0x0332

Hardware Function:	0x0
Device Address:	0x0332
Read/Write:	Read Only

These are the Capture RAM Read Data 1 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 4-word burst read transaction.

Table I.38: Capture RAM Read Data 1 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD1_DATA[15:0]	0x0000

I.3.27 Capture RAM Read Data 2 Register, Address: 0x0334

Hardware Function:	0x0
Device Address:	0x0334
Read/Write:	Read Only

These are the Capture RAM Read Data 2 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 4-word burst read transaction.

Table I.39: Capture RAM Read Data 2 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD2_DATA[15:0]	0x0000

I.3.28 Capture RAM Read Data 3 Register, Address: 0x0336

Hardware Function:	0x0
Device Address:	0x0336
Read/Write:	Read Only

These are the Capture RAM Read Data 3 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 4-word burst read transaction.

Table I.40: Capture RAM Read Data 3 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD3_DATA[15:0]	0x0000

I.3.29 Capture RAM Read Data 4 Register, Address: 0x0338

Hardware Function:	0x0
Device Address:	0x0338
Read/Write:	Read Only

These are the Capture RAM Read Data 4 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 4-word burst read transaction.

Table I.41: Capture RAM Read Data 4 Register

BIT	**NAME**	**RESET**
31:16	RSVD	0
15:0	CAP_RD4_DATA[15:0]	0x0000

I.3.30 Capture RAM Read Data 5 Register, Address: 0x033A

Hardware Function:	0x0
Device Address:	0x033A
Read/Write:	Read Only

These are the Capture RAM Read Data 5 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 4-word burst read transaction.

Table I.42: Capture RAM Read Data 5 Register

BIT	**NAME**	**RESET**
31:16	RSVD	0
15:0	CAP_RD5_DATA[15:0]	0x0000

I.3.31 Capture RAM Read Data 6 Register, Address: 0x033C

Hardware Function:	0x0
Device Address:	0x033C
Read/Write:	Read Only

These are the Capture RAM Read Data 6 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 4-word burst read transaction.

Table I.43: Capture RAM Read Data 6 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD6_DATA[15:0]	0x0000

I.3.32 Capture RAM Read Data 7 Register, Address: 0x033E

Hardware Function:	0x0
Device Address:	0x033E
Read/Write:	Read Only

These are the Capture RAM Read Data 7 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 4-word burst read transaction. This 16-bit packet is the MSB word in the 128-bit data word.

Table I.44: Capture RAM Read Data 6 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD7_DATA[15:0]	0x0000

I.3.33 Capture RAM Read Data 8 Register, Address: 0x0340

Hardware Function:	0x0
Device Address:	0x0340
Read/Write:	Read Only

These are the Capture RAM Read Data 8 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 4-word burst read transaction. This 16-bit packet contains the corresponding overflow flags for the 128-bit packet.

Table I.45: Capture RAM Read Data 7 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD8_DATA[15:0]	0x0000

I.3.34 Capture RAM Read Status Register, Address: 0x0342

Hardware Function:	0x0
Device Address:	0x0342
Read/Write:	Read Only

This is the Capture RAM Read Status register. The CAP_SRAM_RD_DONE bit indicates when a single 128-bit packet has been successfully read from the QDR-II SRAM. The Control FPGA must wait until the CAP_SRAM_RD_DONE bit goes high before initiating a new QDR-II SRAM Read Sequence.

Table I.46: Capture RAM Read Status Register

BIT	NAME	RESET
31:2	RSVD	0
1	CAP_SRAM_RD_DONE	0
0	RSVD	1

I.3.35 Playback RAM FIFO Read Count Register, Address: 0x0344

Hardware Function:	0x0
Device Address:	0x0344
Read/Write:	Read Only

This is the Playback RAM FIFO Read Count register. The PLAY_SRAM_FIFO_RD_CNT register indicates how many words are in the SRAM Playback FIFO.

Table I.47: Playback RAM FIFO Read Count Register

BIT	NAME	RESET
31:12	RSVD	0
11:0	PLAY_SRAM_FIFO_RD_CNT[11:0]	0

I.3.36 Playback Block RAM Write Control Register, Address: 0x0380

Hardware Function:	0x0
Device Address:	0x0380
Read/Write:	Read and Write

This is the Playback Block RAM Write Control register. This register initiates a Block RAM 128-bit write. When performing a Block RAM 128-bit write, the CPU must set these bits high.

Table I.48: Playback Block RAM Write Control Register

BIT	NAME	RESET
31:1	RSVD	0
0	PLAY_BRAM_WR_EN	0

I.3.37 Playback Block RAM Write Address Register, Address: 0x0382

Hardware Function:	0x0
Device Address:	0x0382
Read/Write:	Read and Write

This is the Playback Block RAM Write Address register. This register is 14-bits wide, and addresses the 128-bit write transaction.

Table I.49: Playback Block RAM Write Address Register

BIT	NAME	RESET
31:14	RSVD	0
13:0	PLAY_BRAM_WR_ADDR[13:0]	0x0000

I.3.38 Playback Block RAM Write Data 0 Register, Address: 0x0384

Hardware Function:	0x0
Device Address:	0x0384
Read/Write:	Read and Write

This is the Playback Block RAM Write Data 0 register. This register is 32-bits wide, and makes up one word of the 128-bit write transaction. This 32-bit packet is the LSB Word in the 128-bit packet.

Table I.50: Playback Block RAM Write Data 0 Register

BIT	NAME	RESET
31:0	PLAY_BRAM_WR0_DATA[31:0]	0x00000000

I.3.39 Playback Block RAM Write Data 1 Register, Address: 0x0386

Hardware Function:	0x0
Device Address:	0x0386
Read/Write:	Read and Write

This is the Playback Block RAM Write Data 1 register. This register is 32-bits wide, and makes up one word of the 128-bit write transaction.

Table I.51: Playback Block RAM Write Data 1 Register

BIT	NAME	RESET
31:0	PLAY_BRAM_WR1_DATA[31:0]	0x00000000

I.3.40 Playback Block RAM Write Data 2 Register, Address: 0x0388

Hardware Function:	0x0
Device Address:	0x0388
Read/Write:	Read and Write

This is the Playback Block RAM Write Data 2 register. This register is 32-bits wide, and makes up one word of the 128-bit write transaction.

Table I.52: Playback Block RAM Write Data 2 Register

BIT	NAME	RESET
31:0	BRAM_WR2_DATA[31:0]	0x00000000

I.3.41 Playback Block RAM Write Data 3 Register, Address: 0x038A

Hardware Function:	0x0
Device Address:	0x038A
Read/Write:	Read and Write

This is the Playback Block RAM Write Data 3 register. This register is 32-bits wide, and makes up one word of the 128-bit write transaction. This 32-bit packet is the MSB Word in the 128-bit packet.

Table I.53: Playback Block RAM Write Data 3 Register

BIT	**NAME**	**RESET**
31:0	PLAY_BRAM_WR3_DATA[31:0]	0x00000000

I.3.42 Playback Block RAM Read Control Register, Address: 0x038C

Hardware Function:	0x0
Device Address:	0x038C
Read/Write:	Read and Write

This is the Playback Block RAM Read Control register. This register initiates an Block RAM read. When performing a Block RAM Read initiation, the CPU must set bit 0 high.

Table I.54: Playback Block RAM Read Control Register

BIT	**NAME**	**RESET**
31:1	RSVD	0
0	PLAY_BRAM_RD_EN	0

I.3.43 Playback Block RAM Read Start Address Register, Address: 0x038E

Hardware Function:	0x0
Device Address:	0x038E
Read/Write:	Read and Write

This is the Playback Block RAM Read Start Address (Pattern A) register. This register is 14-bits wide, and addresses the 4-word burst read transaction.

Table I.55: Block RAM Read Start Address Register

BIT	NAME	RESET
31:14	RSVD	0
13:0	PLAY_BRAM_RD_START_ADDR[13:0]	0

I.3.44 Playback Block RAM Read Stop Address Register, Address: 0x0390

Hardware Function:	0x0
Device Address:	0x0390
Read/Write:	Read and Write

These are the Playback Block RAM Read Stop Address register. This register is 14-bits wide, and addresses the 128-bit read transaction.

Table I.56: Playback Block RAM Read Stop Address Register

BIT	NAME	RESET
31:14	RSVD	0
13:0	PLAY_BRAM_RD_STOP_ADDR[13:0]	0

I.3.45 Capture Block RAM Read Data 0 Register, Address: 0x039A

Hardware Function:	0x0
Device Address:	0x039A
Read/Write:	Read Only

These are the Capture Block RAM Read Data 0 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 128-bit read transaction. This 16-bit packet is the LSB Word in the 128-bit packet.

Table I.57: Capture Block RAM Read Data 0 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD0_DATA[15:0]	0x0000

I.3.46 Capture Block RAM Read Data 1 Register, Address: 0x039C

Hardware Function:	0x0
Device Address:	0x039C
Read/Write:	Read Only

These are the Capture Block RAM Read Data 1 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 128-bit read transaction.

Table I.58: Capture Block RAM Read Data 1 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD1_DATA[15:0]	0x0000

I.3.47 Capture Block RAM Read Data 2 Register, Address: 0x039E

Hardware Function:	0x0
Device Address:	0x039E
Read/Write:	Read Only

These are the Capture Block RAM Read Data 2 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 128-bit read transaction.

Table I.59: Capture Block RAM Read Data 2 Register

BIT	**NAME**	**RESET**
31:16	RSVD	0
15:0	CAP_RD2_DATA[15:0]	0x0000

I.3.48 Capture Block RAM Read Data 3 Register, Address: 0x03A0

Hardware Function:	0x0
Device Address:	0x03A0
Read/Write:	Read Only

These are the Capture Block RAM Read Data 3 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 128-bit read transaction.

Table I.60: Capture Block RAM Read Data 3 Register

BIT	**NAME**	**RESET**
31:16	RSVD	0
15:0	CAP_RD3_DATA[15:0]	0x0000

I.3.49 Capture Block RAM Read Data 4 Register, Address: 0x03A2

Hardware Function:	0x0
Device Address:	0x03A2
Read/Write:	Read Only

These are the Capture Block RAM Read Data 4 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 128-bit read transaction.

Table I.61: Capture Block RAM Read Data 4 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD4_DATA[15:0]	0x0000

I.3.50 Capture Block RAM Read Data 5 Register, Address: 0x03A4

Hardware Function:	0x0
Device Address:	0x03A4
Read/Write:	Read Only

These are the Capture Block RAM Read Data 5 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 128-bit read transaction.

Table I.62: Capture Block RAM Read Data 5 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD5_DATA[15:0]	0x0000

I.3.51 Capture Block RAM Read Data 6 Register, Address: 0x03A6

Hardware Function:	0x0
Device Address:	0x03A6
Read/Write:	Read Only

These are the Capture Block RAM Read Data 6 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 128-bit read transaction.

Table I.63: Capture Block RAM Read Data 6 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD6_DATA[15:0]	0x0000

I.3.52 Capture Block RAM Read Data 7 Register, Address: 0x03A8

Hardware Function:	0x0
Device Address:	0x03A8
Read/Write:	Read Only

These are the Capture Block RAM Read Data 7 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 128-bit read transaction. This 16-bit packet is the MSB word in the 128-bit data word.

Table I.64: Capture Block RAM Read Data 6 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD7_DATA[15:0]	0x0000

I.3.53 Capture Block RAM Read Data 8 Register, Address: 0x03AA

Hardware Function:	0x0
Device Address:	0x03AA
Read/Write:	Read Only

These are the Capture Block RAM Read Data 8 registers. These registers comprise a register that is 16-bits wide, and make up one word of the 128-bit read transaction. This 16-bit packet contains the corresponding overflow flags for the 128-bit packet.

Table I.65: Capture Block RAM Read Data 7 Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	CAP_RD8_DATA[15:0]	0x0000

I.3.54 Capture Block RAM Write Status Register, Address: 0x03AC

Hardware Function:	0x0
Device Address:	0x03AC
Read/Write:	Read Only

This is the Capture Block RAM Write Status register. The CAP_BRAM_WR_DONE bit indicates when a single 128-bit packet has been successfully read from the Block RAM. The Control FPGA must wait until the CAP_BRAM_WR_DONE bit goes high before initiating a new Block RAM Write Sequence.

Table I.66: Capture Block RAM Write Status Register

BIT	NAME	RESET
31:1	RSVD	0
0	CAP_BRAM_WR_DONE	0

I.3.55 Playback Block RAM Write Trigger Register, Address: 0x03AE

Hardware Function:	0x0
Device Address:	0x03AE
Read/Write:	Read and Write

These are the Playback Block RAM Write Trigger registers. This 16-bit packet is the trigger word for the 128-bit data word.

Table I.67: Playback Block RAM Write Trigger Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	PLAY_BRAM_WR_TRIG[15:0]	0x0000

I.4 Base Address: 0x0400, Miscellaneous Control Registers

I.4.1 AUX Input Control Register, Address: 0x0400

Hardware Function:	0x0
Device Address:	0x0400
Read/Write:	Read and Write

The AUX Input Control register is used to set the current mode of the external AUX Input. The states of the AUX Input Control register are shown in Table I.69.

Table I.68: AUX Input Control Register

BIT	NAME	RESET
31:3	RSVD	0
2:0	AUX_CTRL	0

Table I.69: AUX Input Control Modes

AUX_CTRL			**Mode**
2	**1**	**0**	
0	x	x	Disabled
1	0	0	Gating
1	1	0	Sweep

I.4.2 RAM Select Register, Address: 0x0402

Hardware Function:	0x0
Device Address:	0x0402
Read/Write:	Read and Write

The RAM Select register is used to select between Block RAM patterns and QDR-II SRAM patterns. The states of the RAM Select register are shown in Table I.71.

Table I.70: Pattern Select Register

BIT	**NAME**	**RESET**
31:1	RSVD	0
0	BRAM_nSRAM	0

Table I.71: RAM Select Modes

BRAM_nSRAM	**Mode**
0	QDR-II SRAM Pattern
1	Block RAM Pattern

I.4.3 Sweep Control Register, Address: 0x0404

Hardware Function:	0x0
Device Address:	0x0404
Read/Write:	Read and Write

This is the Sweep Control register. This register is used to enable sweep.

Table I.72: Sweep Control Register

BIT	**NAME**	**RESET**
31:1	RSVD	0
0	SWEEP	0

I.4.4 Gating Control Register, Address: 0x0406

Hardware Function:	0x0
Device Address:	0x0406
Read/Write:	Read and Write

This is the Gating Control register. This register is used to enable waveform gating.

Table I.73: Gating Control Register

BIT	**NAME**	**RESET**
31:1	RSVD	0
0	GATE	0

I.4.5 Waveform Enable Register, Address: 0x0408

Hardware Function:	0x0
Device Address:	0x0408
Read/Write:	Read and Write

This is the Waveform Enable register. This register is used to enable waveform.

Table I.74: Waveform Enable Register

BIT	**NAME**	**RESET**
31:1	RSVD	0
0	WAVE_EN	0

I.4.6 DC/RAM Select Register, Address: 0x040A

Hardware Function:	0x0
Device Address:	0x040A
Read/Write:	Read and Write

The DC/RAM Select register is used to select between RAM patterns and a DC value of 16'h0000. The states of the DC/RAM Select register are shown in Table I.76.

Table I.75: Pattern Select Register

BIT	**NAME**	**RESET**
31:1	RSVD	0
0	DC_nRAM	0

Table I.76: RAM Select Modes

BRAM_nSRAM	**Mode**
0	RAM Pattern
1	DC Value

I.4.7 Pattern Shift Register, Address: 0x040C

Hardware Function:	0x0
Device Address:	0x040C
Read/Write:	Read and Write

The Pattern Shift register is used to perform a shift by 16-K bits, where the value of this register is 2^K. The states of the Pattern Shift register are shown in Table I.78.

Table I.77: Pattern Shift Register

BIT	**NAME**	**RESET**
31:16	RSVD	0
15:0	WAVE_SHIFT[15:0]	0x0001

Table I.78: Pattern Shift Modes

Bits to Shift	**Value**
0	0x10000
1	0x08000
2	0x04000
3	0x02000
4	0x01000
5	0x00800
6	0x00400
7	0x00200
8	0x00100
9	0x00080
10	0x00040
11	0x00020
12	0x00010
13	0x00008
14	0x00004
15	0x00002
16	0x00001

I.4.8 Pattern Invert Register, Address: 0x040E

Hardware Function:	0x0
Device Address:	0x040E
Read/Write:	Read and Write

The Pattern Invert register is used to perform a two's complement invert of the current pattern. The states of the Pattern Invert register are shown in Table I.80.

Table I.79: Pattern Invert Register

BIT	**NAME**	**RESET**
31:16	RSVD	0
15:0	WAVE_INVERT[15:0]	0x0001

Table I.80: Pattern Invert Modes

PAT_INV	**Mode**
0x0001	Normal Pattern
0xFFFF	Inverted Pattern

I.5 Base Address: 0x0500, Trigger Control Registers

I.5.1 Trigger Output Mode Register, Address: 0x0500

Hardware Function:	0x0
Device Address:	0x0500
Read/Write:	Read and Write

This is the Trigger Output Mode register. This register controls the type of trigger present on the front panel.

Table I.81: Trigger Output Mode Register

BIT	NAME	RESET
31:1	RSVD	0
0	TRIG_OUT_MODE	1

Table I.82: Trigger Output Modes

TRIG_OUT_MODE	Mode
0	Divided Clock Trigger
1	Pattern Trigger

I.5.2 Trigger RAM Mode Register, Address: 0x0502

Hardware Function:	0x0
Device Address:	0x0502
Read/Write:	Read and Write

This is the Trigger RAM Mode register. This register controls the type of trigger for the RAM Patterns.

Table I.83: Trigger RAM Mode Register

BIT	NAME	RESET
31:1	RSVD	0
0	TRIG_RAM_MODE	0

Table I.84: Trigger RAM Modes

TRIG_RAM_MODE	Mode
0	A Only Pattern Trigger
1	A/B Pattern Trigger

I.5.3 RAM Pattern A Trigger Address Register, Address: 0x0504

Hardware Function:	0x0
Device Address:	0x0504
Read/Write:	Read and Write

This is the RAM Pattern A Trigger Address register. This register is 18-bits wide, and makes up the RAM pattern A trigger address register used by the RAM Pattern Trigger Module.

Table I.85: RAM Pattern A Trigger Address Register

BIT	NAME	RESET
31:18	RSVD	0
17:0	RAM_APATRN_TRIG[17:0]	0x2AAAA

I.5.4 Divided Clock Trigger Control Register, Address: 0x0506

Hardware Function:	0x0
Device Address:	0x0506
Read/Write:	Read and Write

This is the Divided Clock Trigger Control register. This register is used to select the divided clock frequency.

Table I.86: Divided Clock Trigger Control Register

BIT	NAME	RESET
31:2	RSVD	0
1:0	DIV_CLK_TRIG[1:0]	0b01

Table I.87: Divided Clock Trigger Modes

DIV_CLK_TRIG	Mode	Frequency
00	clk	500 MHz
01	$\frac{clk}{2}$ (default)	250 MHz
10	$\frac{clk}{4}$	125 MHz
11	$\frac{clk}{8}$	62.5 MHz

I.6 Base Address: 0x0700, ADC Control Registers

I.6.1 ADC Data FIFO Empty Register, Address: 0x071A

Hardware Function:	0x0
Device Address:	0x071A
Read/Write:	Read and Write

The ADC Data FIFO Empty register is used to read the Empty status of the FIFOs. These will most likely be used for debug only as the FIFOs are continuously filled and emptied.

Table I.88: ADC Data FIFO Empty Register

BIT	**NAME**	**RESET**
31:18	RSVD	0
17:0	MUX_FIFO_EMPTY	0

I.6.2 ADC Data FIFO Full Register, Address: 0x071C

Hardware Function:	0x0
Device Address:	0x071C
Read/Write:	Read and Write

The ADC Data FIFO Full register is used to read the Full status of the FIFOs. These will most likely be used for debug only as the FIFOs are continuously filled and emptied.

Table I.89: ADC Data FIFO Full Register

BIT	**NAME**	**RESET**
31:18	RSVD	0
17:0	MUX_FIFO_FULL	0

I.7 Base Address: 0x0800, AsAP Control Registers

I.7.1 AsAP #0 Read Data Register, Address: 0x0800

Hardware Function:	0x0
Device Address:	0x0800
Read/Write:	Read Only

The AsAP #0 Read Data register is used to store the data read from AsAP #0.

Table I.90: AsAP #0 Read Data Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	ASAP0_RD_DATA[15:0]	0

I.7.2 AsAP #1 Read Data Register, Address: 0x0802

Hardware Function:	0x0
Device Address:	0x0802
Read/Write:	Read Only

The AsAP #1 Read Data register is used to store the data read from AsAP #1.

Table I.91: AsAP #1 Read Data Register

BIT	NAME	RESET
31:16	RSVD	0
15:0	ASAP1_RD_DATA[15:0]	0

I.7.3 AsAP #0 Control Register, Address: 0x0804

Hardware Function:	0x0
Device Address:	0x0804
Read/Write:	Read Only

The AsAP #0 Control register is used to control AsAP #0.

Table I.92: AsAP #0 Control Register

BIT	NAME	RESET
31:1	RSVD	0
0	ASAP0_CTRL	0

I.7.4 AsAP #1 Control Register, Address: 0x0806

Hardware Function:	0x0
Device Address:	0x0806
Read/Write:	Read Only

The AsAP #1 Control register is used to control AsAP #1.

Table I.93: AsAP #1 Control Register

BIT	NAME	RESET
31:1	RSVD	0
0	ASAP1_CTRL	0

Appendix J

Control FPGA EDK/MicroBlaze Peripherals

The Control FPGA on the measurement board serves many purposes, from configuring the Data Path FPGA to loading waveform files for the baseband signal source. The Control FPGA uses a Xilinx MicroBlaze 32-bit soft-core processor, which is implemented in the FPGA fabric. The MicroBlaze processor has many peripherals attached to its processor local bus (PLB) for controlling external SPI and I^2C devices. These devices are controlled by software written in C and C++, and running on the MicroBlaze processor. The software is developed using the Xilinx embedded development kit (EDK), which contains a library of common peripherals from SPI interface controllers to general purpose I/O controllers. In addition, the EDK software application allows custom processor peripherals to be written in either Verilog HDL or VHDL. This chapter will describe each of the EDK peripherals used by the MicroBlaze processor in the Control FPGA.

J.1 Field Upgrade

The measurement board contains both a 64 MB serial peripheral interface (SPI) configuration flash programmable read only memory (PROM) and a 2 GB microSD card for configuration file and calibration data storage. The boot-loader application will always be contained in the block ram of each FPGA image, and will rarely need to change. The factory image will always reside at location 0x0 in the SPI Configuration Flash PROM to provide a reliable way of recovering from a configuration error.

The upgrade image is stored in a file named "UPGRADE.MCS" in the CONFIG directory of the microSD. The MicroBlaze 32-bit micro-controller will then perform a CRC checksum test to verify that the image is valid and move the upgrade image into the upgrade location in the SPI Configuration Flash PROM.

At system power-up or restart, the Xilinx Spartan-3A control FPGA will be configured from whichever image the internal configuration access port (ICAP) peripheral is currently programmed to boot from in the SPI configuration flash PROM. The factory default is to boot from the factory image in the SPI configuration flash PROM. Once the control FPGA is configured the boot-loader will transfer the control application data, as part of the power-up procedure, from the SPI configuration flash PROM to the DDR SDRAM memory.

J.1.1 hw_icap_registers_0: ICAP Software Register Peripheral

The *hw_icap_registers_v1_00_a* EDK Peripheral is used to control the Xilinx Spartan-3A ICAP primitive (ICAP_SPARTAN3A). The ICAP_SPARTAN3A primitive works similar to the slave parallel (SelectMAP) configuration interface except it is available to the FPGA application using internal routing connections. Furthermore, the ICAP primitive has separate read and write data ports, as opposed to the bidirectional bus on the SelectMAP interface. ICAP allows the FPGA application to access configuration registers, readback configuration data, or to trigger a MultiBoot event after configuration successfully completes. The register map of the ICAP_SPARTAN3A primitive is shown in Table J.2. The variables shown in the *Default Value* column of Table J.2 are described in Table J.1. The *hw_icap_registers_v1_00_a* EDK Peripheral's register map is shown in Table J.3.

Table J.1: ICAP Variable Descriptions

Variable Name	**# of Bits**	**Description**
internal_addr	32	MultiBoot Address: SPI PROM Boot Location
internal_mode	3	BOOTMODE: M[2:0] Configuration Mode Pins
internal_vsel	3	VS[2:0] SPI Configuration Mode Pins
internal_use	1	NEW_MODE 0 = Sample M[2:0] and VS[2:0] pins 1 = Use BOOTMODE bits

Table J.2: **ICAP_SPARTAN3A** Register Map

Address	**Name**	**Default Value**
1	sync_H	0xAA
2	sync_L	0x99
3	t1w_gen1_H	0x32
4	t1w_gen1_L	0x61
5	addr[15:8]	internal_addr[15:8]
6	addr[7:0]	internal_addr[7:0]
7	t1w_gen2_H	0x32
8	t1w_gen2_L	0x81
9	addr[31:24]	internal_addr[31:24]
10	addr[23:16]	internal_addr[23:16]
11	t1w_mode_H	0x32
12	t1w_mode_L	0xA1
13	reserved	{1'b0, internal_use, internal_mode[2:0], internal_vsel[2:0]}
14	user_mode	0x00
15	t1w_cmd_H	0x30
16	t1w_cmd_L	0xA1
17	reboot_H	0x00
18	reboot_L	0x0E
19	noop_H	0x20
20	noop_L	0x00

Table J.3: ***hw_icap_registers_v1_00_a*** **EDK Peripheral Register Map**

Address	**Name**	**R/W**	**Default Value**	**Description**
0	{24'b0,icap_wr_reg[24:31]}	R/W	0x00000000	ICAP Write Byte Register.
1	{24'b0,icap_rd_reg[24:31]}	RO	0x00000000	ICAP Read Byte Register.
2	{30'b0,icap_ctrl[30:31]}	R/W	0x00000003	ICAP Control Register. [0:29]: reserved; [30]: icap_rnw; ICAP Read/nWrite Signal 1 = Read Operation. 0 = Write Operation. [31]: icap_ce; ICAP Clock Enable (Active Low)
3	{31'b0,icap_clk_r[31]}	R/W	0x00000001	ICAP Clock Register. [0:30]: reserved; [31]: icap_clk; ICAP Clock Toggle high then low to clock a data byte into or out of the ICAP module.
4	{31'b0,icap_busy}	RO	0x00000000	ICAP Busy/Ready output Register.
				[0:30]: reserved; [31]: icap_busy; ICAP Busy/Ready output (Active High). busy status. Only used in read operations. Busy remains Low during writes. Ops can be started when this bit goes high.

J.1.1.1 ICAP: Get Boot Address

To read the boot address from the ICAP module follow the steps outlined below:

1. **First get lower 16-bits**
2. Set the icap_ctrl[30:31] register bits (icap_rnw bit and icap_ce) to a 1; or 0x3.
3. Set the icap_ctrl[30] register bit (icap_rnw bit) to a 0.
4. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.
5. Write 0x20 (noop_H) to the icap_wr_reg register.
6. Set the icap_clk register bit to a 0.
7. Set the icap_clk register bit to a 1.
8. Write 0x00 (noop_L) to the icap_wr_reg register.
9. Set the icap_clk register bit to a 0.
10. Set the icap_clk register bit to a 1.
11. Write 0xAA (sync_H) to the icap_wr_reg register.
12. Set the icap_clk register bit to a 0.
13. Set the icap_clk register bit to a 1.
14. Write 0x99 (sync_L) to the icap_wr_reg register.
15. Set the icap_clk register bit to a 0.
16. Set the icap_clk register bit to a 1.
17. Write 0x2A (t1r_gen2_H) to the icap_wr_reg register.
18. Set the icap_clk register bit to a 0.
19. Set the icap_clk register bit to a 1.
20. Write 0x81 (t1r_gen2_L) to the icap_wr_reg register.
21. Set the icap_clk register bit to a 0.
22. Set the icap_clk register bit to a 1.

23. Write 0x20 (noop_H) to the icap_wr_reg register.
24. Set the icap_clk register bit to a 0.
25. Set the icap_clk register bit to a 1.
26. Write 0x00 (noop_L) to the icap_wr_reg register.
27. Set the icap_clk register bit to a 0.
28. Set the icap_clk register bit to a 1.
29. Set the icap_ctrl[31] register bit (icap_ce bit) to a 1.
30. Set the icap_ctrl[30] register bit (icap_rnw bit) to a 1.
31. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.
32. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.
33. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.
34. Read the value in the icap_rd_reg, store the value in the lower 8-bits of the Address variable (i.e., addr[7:0]).
35. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.
36. Read the value in the icap_rd_reg, store the value in the next 8-bits of the Address variable (i.e., addr[15:8]).
37. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.
38. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.
39. Set the icap_ctrl[31] register bit (icap_ce bit) to a 1.
40. **Now get upper 16-bits**
41. Set the icap_ctrl[30:31] register bits (icap_rnw bit and icap_ce) to a 1; or 0x3.
42. Set the icap_ctrl[30] register bit (icap_rnw bit) to a 0.
43. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.
44. Write 0x20 (noop_H) to the icap_wr_reg register.

45. Set the icap_clk register bit to a 0.
46. Set the icap_clk register bit to a 1.
47. Write 0x00 (noop_L) to the icap_wr_reg register.
48. Set the icap_clk register bit to a 0.
49. Set the icap_clk register bit to a 1.
50. Write 0xAA (sync_H) to the icap_wr_reg register.
51. Set the icap_clk register bit to a 0.
52. Set the icap_clk register bit to a 1.
53. Write 0x99 (sync_L) to the icap_wr_reg register.
54. Set the icap_clk register bit to a 0.
55. Set the icap_clk register bit to a 1.
56. Write 0x2A (t1r_gen1_H) to the icap_wr_reg register.
57. Set the icap_clk register bit to a 0.
58. Set the icap_clk register bit to a 1.
59. Write 0x61 (t1r_gen1_L) to the icap_wr_reg register.
60. Set the icap_clk register bit to a 0.
61. Set the icap_clk register bit to a 1.
62. Write 0x20 (noop_H) to the icap_wr_reg register.
63. Set the icap_clk register bit to a 0.
64. Set the icap_clk register bit to a 1.
65. Write 0x00 (noop_L) to the icap_wr_reg register.
66. Set the icap_clk register bit to a 0.
67. Set the icap_clk register bit to a 1.
68. Set the icap_ctrl[31] register bit (icap_ce bit) to a 1.

69. Set the icap_ctrl[30] register bit (icap_rnw bit) to a 1.

70. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.

71. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.

72. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.

73. Read the value in the icap_rd_reg, store the value in the next 8-bits of the Address variable (i.e., addr[23:16]).

74. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.

75. Read the value in the icap_rd_reg, store the value in the next 8-bits of the Address variable (i.e., addr[31:24]).

76. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.

77. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.

78. Set the icap_ctrl[31] register bit (icap_ce bit) to a 1.

79. Now we have the boot address stored in the Address variable.

J.1.1.2 ICAP: Reboot

To perform an FPGA reboot using the ICAP module follow the steps outlined below:

1. Set the icap_ctrl[30:31] register bits (icap_rnw bit and icap_ce) to a 1; or 0x3.
2. Set the icap_ctrl[30] register bit (icap_rnw bit) to a 0.
3. Set the icap_ctrl[31] register bit (icap_ce bit) to a 0.
4. Write 0x20 (noop_H) to the icap_wr_reg register.
5. Set the icap_clk register bit to a 0.
6. Set the icap_clk register bit to a 1.
7. Write 0x00 (noop_L) to the icap_wr_reg register.
8. Set the icap_clk register bit to a 0.
9. Set the icap_clk register bit to a 1.
10. Write 0xAA (sync_H) to the icap_wr_reg register.
11. Set the icap_clk register bit to a 0.
12. Set the icap_clk register bit to a 1.
13. Write 0x99 (sync_L) to the icap_wr_reg register.
14. Set the icap_clk register bit to a 0.
15. Set the icap_clk register bit to a 1.
16. Write 0x32 (t1w_gen1_H) to the icap_wr_reg register.
17. Set the icap_clk register bit to a 0.
18. Set the icap_clk register bit to a 1.
19. Write 0x61 (t1w_gen1_L) to the icap_wr_reg register.
20. Set the icap_clk register bit to a 0.
21. Set the icap_clk register bit to a 1.
22. Write addr[15:8] to the icap_wr_reg register.

23. Set the icap_clk register bit to a 0.
24. Set the icap_clk register bit to a 1.
25. Write addr[7:0] to the icap_wr_reg register.
26. Set the icap_clk register bit to a 0.
27. Set the icap_clk register bit to a 1.
28. Write 0x32 (t1w_gen2_H) to the icap_wr_reg register.
29. Set the icap_clk register bit to a 0.
30. Set the icap_clk register bit to a 1.
31. Write 0x81 (t1w_gen2_L) to the icap_wr_reg register.
32. Set the icap_clk register bit to a 0.
33. Set the icap_clk register bit to a 1.
34. Write 0x03 (C7:0) to the icap_wr_reg register.
35. Set the icap_clk register bit to a 0.
36. Set the icap_clk register bit to a 1.
37. Write addr[23:16] to the icap_wr_reg register.
38. Set the icap_clk register bit to a 0.
39. Set the icap_clk register bit to a 1.
40. Write 0x32 (t1w_mode_H) to the icap_wr_reg register.
41. Set the icap_clk register bit to a 0.
42. Set the icap_clk register bit to a 1.
43. Write 0xA1 (t1w_mode_L) to the icap_wr_reg register.
44. Set the icap_clk register bit to a 0.
45. Set the icap_clk register bit to a 1.
46. Write 0x00 to the icap_wr_reg register.

47. Set the icap_clk register bit to a 0.
48. Set the icap_clk register bit to a 1.
49. Write 0x4D (sample M[2:0] and VS[2:0] pins) to the icap_wr_reg register.
50. Set the icap_clk register bit to a 0.
51. Set the icap_clk register bit to a 1.
52. Write 0x30 (t1w_cmd_H) to the icap_wr_reg register.
53. Set the icap_clk register bit to a 0.
54. Set the icap_clk register bit to a 1.
55. Write 0xA1 (t1w_cmd_L) to the icap_wr_reg register.
56. Set the icap_clk register bit to a 0.
57. Set the icap_clk register bit to a 1.
58. Write 0x00 (reboot_H) to the icap_wr_reg register.
59. Set the icap_clk register bit to a 0.
60. Set the icap_clk register bit to a 1.
61. Write 0x0E (reboot_L) to the icap_wr_reg register.
62. Set the icap_clk register bit to a 0.
63. Set the icap_clk register bit to a 1.
64. Write 0x20 (noop_H) to the icap_wr_reg register.
65. Set the icap_clk register bit to a 0.
66. Set the icap_clk register bit to a 1.
67. Write 0x00 (noop_L) to the icap_wr_reg register.
68. Set the icap_clk register bit to a 0.
69. Set the icap_clk register bit to a 1.
70. Write 0x20 (noop_H) to the icap_wr_reg register.

71. Set the icap_clk register bit to a 0.
72. Set the icap_clk register bit to a 1.
73. Write 0x00 (noop_L) to the icap_wr_reg register.
74. Set the icap_clk register bit to a 0.
75. Set the icap_clk register bit to a 1.
76. Set the icap_ctrl[31] register bit (icap_rnw bit) to a 1.
77. Set the icap_ctrl[30] register bit (icap_ce bit) to a 1.

J.2 Data Path FPGA Configuration

The Control FPGA is responsible for configuring the Data Path FPGA at power-up. The Control FPGA will configure the Data Path FPGA using the slave serial method. The slave serial interface consists of the following 5 signals:

1. CCLK: Serial Configuration Clock
2. DIN: Serial Data In
3. INIT_B: Initialization Flag (Active Low)
4. PROG_B: Configuration Reset (Active Low)
5. DONE: Configuration Done Flag (Active High)

The Data Path FPGA configuration process consists of sending 2,730,704 bytes to the Data Path slave serial interface 32-bits at a time using the *ctx_dp_fpga_config_v1_00_a* EDK peripheral. First the PROG_B signal is toggled low then high to initiate the Data Path FPGA configuration process. The MicroBlaze Data Path FPGA configuration application will wait until the INIT_B signal transitions from low to high. Once INIT_B goes high, the MicroBlaze Data Path FPGA configuration application will begin to send 32-bit words one at a time, while waiting for the txfer_done signal of the *ctx_dp_fpga_config_v1_00_a* EDK peripheral to go high between each 32-bit word. Once the last 32-bit configuration data packet has been sent via the slave serial interface, the MicroBlaze Data Path FPGA configuration application will wait for the configuration done flag to go high. This will signal the completion of the Data Path FPGA configuration process. If the configuration done flag does not go high, the process should be executed again.

J.2.1 ctx_dp_fpga_config_0: Data Path FPGA Configuration Peripheral

The *ctx_dp_fpga_config_v1_00_a* EDK peripheral is used to configure the Data Path FPGA, and appears as a set of 4 software accessible registers to the applications running on the MicroBlaze. The *ctx_dp_fpga_config_v1_00_a* EDK peripheral's register map is shown in Table J.5.

To initiate a write transaction using the *ctx_dp_fpga_config_v1_00_a* EDK peripheral first set the read/write select to write (rnw_i = 0), 16-bit address (saddr_i) and 32-bit data (sdata_i), then toggle the start_spi_i bit low-high-low to initiate the transfer.

To initiate a read transaction using the *ctx_dp_fpga_config_v1_00_a* EDK peripheral first set the read/write select to read (rnw_i = 1) and 16-bit address (saddr_i), then toggle the start_spi_i bit low-high-low to initiate the transfer.

The *ctx_dp_fpga_config_v1_00_a* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.4.

Table J.4: ***ctx_dp_fpga_config_v1_00_a* EDK peripheral I/O descriptions**

***xps_spi* Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
i_dp_fpga_done	CUST_CFG_DONE	I	B22	Configuration Done (Active High).
i_dp_fpga_initb	CUST_INIT_B	I	B21	Initialization Flag (Active Low).
o_dp_fpga_cclk	CUST_CCLK	O	C21	Serial Configuration Clock.
o_dp_fpga_din	S3A_SPI_DATA_TO_V5	O	AA15	Serial Data.
o_dp_fpga_progb	CUST_PROG_B	O	C22	Configuration Reset (Active Low).

Table J.5: ***ctx_dp_fpga_config_v1_00_a*** **EDK peripheral register map**

Address	**Name**	**R/W**	**Default Value**	**Description**
0	cfg_data[0:31]	R/W	0x00000000	32-bit Configuration Write Data Register.
1	{29'b0,prog_reg[29:31]}	R/W	0x00000001	Configuration Control Register. [0:28]: reserved; [29]: send_clks: Send 16 Clocks (Active High). Toggle high-then-low to initiate transaction. [30]: send_data: Send 32-bits of Data (Active High). Toggle high-then-low to initiate transaction. [31]: PROGB: Initiate FPGA Program Sequence. Toggle low-then-high to initiate transaction.
2	slv_reg2[0:31]	R/W	0x00000000	Reserved Register.
3	{29'b0,txfer_done,DONE,INITB}	RO	0x00000000	Configuration Transfer Done Register. [0:28]: reserved; [29]: txfer_done: Serial Transfer Done (Active High). [30]: DONE: Configuration Done (Active High). [31]: INITB: Initialization Flag (Active Low).

J.3 Data Path FPGA Control

The data path FPGA is responsible for generating and capturing waveforms as well as interfacing with the AsAP processors and a variety of memory devices. It is implemented in SystemVerilog, and targeted on a Xilinx Virtex-5 SX50T (XC5VSX50T-1FFG1136C). The data path FPGA contains hardware registers that need to be initialized at power-up and changed based on user input. The hardware registers are accessed via an instrument local bus (ILB), which is based on the serial peripheral interface standard. The SPI bus consists of the following 6 signals:

1. SCK: Serial Clock
2. MOSI: Master-Out/Slave-In
3. MISO: Master-In/Slave-Out
4. ADDR_CSn: Address Chip Select (Active Low)
5. DATA_CSn: Data Chip Select (Active Low)
6. RNW: Read/Write Select; 0 = Write, 1 = Read

The data path FPGA is a slave, so it needs a means of requesting service from the master. An additional signal is provided for this purpose:

1. SRQn: Service Request (Active Low)

A single ILB transaction consists of two segments: address and data. The address value consists of 16-bits, and the data value consists of 32-bits. First, the desired address shifted out over MOSI as the ADDR_CSn chip select is driven low. Once the address has been shifted out to the slave, the ADDR_CSn chip select is driven high. Then two SCK pulses are generated before the DATA_CSn chip select is driven low while shifting the write data out over MOSI. Once the data has been shifted out to the slave, the DATA_CSn chip select is driven high with two SCK pulses following. The two SCK pulses that precede the DATA_CSn chip select allow for the addressed register contents to be shifted to the master over MISO. The two SCK pulses that follow the DATA_CSn chip select allow for the slave to perform any last minute tasks before the transaction is completed.

As the data path FPGA control interface is a non-standard SPI interface, a custom EDK peripheral called *ctx_dp_fpga_ctrl_v1_00_a* was designed to implement the SPI transactions.

J.3.1 ctx_dp_fpga_ctrl_0: data path FPGA Control Peripheral

The *ctx_dp_fpga_ctrl_v1_00_a* EDK peripheral is used to control the data path FPGA, and appears as a set of 8 software accessible registers to the applications running on the MicroBlaze. The *ctx_dp_fpga_ctrl_v1_00_a* EDK peripheral's register map is shown in Table J.7.

To initiate a write transaction using the *ctx_dp_fpga_ctrl_v1_00_a* EDK peripheral first set the read/write select to write (rnw_i = 0), 16-bit address (saddr_i) and 32-bit data (sdata_i), then toggle the start_spi_i bit low-high-low to initiate the transfer.

To initiate a read transaction using the *ctx_dp_fpga_ctrl_v1_00_a* EDK peripheral first set the Read/Write Select to read (rnw_i = 1) and 16-bit address (saddr_i), then toggle the start_spi_i bit low-high-low to initiate the transfer.

The *ctx_dp_fpga_ctrl_v1_00_a* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.6.

Table J.6: ***ctx_dp_fpga_ctrl_v1_00_a* EDK peripheral I/O descriptions**

***xps_spi* Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
i_ilb_srqn	FPGA_DP_CTRL_INTN	I	G18	Service Request (Active Low).
i_ilb_miso	FPGA_DP_CTRL_MISO	I	F18	Master-In/Slave-Out Serial Data.
o_ilb_sck	FPGA_DP_CTRL_SCK	O	F20	Serial Clock.
o_ilb_mosi	FPGA_DP_CTRL_MOSI	O	F22	Master-Out/Slave-In Serial Data.
o_ilb_addr_csn	FPGA_DP_CTRL_CSN	O	F21	Address Chip Select (Active Low).
o_ilb_data_csn	FPGA_DP_CTRL_GPIO4	O	E22	Data Chip Select (Active Low).
o_ilb_rnw	FPGA_DP_CTRL_GPIO3	O	D22	Read/nWrite Select.
o_ilb_rstn	FPGA_DP_CTRL_RSTN	O	F19	data path FPGA Reset.

Table J.7: ***ctx_dp_fpga_ctrl_v1_00_a*** **EDK peripheral register map**

Address	**Name**	**R/W**	**Default Value**	**Description**
0	{16'b0,saddr_i[16:31]}	R/W	0x00000000	16-bit SPI Address Register.
1	sdata_i[0:31]	R/W	0x00000000	32-bit SPI Write Data Register.
2	{30'b0,ilb_ctrl[30:31]}	R/W	0x00000002	ILB Control Register. [0:29]: reserved; [30]: rnw_i: Read/Write Select; 0 = Write, 1 = Read. [31]: start_spi_i: Start current SPI transaction. Toggle high-then-low to initiate transaction.
3	slv_reg3[0:31]	R/W	0x00000000	Reserved Register.
4	miso_read_o[0:31]	RO	0x00000000	SPI Read Data Register.
5	{31'b0,txfer_done}	RO	0x00000000	SPI Transfer Done Register. [0:30]: reserved; [31]: txfer_done: SPI Transfer Done (Active High).
6	slv_reg6[0:31]	RO	0xAAAAAAAA	Reserved Register.
7	slv_reg7[0:31]	RO	0x55555555	Reserved Register.

J.4 SPI Configuration Flash PROM Organization

The SPI configuration flash PROM will be organized as shown in Figure J.1. Figure J.1 shows the images stored at adjacent address locations, but in the final implementation the images will be re-aligned to the beginning of a memory sector.

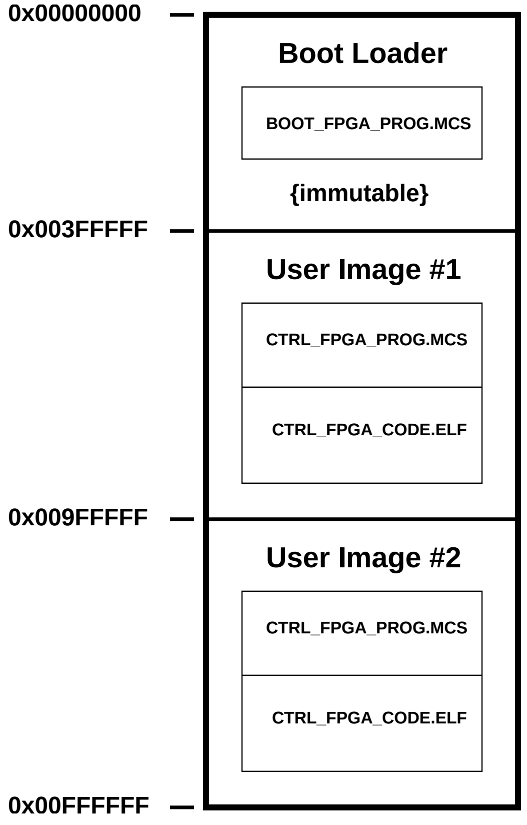

Figure J.1: SPI configuration flash PROM organization

J.4.1 SPI_FLASH: ST Microelectronics M25P64 Flash Prom SPI Controller

The *SPI_FLASH* EDK peripheral is an instance of the *xps_spi* IP and is the main interface to the ST Microelectronics M25P64 Flash Prom. The *SPI_FLASH* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.8.

Table J.8: ***SPI_FLASH*** **EDK peripheral I/O descriptions**

xps_spi **Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
SCK	S3A_CCLK	O	AA20	Spartan-3A Configuration Clock.
SS	S3A_SPI_CSO	O	Y4	Spartan-3A SPI Chip Select (Active Low).
MISO	S3A_SPI_MISO	I	AB20	Spartan-3A SPI Master-In-Slave-Out (MISO).
MOSI	S3A_SPI_MOSI	O	AB14	Spartan-3A SPI Master-Out-Slave-In (MOSI).

J.4.2 GPIO_FLASH: ST Microelectronics M25P64 Flash Prom GPIO Controller

The *GPIO_FLASH* EDK peripheral is used to control the write protect and hold pins of the ST Microelectronics M25P64 Flash PROM. The *GPIO_FLASH* EDK peripheral connects to the Spartan-3A FPGA pins shown in Table J.9.

Table J.9: ***GPIO_FLASH*** **EDK peripheral I/O descriptions**

xps_gpio **Pin #**	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	S3A_SPI_HOLDN	O	AB13	1	Hold Signal (Active Low).
1	S3A_SPI_WPN	O	AA14	1	Write Protect Signal (Active Low).

J.5 AsAP Configuration

The measurement board contains two AsAP processors, which need to be configured at power-up or at receipt of the AsAP configuration command from the command line interface.

The configuration data files for each AsAP will be stored in the microSD card. Each configuration data file will be retrieved upon power-up, and shifted serially to the appropriate AsAP device. See Section J.6 for information on the microSD card directory structure.

The control FPGA will configure each of the AsAP processors using a custom SPI Interface. The custom SPI interface consists of the following 8 signals:

1. SCK: Serial Configuration Clock
2. CSn: Chip Select (Active Low)
3. MOSI: Master-Out/Slave-In Serial Data
4. MISO: Master-In/Slave-Out Serial Data
5. LOAD_EN: Parallel Data Load Enable
6. CFG_CLK: Configuration Clock Pulse
7. CFG_VLD: Configuration Valid Pulse
8. RESET_COLD: Perform a Cold Reset (Active Low)

The outgoing serial data is sent in 20-bit packets/symbols, and the parallel data is formatted as follows:

Table J.10: AsAP Configuration Packet descriptions

par[19]	**par[18]**	**par[17:0]**	**Description**
0b0	0b0	upper_addr	Upper Address Word (config_addr[35:18]).
0b1	0b0	lower_addr	Lower Address Word (config_addr[17:0]).
0b0	0b1	upper_data	Upper Data Word (config_data[35:18]).
0b1	0b1	lower_data	Lower Data Word (config_data[17:0]).

The par[18] bit is used for determining if address or data is being sent by the SPIE to the AsAP processor. If par[18] is a 0, then address is being sent. If par[18] is a 1, then data is being sent.

The par[19] bit is used for determining which part of the word is being sent: upper or lower. If par[19] is a 0, then the upper part of the word is being sent. If par[19] is a 1, then the lower part of the word is being sent.

The configuration blocks inside the AsAP processor know how to interpret the data based on the status of the 2 MSB bits of each 20-bit packet. Each of the 4 serial transactions outlined above are sent with a configuration clock and a configuration valid signal to clock the parallel data into the AsAP processor configuration blocks.

J.5.1 asap_config_v1_00_a: AsAP Configuration Peripheral

The *asap_config_v1_00_a* EDK peripheral is used to configure a single AsAP processor, and appears as a set of 8 software accessible registers to the applications running on the MicroBlaze. The *asap_config_v1_00_a* EDK peripheral's register map is shown in Table J.11.

To initiate a write transaction using the *asap_config_v1_00_a* EDK peripheral first, set the 40-bit address registers (config_addr[39:0]) and 40-bit data registers (config_data[39:0]), then toggle the start_spi bit high then low to initiate the transfer. For subsequent write transactions wait for the send bit to go high. Also, before the config_addr[39:0] and config_data[39:0] registers can be updated, the application must wait for the hold bit to go high. When hold is low, the SPIE state machine is writing the current config_addr[39:0] and config_data[39:0] registers in to the upper_addr, lower_addr, upper_data, and lower_data registers.

To initiate a read transaction using the *asap_config_v1_00_a* EDK peripheral wait for the miso_rdy bit to go high, then read the data from the miso_read[19:0] register.

The *asap_config_0* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.12. The *asap_config_1* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.13.

Table J.11: ***asap_config_v1_00_a* EDK peripheral register map**

Address	**Name**	**R/W**	**Default Value**	**Description**
0	config_data[31:0]	R/W	0x00000000	Lower 32-bits of Configuration Data Register.
1	{24'b0,config_data[39:32]}	R/W	0x00000000	Upper 8-bits of Configuration Data Register.
2	config_addr[31:0]	R/W	0x00000000	Lower 32-bits of Configuration Address Register.
3	{24'b0,config_addr[39:32]}	R/W	0x00000000	Upper 8-bits of Configuration Address Register.
4	{29'b0,spie_cfg[2:0]}	R/W	0x00000006	SPIE Configuration Control Register. [0:28]: reserved; [29]: start_spi: Start SPIE Transaction (Active High). Toggle high-then-low to initiate transaction. [30]: reset_cold: AsAP Cold Reset (Active Low). Toggle low-then-high to initiate transaction. [31]: reset_sel: Reset Selection. 0: 20ns wide re-timed pulse. 1: MicroBlaze pulse high-low-high.
5	slv_reg5[31:0]	R/W	0x00000000	Reserved Register.
6	{29'b0,spie_stat[2:0]}	RO	0x00000000	SPIE Status Register. [0:28]: reserved; [29]: miso_rdy: MISO Read Data Ready (Active High). [30]: send: Ready to Start New Transaction (Active High). [31]: hold: 0: receiving data/addr. 1: transaction in process.
7	{12'b0,miso_read[19:0]}	RO	0x00000000	SPIE MISO Read Data Register.

Table J.12: ***asap_config_0*** **EDK peripheral I/O descriptions**

asap_config_0 **Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
i_spi_asap_miso	FPGA_ASAP1_MISO	I	AA22	Master-In/Slave-Out Serial Data.
o_spi_asap_cfg_clk	FPGA_ASAP1_CFG_CLK	O	W20	Configuration Clock Pulse.
o_spi_asap_cfg_vld	FPGA_ASAP1_CFG_VALID	O	W19	Configuration Valid Pulse.
o_spi_asap_sck	FPGA_ASAP1_SPI_CLK	O	Y21	Serial Configuration Clock.
o_spi_asap_csn	FPGA_ASAP1_SPI_CSN	O	W22	Chip Select (Active Low).
o_spi_asap_mosi	FPGA_ASAP1_SPI_MOSI	O	Y22	Master-Out/Slave-In Serial Data.
o_spi_asap_load_en	FPGA_ASAP1_SPI_LOAD	O	W21	Parallel Data Load Enable (Active High).
o_asap_reset_cold	FPGA_ASAP1_RESET_COLD	O	V19	Perform a Cold Reset (Active Low).
o_asap_rst_cntclk	FPGA_ASAP1_RST_CNTCLK	O	V20	Reset Counter Clock (Active Low).

Table J.13: ***asap_config_1* EDK peripheral I/O descriptions**

***asap_config_1* Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
i_spi_asap_miso	FPGA_ASAP2_MISO	I	W18	Master-In/Slave-Out Serial Data.
o_spi_asap_cfg_clk	FPGA_ASAP2_CFG_CLK	O	AA21	Configuration Clock Pulse.
o_spi_asap_cfg_vld	FPGA_ASAP2_CFG_VALID	O	AB21	Configuration Valid Pulse.
o_spi_asap_sck	FPGA_ASAP2_SPI_CLK	O	AB18	Serial Configuration Clock.
o_spi_asap_csn	FPGA_ASAP2_SPI_CSN	O	AA19	Chip Select (Active Low).
o_spi_asap_mosi	FPGA_ASAP2_SPI_MOSI	O	Y18	Master-Out/Slave-In Serial Data.
o_spi_asap_load_en	FPGA_ASAP2_SPI_LOAD	O	AB19	Parallel Data Load Enable (Active High).
o_asap_reset_cold	FPGA_ASAP2_RESET_COLD	O	AB17	Perform a Cold Reset (Active Low).
o_asap_rst_cntclk	FPGA_ASAP2_RST_CNTCLK	O	AA17	Reset Counter Clock (Active Low).

J.6 microSD Card Organization

The microSD card will be used to store both configuration and waveform data. The microSD card is accessed and controlled using both an SPI interface and a FAT File System, and as such, must abide by the file name, size, and path restrictions. The current microSD card contains 2GB of storage capacity, which means a FAT16 File Sytem will be implemented in the MicroBlaze 32-bit micro-controller residing in the Xilinx Spartan 3A FPGA. The maximum filename length is 8.3, therefore the descriptor can be at most 8 characters and the extension can be at most 3 characters; there is no limit defined on the maximum pathname length. A proposed directory structure and file naming convention is shown below:

Listing J.1: IODELAY Tap Calculation Perl Script

```
ASAP/
        PROG1/
                ASAP1.BIN
                ASAP2.BIN
                RUN.BIN
        PROG2/
                ASAP1.BIN
                ASAP2.BIN
                RUN.BIN
        PROG3/
                ASAP1.BIN
                ASAP2.BIN
                RUN.BIN
CONFIG/
        FACTORY.MCS
        UPGRADE.MCS
        DPIMAGE.BIN
PATTERN/
        BRAM/
        SRAM/
```

J.6.1 SPI_SD: microSD Card SPI Controller

The *SPI_SD* EDK peripheral is an instance of the *xps_spi* IP and is the main interface to the microSD card. The *SPI_SD* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.14.

Table J.14: ***SPI_SD*** **EDK peripheral I/O descriptions**

xps_spi **Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
MOSI	FPGA_SD_TX	O	AB2	Receive Data Input.
SS	FPGA_SD_CARD_DETECT	I/O	AA4	Chip Select (Active Low).
SCK	FPGA_SD_CLK	O	AA3	Serial Clock.
MISO	FPGA_SD_RX	I	AB3	Transmit Data Output.

J.6.2 GPIO_SD: microSD Card GPIO Controller

The *GPIO_SD* EDK peripheral is used to control the busy LED placed next to the microSD card. The Control Application will blink the LED as a warning while the microSD card is being accessed in order to avoid accidental removal and data loss. The *GPIO_SD* EDK peripheral connects to the Xilinx Spartan-3A FPGA pins shown in Table J.15.

Table J.15: ***GPIO_SD*** **EDK peripheral I/O descriptions**

xps_gpio **Pin #**	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_SD_BUSY_LED	O	AB4	1	Busy LED (Active Low).

J.7 Digital-to-Analog Converter

A Texas Instruments DAC5682Z dual-channel, 16-bit, 1 GS/s digital-to-analog converter is used to generate the analog waveforms for the baseband signal source [6]. This DAC must be properly configured before it can be used to generate waveforms. The baseband signal source uses the dual-channel, interpolating DAC in a single-channel, non-interpolating mode. The data path FPGA drives the DAC5682Z with data encoded in two's complement format. Table J.18 describes the configuration data that must be set via the SPI control interface on the DAC5682Z.

J.7.1 SPI_DAC5682Z: TI DAC5682Z DAC SPI Controller

The *SPI_DAC5682Z* EDK peripheral is an instance of the *xps_spi* IP and is the main interface to the DAC5682Z high-speed DAC. The *SPI_DAC5682Z* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.16.

Table J.16: ***SPI_DAC5682Z*** **EDK peripheral I/O descriptions**

xps_spi **Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
SCK	FPGA_DAC5682_SCLK	O	V22	DAC5682Z Serial Clock.
MOSI	FPGA_DAC5682_SDIO	O	U22	DAC5682Z Serial Data Input.
MISO	FPGA_DAC5682_SDO	I	U19	DAC5682Z Serial Data Output.
SS	FPGA_DAC5682_SDENB	O	U21	DAC5682Z Chip Select.

J.7.2 GPIO_DAC5682Z_OUTS: TI DAC5682Z DAC GPIO Controller

The *GPIO_DAC5682Z_OUTS* EDK peripheral is used to control the non-spi pins of the TI DAC5682Z high-speed DAC. The *GPIO_DAC5682Z_OUTS* EDK peripheral connects to the Xilinx Spartan-3A FPGA pins shown in Table J.17.

Table J.17: ***GPIO_DAC5682Z_OUTS*** **EDK peripheral I/O descriptions**

xps_gpio **Pin #**	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_DAC5682_RSTB	O	U20	1	DAC5682Z Reset.

Table J.18: TI DAC5682Z DAC Register Settings

Register Name	**Register Address**	**Value**	**Description**
STATUS0	0x00	0x03	PLL/DLL Locked Status.
CONFIG1	0x01	0x00	DAC Delay, FIR Enable, Self Test, and FIFO Offset Control.
CONFIG2	0x02	0x80	Two's Comp Enable, Dual/Single DAC Mode, and FIR Mode Control.
CONFIG3	0x03	0x00	DAC Offset Enable, A/B Swap, A equal B, SW Sync, and SW Sync Enable.
STATUS4	0x04	0x00	Self Test Error, FIFO Error, and Pattern Error Status.
CONFIG5	0x05	0x92	SPI Setup, Clock Divider, FIFO Offset Sync, and PLL/DLL Bypass Control.
CONFIG6	0x06	0xae	Hold Sync, Sleep, Bias and PLL/DLL Sleep Control.
CONFIG7	0x07	0xff	Channel A and B Gain Control.
CONFIG8	0x08	0x00	DLL Restart Control. Toggle bit 2 high then low.
CONFIG9	0x09	0x00	PLL M/N Control.
CONFIG10	0x0a	0x00	DLL Delay and Clock Control.
CONFIG11	0x0b	0x00	PLL Loop Filter, VCO Divider, PLL Gain and Range Control.
CONFIG12	0x0c	0x00	Offset Sync and Channel A Offset (MSB) Control.
CONFIG13	0x0d	0x00	Channel A Offset (LSB) Control.
CONFIG14	0x0e	0x00	SPI Serial Data Out and Channel B Offset (MSB) Control.
CONFIG15	0x0f	0x00	Channel B Offset (LSB) Control.

J.8 Reference Clock Control

The measurement board contains a Vectron VTC2 10 MHz voltage controlled temperature compensated crystal oscillator (TCXO) for use as a precision reference. The VTC2 is a ±0.5 ppm quartz stabilized, CMOS square wave, temperature compensated oscillator, operating off a 3.3 V supply.

- http://vectron.com/products/tcxo/VTC2.pdf

The internal 10 MHz reference can be used as the reference to the AD9516-3 clock generator IC. If a more precise 10 MHz reference clock is required, an external 10 MHz reference can be provided via the rear-panel connector of the measurement board.

The internal 10 MHz reference is also used to generate a common 312.5 kHz sync clock for the Texas Instruments PTH08T2xxWAZ DC-DC Converters. A Xilinx CPLD (XC9572XL) is employed to perform the divide-by-32 and multi-phase clock generation.

Table J.19: Reference clock control descriptions

Bit	**Description**
0	Internal/External 10 MHz Reference Select 1 = external 0 = internal
1	External 10 MHz Enable 1 = external 0 = internal
2	External 10 MHz Input Disable 1 = external 0 = internal
3	Internal 10 MHz Enable 1 = external 0 = internal

Table J.20: Reference clock control register map

Bits				**Clock Mode Description**
3	**2**	**1**	**0**	
1	1	0	0	Internal 10 MHz Clock Selected **(Default)**
0	0	1	1	External 10 MHz Clock Selected
1	0	1	0	Internal 10 MHz used as reference, External 10 MHz pass to Control FPGA.

The *CLK10MHZ_CTRL_OUTS* EDK peripheral connects to the Xilinx Spartan-3A FPGA pins shown in Table J.21.

Table J.21: ***CLK10MHZ_CTRL_OUTS*** **EDK peripheral I/O descriptions**

xps_gpio **Pin #**	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_CLK10MHZ_REF_CTRL[0]	O	B2	1	10 MHz Ref. Select 1 = int 0 = ext
1	FPGA_CLK10MHZ_REF_CTRL[1]	O	B3	1	10 MHz Ref. Select 1 = int 0 = ext
2	FPGA_CLK10MHZ_REF_CTRL[2]	O	A3	1	10 MHz Ref. Select 1 = int 0 = ext
3	FPGA_CLK10MHZ_REF_CTRL[3]	O	A4	1	10 MHz Ref. Select 1 = int 0 = ext

The *CLK10MHZ_CTRL_INS* EDK peripheral connects to the Xilinx Spartan-3A FPGA pins shown in Table J.22.

Table J.22: ***CLK10MHZ_CTRL_INS* EDK peripheral I/O descriptions**

***xps_gpio* Pin #**	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_EXT_CLK10MHZ_LOS	I	A2	NA	10 MHz Loss of signal 1 = Loss of Signal 0 = Signal Present

J.8.1 INT_CLK_10MHZ_INPUT: Internal 10 MHz Clock Input Buffer

The *INT_CLK_10MHZ_INPUT* EDK peripheral is an instance of the *util_ds_buf* IP and connects to the Xilinx Spartan-3A FPGA pins shown in Table J.24. The *util_ds_buf* IP is essentially an *IBUFGDS* Xilinx Spartan-3A input clock buffer primitive.

Table J.23: ***INT_CLK_10MHZ_INPUT* EDK peripheral I/O descriptions**

***util_ds_buf* Pin #**	**FPGA Pin Name**	**Dir**	**Pin**	**Description**
IBUF_DS_P	FPGA_INT_CLK10MHZ_REF_P	I	A12	Int 10 MHz Clock (Positive).
IBUF_DS_P	FPGA_INT_CLK10MHZ_REF_N	I	A11	Int 10 MHz Clock (Negative).
IBUF_OUT	int_clk10mhz_ref	O	NA	Int 10 MHz Clock.

Listing J.2: Xilinx EDK MHS Internal Clock Input Buffer Instantiation

```
BEGIN util_ds_buf
 PARAMETER INSTANCE = INT_CLK_10MHZ_INPUT
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BUF_TYPE = IBUFGDS
 PORT IBUF_DS_P = FPGA_INT_CLK10MHZ_REF_P
 PORT IBUF_DS_N = FPGA_INT_CLK10MHZ_REF_N
 PORT IBUF_OUT = int_clk10mhz_ref
END
```

J.8.2 EXT_CLK_10MHZ_INPUT: External 10 MHz Clock Input Buffer

The *EXT_CLK_10MHZ_INPUT* EDK peripheral is an instance of the *util_ds_buf* IP and connects to the Xilinx Spartan-3A FPGA pins shown in Table J.24. The *util_ds_buf* IP is essentially an *IBUFGDS* Xilinx Spartan-3A input clock buffer primitive.

Table J.24: ***EXT_CLK_10MHZ_INPUT* EDK peripheral I/O descriptions**

***util_ds_buf* Pin #**	**FPGA Pin Name**	**Dir**	**Pin**	**Description**
IBUF_DS_P	FPGA_EXT_CLK10MHZ_REF_P	I	B11	Ext 10 MHz Clock (Positive).
IBUF_DS_P	FPGA_EXT_CLK10MHZ_REF_N	I	C11	Ext 10 MHz Clock (Negative).
IBUF_OUT	ext_clk10mhz_ref	O	NA	Ext 10 MHz Clock.

Listing J.3: Xilinx EDK MHS External Clock Input Buffer Instantiation

```
BEGIN util_ds_buf
 PARAMETER INSTANCE = EXT_CLK_10MHZ_INPUT
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BUF_TYPE = IBUFGDS
 PORT IBUF_DS_P = FPGA_EXT_CLK10MHZ_REF_P
 PORT IBUF_DS_N = FPGA_EXT_CLK10MHZ_REF_N
 PORT IBUF_OUT = ext_clk10mhz_ref
END
```

J.9 High-Speed Clock Generation

The Analog Devices AD9516 clock generator IC does not require programming during normal use, but does need to be initialized during the normal power-up routines. Once the initialization is complete, the AD9516 can generate the appropriate clock frequencies. Table J.25 shows the generated output frequencies of the active AD9516 Clock Generator IC outputs. Table J.26 shows the default value for each of the AD9516 clock generator IC registers. Writing to the register at 0x232 applies all updates to the AD9516 clock generator IC. The data sheet for the AD9516 can be found here:

- Analog Devices AD9516-3 14-Output Clock Generator with Integrated 2.0 GHz VCO
  - http://www.analog.com/UploadedFiles/Data_Sheets/AD9516_3.pdf

The Universal Microwave Corp UMX Series 1GHz VCO (UMX-244-B14) is used to drive the AD9516-3 External Clock Input.

- http://www.vco1.com/SCDs/scd244-a.pdf

J.9.1 AD9516 Clock Generator Output Assignments

Table J.25: AD9516 clock generator outputs

Output Number	**Description**
0	ADC Sampling Clock (500 MHz).
1	DAC Sampling Clock (500 MHz).
2	FPGA AsAP Clock (250 MHz).
3	FPGA DSP Clock (250 MHz).
4	Unused Output.
5	Unused Output.
6	FPGA SRAM Clock (250 MHz).
7	FPGA SDRAM Clock (250 MHz).
8	FPGA IODELAY Reference Clock (200 MHz).
9	Test Clock Output.

J.9.2 AD9516 Clock Generator PLL Calculations

Equations J.1 and J.2 were used to determine the appropriate values of the PLL parameters.

$$f_{vco} = \left(\frac{f_{ref}}{R}\right) \cdot ((P \cdot B) + A) = f_{ref} \cdot \left(\frac{N}{R}\right) \tag{J.1}$$

$$N = ((P \cdot B) + A) \tag{J.2}$$

The known variables of Equation J.1 are:

- f$_{vco}$ = 1 GHz
- f$_{ref}$ = 10 MHz

Solving for N given R = 1, shows that N should be equal to 100. The next step is to solve for P, B, and A in Equation J.2. First assume that A = 0 and P = 4. Solving for B in Equation J.2 yields 25. A summary of the calculated values is shown below:

- R = 1
- P = 4
- B = 25
- A = 0

J.9.3 AD9516 Clock Generator Registers

Table J.26: AD9516 Clock Generator Register Settings

AD9516 Clock Generator Register Settings		
Register Address	**Value**	**Description**
Serial Port Configuration		
0x000	0xbd	Serial Port Configuration (Reset SPI Registers).
0x000	0x99	Serial Port Configuration (Normal Operation).
0x001	blank	Unused Register
0x002	reserved	Unused Register
0x003	reserved	Unused Register
0x004	0x01	Read Back Control

Continued on Next Page...

AD9516 Clock Generator Register Settings		
Register Address	**Value**	**Description**
PLL		
0x010	0x4c	PFD and Charge Pump
0x011	0x01	R Counter
0x012	0x00	R Counter
0x013	0x00	A Counter
0x014	0x19	B Counter
0x015	0x00	B Counter
0x016	0x03	PLL Control 1
0x017	0x00	PLL Control 2
0x018	0x66	PLL Control 3
0x019	0x00	PLL Control 4
0x01a	0x00	PLL Control 5
0x01b	0x00	PLL Control 6
0x01c	0x07	PLL Control 7
0x01d	0x00	PLL Control 8
0x01e	0x00	PLL Control 9
0x01f	0x00	PLL Readback
0x020 - 0x04f	blank	Unused Registers
Fine Delay Adjust: OUT6 to OUT9		
0x0a0	0x01	OUT6 Delay Bypass
0x0a1	0x00	OUT6 Delay Full-Scale
0x0a2	0x00	OUT6 Delay Fraction
0x0a3	0x01	OUT7 Delay Bypass
0x0a4	0x00	OUT7 Delay Full-Scale
0x0a5	0x00	OUT7 Delay Fraction
0x0a6	0x01	OUT8 Delay Bypass
0x0a7	0x00	OUT8 Delay Full-Scale
0x0a8	0x00	OUT8 Delay Fraction
0x0a9	0x01	OUT9 Delay Bypass
0x0aa	0x00	OUT9 Delay Full-Scale
0x0ab	0x00	OUT9 Delay Fraction
0x0ac - 0x0ef	blank	Unused Registers
LVPECL Outputs		
0x0f0	0x0c	OUT0
0x0f1	0x0c	OUT1
0x0f2	0x08	OUT2
0x0f3	0x08	OUT3
0x0f4	0x03	OUT4
0x0f5	0x03	OUT5

Continued on Next Page. . .

AD9516 Clock Generator Register Settings		
Register Address	**Value**	**Description**
0x0f6 - 0x13f	blank	Unused Registers
LVDS/CMOS Outputs		
0x140	0x02	OUT6
0x141	0x02	OUT7
0x142	0x01	OUT8
0x143	0x02	OUT9
0x144 - 0x18f	blank	Unused Registers
LVPECL Channel Dividers		
0x190	0x00	Divider 0 (PECL)
0x191	0x40	Divider 0 (PECL)
0x192	0x01	Divider 0 (PECL)
0x193	0x44	Divider 1 (PECL)
0x194	0x40	Divider 1 (PECL)
0x195	0x01	Divider 1 (PECL)
0x196	0x00	Divider 2 (PECL)
0x197	0x00	Divider 2 (PECL)
0x198	0x00	Divider 2 (PECL)
LVDS/CMOS Channel Dividers		
0x199	0x00	Divider 3 (LVDS/CMOS)
0x19a	0x00	Divider 3 (LVDS/CMOS)
0x19b	0x00	Divider 3 (LVDS/CMOS)
0x19c	0x08	Divider 3 (LVDS/CMOS)
0x19d	0x01	Divider 3 (LVDS/CMOS)
0x19e	0x32	Divider 4 (LVDS/CMOS)
0x19f	0x00	Divider 4 (LVDS/CMOS)
0x1a0	0x11	Divider 4 (LVDS/CMOS)
0x1a1	0x28	Divider 4 (LVDS/CMOS)
0x1a2	0x01	Divider 4 (LVDS/CMOS)
0x1a3	reserved	Unused Register
0x1a4 - 0x1df	blank	Unused Registers
VCO Divider and CLK Input		
0x1e0	0x02	VCO Divider
0x1e1	0x01	Input CLKs
0x1e2 - 0x22a	blank	Unused Registers
System		
0x230	0x04	Power Down and Sync
0x231	blank/reserved	Unused Register
Update All Registers		
0x232	0x01	Update All Registers

J.9.4 AD9516_SPI: AD9516 Clock Generator SPI Controller

The *AD9516_SPI* EDK peripheral is an instance of the *xps_spi* IP and is the main interface to the configuration pins of the AD9516 clock generator IC. The *AD9516_SPI* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.27.

Table J.27: ***AD9516_SPI*** **EDK peripheral I/O descriptions**

xps_spi **Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
SCK	FPGA_AD9516_SCLK	O	N18	Serial Clock.
SS	FPGA_AD9516_CSN	O	N19	Chip Select (Active Low).
MISO	FPGA_AD9516_SDO	I	P22	Master-In/Slave-Out Serial Data.
MOSI	FPGA_AD9516_SDIO	O	N20	Master-Out/Slave-In Serial Data.

J.9.5 AD9516_GPIO: AD9516 Clock Generator Output GPIO Controller

The *AD9516_GPIO* EDK peripheral is used to control the RESET_N and PD_N pins of the AD9516 clock generator IC. The *AD9516_GPIO* EDK peripheral connects to the Xilinx Spartan-3A FPGA pins shown in Table J.28.

Table J.28: ***AD9516_GPIO*** **EDK peripheral I/O descriptions**

xps_gpio **Pin #**	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_AD9516_RESET_N	O	N21	1	Reset (Active Low).
1	FPGA_AD9516_PD_N	O	P20	1	Power Down Enable.
2	FPGA_AD9516_REF_SEL	O	N22	1	Reference Select.

J.9.6 GPIO_AD9516_INS: ADI AD9516 GPIO Controller

The *GPIO_AD9516_INS* EDK peripheral is used to control the non-spi pins of the Analog Devices AD9516 clock generator IC. The *GPIO_AD9516_OUTS* EDK peripheral connects to the Xilinx Spartan-3A FPGA pins shown in Table J.29.

Table J.29: ***GPIO_AD9516_INS* EDK peripheral I/O descriptions**

***xps_gpio* Pin #**	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_AD9516_LD	I	P19	NA	Lock Detect.
1	FPGA_AD9516_REFMON	I	M17	NA	Reference Monitor.
1	FPGA_AD9516_STATUS	I	N17	NA	Status.

J.10 Instrument Fan Control

The measurement board contains two instrument fan controllers for maintaining proper cooling and airflow, while avoiding the generation of beat frequencies between the two fans. The Texas Instruments AMC6821 [28] is an intelligent temperature monitor and pulse-width modulation (PWM) fan controller. Using either a low-frequency or a high-frequency PWM signal, this device can simultaneously drive a fan, monitor remote sensor diode temperatures, and measure and control the fan speed so that it operates with minimal acoustic noise at the lowest possible speed.

The AMC6821 has three fan control modes:

1. Auto Temperature-Fan mode
2. Software-RPM mode
3. Software-DCY mode

The measurement board uses the AMC6821 in *software-RPM mode*. The software-RPM mode is a closed-loop control. In this mode, the AMC6821 adjusts the PWM output to maintain a consistent fan speed at a user-specified target value; that is, the device functions as a fan speed regulator. Software-RPM mode can also be used to allow the AMC6821 to operate as a stand-alone device.

The I^{2}C address of the two fan controllers is:

1. 0b1001100
2. 0b1001101

J.10.1 AMC6821 Fan Controller #1 Registers

Table J.30: AMC6821 Fan Controller #1 Register Settings

AMC6821 Fan Controller #1 Register Settings			
Register Address	**Value**	**R/W**	**Description**
IDENTIFICATION REGISTERS			
0x3D	0x21	R	Device ID Register
0x3E	0x49	R	Company ID Register
CONFIGURATION REGISTERS			
0x00	0xb5	R/W	Configuration Register 1
0x01	0x3d	R/W	Configuration Register 2
0x3F	0x82	R/W	Configuration Register 3
0x04	0x88	R/W	Configuration Register 4
0x02	0x00	R	Status Register 1
0x03	0x00	R	Status Register 2
TEMPERATURE MONITORING			
0x06	0x00	R	Temp-DATA-LByte
0x0A	0x80	R	Local-Temp-DATA-HByte
0x0B	0x80	R	Remote-Temp-DATA-HByte
0x14	0x3c	R/W	Local-High-Temp-Limit
0x15	0x00	R/W	Local-Low-Temp-Limit
0x16	0x46	R/W	Local-THERM-Limit
0x18	0x50	R/W	Remote-High-Temp-Limit
0x19	0x00	R/W	Remote-Low-Temp-Limit
0x1a	0x64	R/W	Remote-THERM-Limit
0x1b	0x50	R/W	Local-Critical-Temp
0x1c	0x00	R/W	PSV-Temp
0x1d	0x69	R/W	Remote-Critical-Temp
PWM CONTROLLER			
0x20	0x1d	R/W	FAN-Characteristics
0x21	0x55	R/W	DCY-Low-Temp
0x22	0x55	R/W	DCY (Duty Cycle)
0x23	0x52	R/W	DCY-RAMP
0x24	0x41	R/W	Local Temp-Fan Control
0x25	0x61	R/W	Remote Temp-Fan Control
TACH (RPM) MEASUREMENT			
0x08	0x00	R	TACH-DATA-LByte
0x09	0x00	R	TACH-DATA-HByte
0x10	0xff	R/W	TACH-Low-Limit-LByte
0x11	0xff	R/W	TACH-Low-Limit-HByte
			Continued on Next Page. . .

AMC6821 Fan Controller #1 Register Settings			
Register Address	**Value**	**R/W**	**Description**
0x12	0x00	R/W	TACH-High-Limit-LByte
0x13	0x00	R/W	TACH-High-Limit-HByte
0x1E	0xff	R/W	TACH-SETTING-LByte
0x1F	0xff	R/W	TACH-SETTING-HByte
0x3A	0x00	R	Reserved
0x3B	0x00	R	Reserved

J.10.2 ctx_iic_ctrlr_0: I^{2}C Control Peripheral

The *ctx_iic_ctrlr_v1_00_a* EDK peripheral is used to control the I^{2}C fan controller, and appears as a set of 16 software accessible registers to the applications running on the MicroBlaze. The *ctx_iic_ctrlr_v1_00_a* EDK peripheral's register map is shown in Table J.33.

The *ctx_iic_ctrlr_0* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.31.

Table J.31: ***ctx_iic_ctrlr_0* EDK peripheral I/O descriptions**

***ctx_iic_ctrlr_v1_00_a* Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
io_fpga_iic_sda	FPGA_AMC6821_FAN1_SDA	IO	Y7	I^{2}C Serial Data.
io_fpga_iic_scl	FPGA_AMC6821_FAN1_SCK	O	W7	I^{2}C Serial Clock.

The *GPIO_FAN_CTRL1* connects to the Spartan-3A FPGA pins shown in Table J.32.

Table J.32: ***GPIO_FAN_CTRL1* EDK peripheral I/O descriptions**

***xps_gpio* Pin #**	**Pin Name**	**Dir**	**Pin**	**Description**
0	FPGA_AMC6821_FAN1_OVRN	I	T8	Over Temp Flag.
1	FPGA_AMC6821_FAN1_THERMN	I	U7	Thermal Flag.
2	FPGA_AMC6821_FAN1_FAULTN	I	T7	Fault Flag.
3	FPGA_AMC6821_FAN1_SMBALERTN	I	V7	SMB Alert Flag.

Table J.33: ***ctx_iic_ctrlr_v1_00_a*** **EDK peripheral register map**

Address	**Name**	**R/W**	**Default**	**Description**
0	{16'b0,prer[15:0]}	R/W	0x63	I^2C Clock Prescaler Register. [16:31]: prer[15:0]: Clock Prescale.
1	{30'b0,ctrl[1:0]}	R/W	0x0	I^2C Control Register. [30]: core_en: Module Enable. [31]: int_en: Interrupt Enable.
2	{24'b0,txr[7:1],rnw}	R/W	0x0	I^2C Transmit Data Register. [24:30]: txr: Transmit Data. [31]: rnw: Read/Write Bit. 1 = Read. 0 = Write.
3	{26'b0,cmd[5:0]}	R/W	0x0	I^2C Command Register. [26]: iack: Interrupt Acknowledge. Clears a pending interrupt. [27]: ack: ACK = '0' or NACK = '1'. [28]: wr: Write to Slave. [29]: rd: Read from Slave. [30]: sto: Generate Stop Condition. [31]: sta: Generate Start Condition.
4	{31'b0,rst}	R/W	0x0	I^2C Reset Register. [31]: rst: I^2C Module Reset.. Toggle to reset module.
8	{24'b0,rxr[7:0]}	RO	0x0	I^2C Receive Data Register.
9	{26'b0,stat[4:0]}	RO	0x0	I^2C Status Register. [27]: rxack: Received ACK from Slave. 1 = No ACK Received. 0 = ACK Received. [28]: busy: Busy. 1 after START detected. 0 after STOP detected. [29]: al: Arbitration Lost. Arbitration is lost when: STOP detected, but not requested. master drives SDA high, but it is low. [30]: tip: Transfer in Progress. 1 = Transferring Data. 0 = Transfer Complete. [31]: irq_flag: Interrupt Flag. Set when interrupt is pending.

J.10.3 AMC6821 Fan Controller #2 Registers

Table J.34: AMC6821 Fan Controller #2 Register Settings

AMC6821 Fan Controller #2 Register Settings			
Register Address	**Value**	**R/W**	**Description**
IDENTIFICATION REGISTERS			
0x3D	0x21	R	Device ID Register
0x3E	0x49	R	Company ID Register
CONFIGURATION REGISTERS			
0x00	0xb5	R/W	Configuration Register 1
0x01	0x3d	R/W	Configuration Register 2
0x3F	0x82	R/W	Configuration Register 3
0x04	0x88	R/W	Configuration Register 4
0x02	0x00	R	Status Register 1
0x03	0x00	R	Status Register 2
TEMPERATURE MONITORING			
0x06	0x00	R	Temp-DATA-LByte
0x0A	0x80	R	Local-Temp-DATA-HByte
0x0B	0x80	R	Remote-Temp-DATA-HByte
0x14	0x3c	R/W	Local-High-Temp-Limit
0x15	0x00	R/W	Local-Low-Temp-Limit
0x16	0x46	R/W	Local-THERM-Limit
0x18	0x50	R/W	Remote-High-Temp-Limit
0x19	0x00	R/W	Remote-Low-Temp-Limit
0x1a	0x64	R/W	Remote-THERM-Limit
0x1b	0x50	R/W	Local-Critical-Temp
0x1c	0x00	R/W	PSV-Temp
0x1d	0x69	R/W	Remote-Critical-Temp
PWM CONTROLLER			
0x20	0x25	R/W	FAN-Characteristics
0x21	0x55	R/W	DCY-Low-Temp
0x22	0x55	R/W	DCY (Duty Cycle)
0x23	0x52	R/W	DCY-RAMP
0x24	0x41	R/W	Local Temp-Fan Control
0x25	0x61	R/W	Remote Temp-Fan Control
TACH (RPM) MEASUREMENT			
0x08	0x00	R	TACH-DATA-LByte
0x09	0x00	R	TACH-DATA-HByte
0x10	0xff	R/W	TACH-Low-Limit-LByte
0x11	0xff	R/W	TACH-Low-Limit-HByte
			Continued on Next Page...

AMC6821 Fan Controller #2 Register Settings			
Register Address	**Value**	**R/W**	**Description**
0x12	0x00	R/W	TACH-High-Limit-LByte
0x13	0x00	R/W	TACH-High-Limit-HByte
0x1E	0xff	R/W	TACH-SETTING-LByte
0x1F	0xff	R/W	TACH-SETTING-HByte
0x3A	0x00	R	Reserved
0x3B	0x00	R	Reserved

J.10.4 ctx_iic_ctrlr_1: I^{2}C Control Peripheral

The *ctx_iic_ctrlr_v1_00_a* EDK peripheral is used to control the I^{2}C fan controller, and appears as a set of 16 software accessible registers to the applications running on the MicroBlaze. The *ctx_iic_ctrlr_v1_00_a* EDK peripheral's register map is shown in Table J.37.

The *ctx_iic_ctrlr_1* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.35.

Table J.35: ***ctx_iic_ctrlr_1* EDK peripheral I/O descriptions**

***ctx_iic_ctrlr_v1_00_a* Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
io_fpga_iic_sda	FPGA_AMC6821_FAN2_SDA	IO	Y9	I^{2}C Serial Data.
io_fpga_iic_scl	FPGA_AMC6821_FAN2_SCK	O	AB9	I^{2}C Serial Clock.

The *GPIO_FAN_CTRL2* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.36.

Table J.36: ***GPIO_FAN_CTRL2* EDK peripheral I/O descriptions**

***xps_gpio* Pin #**	**Pin Name**	**Dir**	**Pin**	**Description**
0	FPGA_AMC6821_FAN2_OVRN	I	V9	Over Temp Flag.
1	FPGA_AMC6821_FAN2_THERMN	I	V8	Thermal Flag.
2	FPGA_AMC6821_FAN2_FAULTN	I	U8	Fault Flag.
3	FPGA_AMC6821_FAN2_SMBALERTN	I	T9	SMB Alert Flag.

Table J.37: ***ctx_iic_ctrlr_v1_00_a*** **EDK peripheral register map**

Address	**Name**	**R/W**	**Default**	**Description**
0	{16'b0,prer[15:0]}	R/W	0x63	I^2C Clock Prescaler Register. [16:31]: prer[15:0]: Clock Prescale.
1	{30'b0,ctrl[1:0]}	R/W	0x0	I^2C Control Register. [30]: core_en: Module Enable. [31]: int_en: Interrupt Enable.
2	{24'b0,txr[7:1],rnw}	R/W	0x0	I^2C Transmit Data Register. [24:30]: txr: Transmit Data. [31]: rnw: Read/Write Bit. 1 = Read. 0 = Write.
3	{26'b0,cmd[5:0]}	R/W	0x0	I^2C Command Register. [26]: iack: Interrupt Acknowledge. Clears a pending interrupt. [27]: ack: ACK = '0' or NACK = '1'. [28]: wr: Write to Slave. [29]: rd: Read from Slave. [30]: sto: Generate Stop Condition. [31]: sta: Generate Start Condition.
4	{31'b0,rst}	R/W	0x0	I^2C Reset Register. [31]: rst: I^2C Module Reset.. Toggle to reset module.
8	{24'b0,rxr[7:0]}	RO	0x0	I^2C Receive Data Register.
9	{26'b0,stat[4:0]}	RO	0x0	I^2C Status Register. [27]: rxack: Received ACK from Slave. 1 = No ACK Received. 0 = ACK Received. [28]: busy: Busy. 1 after START detected. 0 after STOP detected. [29]: al: Arbitration Lost. Arbitration is lost when: STOP detected, but not requested. master drives SDA high, but it is low. [30]: tip: Transfer in Progress. 1 = Transferring Data. 0 = Transfer Complete. [31]: irq_flag: Interrupt Flag. Set when interrupt is pending.

J.11 Temperature Sensing

The measurement board contains 6 digital temperature sensors in a 2x3 array. The Texas Instruments TMP125 [29] digital temperature sensor is accurate to 2°C over a temperature range of −25°C to +85°C, and is controlled using a serial peripheral interface. The temperature measurement is made with a 10-bit resolution delta-Σ analog to digital converter, which translates to a temperature resolution of 0.25°C. A block diagram of the TMP125 temperature sensor is shown in Figure J.2.

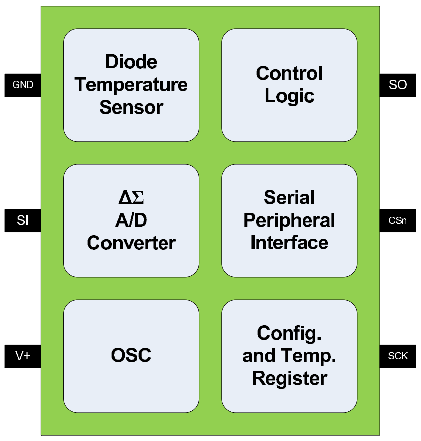

Figure J.2: 2°C accurate digital temperature sensor with SPI interface

An application running on the MicroBlaze 32-bit Soft-Core Processor will periodically sample the temperature of all nine sensors to determine if a temperature calibration is required for the various high-speed devices. Upon characterization of the measurement board during normal operation within a chassis, a set of temperature limits will be defined that denote when a recalibration is necessary. It is possible that there may be only three temperature zones:

1. Too Cold!!
2. Normal Operation
3. Too Hot!!

However, more zones may be necessary depending on the environment and performance limitations of the high-speed devices.

The temperature register of the TMP125 is a 16-bit, read-only register that stores the output of the most recent conversion. However, temperature is represented by only 10-bits, which are in signed two's complement format. The first bit of the temperature register, D15, is a leading zero. Bits D14 to D5 are used to indicate temperature. Bits D4 to D0 are the same as D5 (see Table J.38). Data format for temperature is summarized in Table J.39. When calculating the signed two's complement temperature value, only the 10 data bits should be used.

Following power-up or reset, the temperature register will read 0°C until the first conversion is complete.

Table J.38: TMP125 Temperature Register

D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
0	T9	T8	T7	T6	T5	T4	T3	T2	T1	T0	T0	T0	T0	T0	T0

Table J.39: TMP125 temperature data format

TMP125 Temperature Data Format	
Temperature (°C)	**DIGITAL OUTPUT (T9...T0)**
+127	0b01_1111_1100
+125	0b01_1111_0100
+100	0b01_1001_0000
+75	0b01_0010_1100
+50	0b00_1100_1000
+25	0b00_0110_0100
+10	0b00_0010_1000
+0.25	0b00_0000_0001
0	0b00_0000_0000
-0.25	0b11_1111_1111
-25	0b11_1001_1100
-50	0b11_0011_1000
-55	0b11_0010_0100

J.11.1 TMP_SENSE_1A: TI TMP125 Temperature Sensor SPI Controller

The *TMP_SENSE_1A* EDK peripheral is an instance of the *xps_spi* IP and is the main interface to the temperature sensor #1(A). The *TMP_SENSE_1A* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.40.

Table J.40: ***TMP_SENSE_1A* EDK peripheral I/O descriptions**

***xps_spi* Pin Name**	**Pin Name**	**Direction**	**Pin**	**Description**
SCK	FPGA_TMPSENS_1A_SCK	O	AB7	Sensor 1A Serial Clock.
MOSI	FPGA_TMPSENS_1A_MOSI	O	AB8	Sensor 1A MOSI.
MISO	FPGA_TMPSENS_1A_MISO	I	AB6	Sensor 1A MISO.
SS	FPGA_TMPSENS_1A_CSN	O	AA8	Sensor 1A Chip Select.

J.11.2 TMP_SENSE_1B: TI TMP125 Temperature Sensor SPI Controller

The *TMP_SENSE_1B* EDK peripheral is an instance of the *xps_spi* IP and is the main interface to the temperature sensor #1(B). The *TMP_SENSE_1B* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.41.

Table J.41: ***TMP_SENSE_1B* EDK peripheral I/O descriptions**

***xps_spi* Pin Name**	**Pin Name**	**Direction**	**Pin**	**Description**
SCK	FPGA_TMPSENS_1B_SCK	O	F7	Sensor 1B Serial Clock.
MOSI	FPGA_TMPSENS_1B_MOSI	O	F8	Sensor 1B MOSI.
MISO	FPGA_TMPSENS_1B_MISO	I	E6	Sensor 1B MISO.
SS	FPGA_TMPSENS_1B_CSN	O	E7	Sensor 1B Chip Select.

J.11.3 TMP_SENSE_1C: TI TMP125 Temperature Sensor SPI Controller

The *TMP_SENSE_1C* EDK peripheral is an instance of the *xps_spi* IP and is the main interface to the temperature sensor #1(C). The *TMP_SENSE_1C* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.42.

Table J.42: ***TMP_SENSE_1C*** **EDK peripheral I/O descriptions**

xps_spi **Pin Name**	**Pin Name**	**Direction**	**Pin**	**Description**
SCK	FPGA_TMPSENS_1C_SCK	O	V17	Sensor 1C Serial Clock.
MOSI	FPGA_TMPSENS_1C_MOSI	O	W17	Sensor 1C MOSI.
MISO	FPGA_TMPSENS_1C_MISO	I	U16	Sensor 1C MISO.
SS	FPGA_TMPSENS_1C_CSN	O	Y17	Sensor 1C Chip Select.

J.11.4 TMP_SENSE_2A: TI TMP125 Temperature Sensor SPI Controller

The *TMP_SENSE_2A* EDK peripheral is an instance of the *xps_spi* IP and is the main interface to the temperature sensor #2(A). The *TMP_SENSE_2A* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.43.

Table J.43: ***TMP_SENSE_2A*** **EDK peripheral I/O descriptions**

xps_spi **Pin Name**	**Pin Name**	**Direction**	**Pin**	**Description**
SCK	FPGA_TMPSENS_2A_SCK	O	R19	Sensor 2A Serial Clock.
MOSI	FPGA_TMPSENS_2A_MOSI	O	R20	Sensor 2A MOSI.
MISO	FPGA_TMPSENS_2A_MISO	I	R18	Sensor 2A MISO.
SS	FPGA_TMPSENS_2A_CSN	O	R21	Sensor 2A Chip Select.

J.11.5 TMP_SENSE_2B: TI TMP125 Temperature Sensor SPI Controller

The *TMP_SENSE_2B* EDK peripheral is an instance of the *xps_spi* IP and is the main interface to the temperature sensor #2(B). The *TMP_SENSE_2B* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.44.

Table J.44: ***TMP_SENSE_2B*** **EDK peripheral I/O descriptions**

xps_spi **Pin Name**	**Pin Name**	**Direction**	**Pin**	**Description**
SCK	FPGA_TMPSENS_2B_SCK	O	J21	Sensor 2B Serial Clock.
MOSI	FPGA_TMPSENS_2B_MOSI	O	J22	Sensor 2B MOSI.
MISO	FPGA_TMPSENS_2B_MISO	I	J20	Sensor 2B MISO.
SS	FPGA_TMPSENS_2B_CSN	O	K22	Sensor 2B Chip Select.

J.11.6 TMP_SENSE_2C: TI TMP125 Temperature Sensor SPI Controller

The *TMP_SENSE_2C* EDK peripheral is an instance of the *xps_spi* IP and is the main interface to the temperature sensor #2(C). The *TMP_SENSE_2C* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.45.

Table J.45: ***TMP_SENSE_2C*** **EDK peripheral I/O descriptions**

xps_spi **Pin Name**	**Pin Name**	**Direction**	**Pin**	**Description**
SCK	FPGA_TMPSENS_2C_SCK	O	C17	Sensor 2C Serial Clock.
MOSI	FPGA_TMPSENS_2C_MOSI	O	B17	Sensor 2C MOSI.
MISO	FPGA_TMPSENS_2C_MISO	I	D17	Sensor 2C MISO.
SS	FPGA_TMPSENS_2C_CSN	O	A17	Sensor 2C Chip Select.

J.12 DDR2 SDRAM SODIMM I^{2}C Memory Interface

The measurement board contains a DDR2 SDRAM SODIMM connector for use with memory of a capacity up to 2 GB. The SODIMM contains an I^{2}C memory for storing information related to the control and refresh of the DDR2 SDRAM. The SODIMM data interface is connected to the data path FPGA, and the I^{2}C control interface is connected to the control FPGA.

J.12.1 ctx_iic_ctrlr_2: I^{2}C Control Peripheral

The *ctx_iic_ctrlr_v1_00_a* EDK peripheral is used to control the DDR2 SDRAM SODIMM I^{2}C interface, and appears as a set of 16 software accessible registers to the applications running on the MicroBlaze. The *ctx_iic_ctrlr_v1_00_a* EDK peripheral's register map is shown in Table J.47.

The *ctx_iic_ctrlr_2* connects to the Xilinx Spartan-3A FPGA pins shown in Table J.46.

Table J.46: ***ctx_iic_ctrlr_2*** **EDK peripheral I/O descriptions**

ctx_iic_ctrlr_v1_00_a **Pin Name**	**Pin Name**	**Dir**	**Pin**	**Description**
io_fpga_iic_sda	FPGA_DDR2_SDRAM_SDA	IO	B15	I^{2}C Serial Data.
io_fpga_iic_scl	FPGA_DDR2_SDRAM_SCL	O	A15	I^{2}C Serial Clock.

Table J.47: ***ctx_iic_ctrlr_v1_00_a*** **EDK peripheral register map**

Address	**Name**	**R/W**	**Default**	**Description**
0	{16'b0,prer[15:0]}	R/W	0x63	I^{2}C Clock Prescaler Register. [16:31]: prer[15:0]: Clock Prescale.
1	{30'b0,ctrl[1:0]}	R/W	0x0	I^{2}C Control Register. [30]: core_en: Module Enable. [31]: int_en: Interrupt Enable.
2	{24'b0,txr[7:1],rnw}	R/W	0x0	I^{2}C Transmit Data Register. [24:30]: txr: Transmit Data. [31]: rnw: Read/Write Bit. 1 = Read. 0 = Write.

Continued on Next Page...

Address	**Name**	**R/W**	**Default**	**Description**
3	{26'b0,cmd[5:0]}	R/W	0x0	I^{2}C Command Register. [26]: iack: Interrupt Acknowledge. Clears a pending interrupt. [27]: ack: ACK = '0' or NACK = '1'. [28]: wr: Write to Slave. [29]: rd: Read from Slave. [30]: sto: Generate Stop Condition. [31]: sta: Generate Start Condition.
4	{31'b0,rst}	R/W	0x0	I^{2}C Reset Register. [31]: rst: I^{2}C Module Reset.. Toggle to reset module.
8	{24'b0,rxr[7:0]}	RO	0x0	I^{2}C Receive Data Register.
9	{26'b0,stat[4:0]}	RO	0x0	I^{2}C Status Register. [27]: rxack: Received ACK from Slave. 1 = No ACK Received. 0 = ACK Received. [28]: busy: Busy. 1 after START detected. 0 after STOP detected. [29]: al: Arbitration Lost. Arbitration is lost when: STOP detected, but not requested. master drives SDA high, but it is low. [30]: tip: Transfer in Progress. 1 = Transferring Data. 0 = Transfer Complete. [31]: irq_flag: Interrupt Flag. Set when interrupt is pending.

J.13 Debug Peripherals

The measurement board contains several peripherals intended to be used during debug of both Hardware and Software.

- 3 Push-Buttons (Active Low)
- 4 LEDs (Active Low)
- Logic Analyzer Header
- CP2102 USB-to-UART Bridge
- FT245BL USB Interface
- FPGA Re-Program Push-Button

J.13.1 Buttons_3Bit: Push-Button GPIO Controller

The measurement board control FPGA has three momentary push-button for debugging various applications and hardware. The push-buttons can be polled independently. A logic low indicates a push-button has been pressed, and the push-buttons idle high when not pressed. The push-buttons are controlled by the MicroBlaze via a GPIO peripheral.

Figure J.3: Momentary Push-Button

The push-buttons connect to the Xilinx Spartan-3A FPGA pins shown in Table J.48.

Table J.48: ***Buttons_3Bit* EDK peripheral I/O descriptions**

***xps_gpio* Pin #**	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_PUSHBUTTONS[0]	I	H9	NA	Push-Button 0
1	FPGA_PUSHBUTTONS[1]	I	G8	NA	Push-Button 1
2	FPGA_PUSHBUTTONS[2]	I	G7	NA	Push-Button 2

J.13.2 LEDs_4Bit: LED GPIO Controller

The measurement board control FPGA has four LEDs for debugging various applications and hardware. The LEDs can be controlled independently, and are turned on by setting the appropriate control bit to a logic low. The LEDs are controlled by the MicroBlaze via a GPIO peripheral.

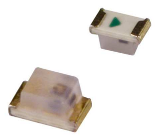

Figure J.4: Light-Emitting Diode

The LEDs connect to the Xilinx Spartan-3A FPGA pins shown in Table J.49.

Table J.49: ***LEDs_4Bit*** **EDK peripheral I/O descriptions**

xps_gpio **Pin #**	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_LEDS[0]	O	D8	1	LED Bit 0 (Active Low).
1	FPGA_LEDS[1]	O	C7	0	LED Bit 1
2	FPGA_LEDS[2]	O	C8	1	LED Bit 2
3	FPGA_LEDS[3]	O	B8	0	LED Bit 3

J.13.3 Logic Analyzer

The measurement board contains a header for probing internal FPGA signals with a Logic Analyzer. The pinout for the 18-pin Logic Analyzer header is shown in Table J.50.

Table J.50: Logic analyzer header pinout

Odd Pins		**Even Pins**	
FPGA_LA_CLK	1	2	GND
FPGA_LA_DATA[7]	3	4	GND
FPGA_LA_DATA[6]	5	6	GND
FPGA_LA_DATA[5]	7	8	GND
FPGA_LA_DATA[4]	9	10	GND
FPGA_LA_DATA[3]	11	12	GND
FPGA_LA_DATA[2]	13	14	GND
FPGA_LA_DATA[1]	15	16	GND
FPGA_LA_DATA[0]	17	18	GND

The LEDs connect to the Xilinx Spartan-3A FPGA pins shown in Table J.51.

Table J.51: **Logic analyzer I/O descriptions**

Pin Name	**Dir**	**Pin**	**Default Value**	**Description**
FPGA_LA_CLK	O	A14	NA	Logic Analyzer Clock.
FPGA_LA_DATA[7]	O	A13	NA	Logic Analyzer Data Bit 7.
FPGA_LA_DATA[6]	O	B13	NA	Logic Analyzer Data Bit 6.
FPGA_LA_DATA[5]	O	C13	NA	Logic Analyzer Data Bit 5.
FPGA_LA_DATA[4]	O	D15	NA	Logic Analyzer Data Bit 4.
FPGA_LA_DATA[3]	O	D13	NA	Logic Analyzer Data Bit 3.
FPGA_LA_DATA[2]	O	E13	NA	Logic Analyzer Data Bit 2.
FPGA_LA_DATA[1]	O	E14	NA	Logic Analyzer Data Bit 1.
FPGA_LA_DATA[0]	O	F13	NA	Logic Analyzer Data Bit 0.

J.13.4 CP2102 USB-to-UART Bridge

The measurement board contains a Silicon Laboratories CP2102 USB-to-UART Bridge for controlling the board in a development environment in lieu of the full-speed USB interface. The CP2102 provides a COM Port Interface over USB, and allows for an RS-232 interface between the data path FPGA and the CP2102. The MicroBlaze design will use an Xilinx EDK XPS UART Lite peripheral, which will be configured as shown in Table J.13.4.

C_BAUDRATE	115200
C_DATA_BITS	8
C_USE_PARITY	0
C_ODD_PARITY	0

The desired terminal settings are shown in Table J.13.4.

Bits per second:	115200
Data bits:	8
Parity:	None
Stop bits:	1
Flow Control:	None

The CP2102 connects to the Xilinx Spartan-3A FPGA pins shown in Table J.53.

Table J.52: CP2102 I/O descriptions

CP2102 Pin Name	**FPGA Pin Name**	**Pin**	**Description**
TXD	FPGA_RS232_RX	AB5	RS-232 Receive Data to FPGA.
RXD	FPGA_RS232_TX	Y5	RS-232 Transmit Data from FPGA.
RTS	FPGA_RS232_RTS	AA6	RS-232 Request to Send to FPGA.
CTS	FPGA_RS232_CTS	Y6	RS-232 Clear to Send from FPGA.

In order to use the CP2102, a driver must be installed on the computer. The driver can be downloaded from the following Silicon Laboratories website:

- https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx

J.13.4.1 RS232_CP2102: RS-232 UartLite Controller

The *RS232_CP2102* EDK peripheral is an instance of the *xps_uartlite* and connects to the Xilinx Spartan-3A FPGA pins shown in Table J.53.

Table J.53: ***RS232_CP2102*** **EDK peripheral I/O descriptions**

xps_uartlite **Pin #**	**FPGA Pin Name**	**Dir**	**Pin**	**Description**
RX	FPGA_RS232_RX	I	AB6	Receive Data to FPGA.
TX	FPGA_RS232_TX	O	Y5	Transmit Data from FPGA.

J.13.4.2 CP2102_RTS: RS-232 UartLite Controller

The *CP2102_RTS* EDK peripheral connects to the Xilinx Spartan-3A FPGA pins shown in Table J.54.

Table J.54: ***CP2102_RTS*** **EDK peripheral I/O descriptions**

xps_gpio **Pin #**	**FPGA Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_RS232_RTS	I	AA6	NA	Request to Send to FPGA.

J.13.4.3 CP2102_CTS: RS-232 UartLite Controller

The *CP2102_CTS* EDK peripheral connects to the Xilinx Spartan-3A FPGA pins shown in Table J.55.

Table J.55: ***CP2102_CTS*** **EDK peripheral I/O descriptions**

xps_gpio **Pin #**	**FPGA Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_RS232_CTS	O	Y6	0	Clear to Send from FPGA.

J.13.5 FTDI FT245BL USB Interface

The measurement board control FPGA has access to a Future Technology Devices International Ltd. FT245BL USB first-in/first-out (FIFO) IC. This USB FIFO can provide higher data throughput than the Silicon Laboratories CP2102 USB-to-UART bridge IC, and may be used as the main communications interface for the General Purpose Instrument.

In order to use the FT245BL USB FIFO IC, a driver must be installed on the computer. FTDI Chip provides two kinds of drivers:

- **Virtual COM Port Drivers**
  http://www.ftdichip.com/Drivers/VCP.htm
- **D2XX Drivers**
  http://www.ftdichip.com/Drivers/D2XX.htm

For higher throughput the D2XX Drivers should be used. If the Virtual COM Port driver is used, then similar data throughput to the CP2102 will be achieved.

The FT245BL USB FIFO IC connects to the Xilinx Spartan-3A FPGA pins shown in Table J.56.

Table J.56: ***FT245BL_GPIO*** **EDK peripheral I/O descriptions**

xps_gpio **Pin #**	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
0	FPGA_FTDI_DATA[0]	I/O	V15	1	Data Bit 0.
1	FPGA_FTDI_DATA[1]	I/O	V16	0	Data Bit 1.
2	FPGA_FTDI_DATA[2]	I/O	W15	1	Data Bit 2.
3	FPGA_FTDI_DATA[3]	I/O	W16	0	Data Bit 3.
4	FPGA_FTDI_DATA[4]	I/O	Y15	1	Data Bit 4.
5	FPGA_FTDI_DATA[5]	I/O	Y16	0	Data Bit 5.
6	FPGA_FTDI_DATA[6]	I/O	AB15	1	Data Bit 6.
7	FPGA_FTDI_DATA[7]	I/O	AB16	0	Data Bit 7.
8	FPGA_FTDI_TXEN	I	P12	0	TX Data Enable.
9	FPGA_FTDI_RXFN	I	R12	0	RX Data Valid.
10	FPGA_FTDI_PWRENN	I	R13	0	Power Enable.
11	FPGA_FTDI_RSTOUTN	I	R14	0	Reset Output.
12	FPGA_FTDI_RDN	O	U13	0	Read Enable.
13	FPGA_FTDI_WRN	O	V14	0	Write Enable.
14	FPGA_FTDI_SI_WU	O	W13	0	Send Immediate and Wake Up Signal.

J.13.6 FPGA Re-Program Push-Button

The measurement board has the ability to precisely control the re-configuration process for the control FPGA. Figure J.5 shows the circuit used on the measurement board. Section J.14 describes how the TPS3823-33DBV device works at power-up along with the manual reset input. Unlike the board reset circuit whose Logical AND gate *A* input is driven by *Config Done*, the PROG_B circuit's *A* input is driven an external connector interface for future controller boards. This input provides a controller board with the capability to initiate an FPGA re-configuration during either the General Purpose Instrument power-up routine or a firmware upgrade.

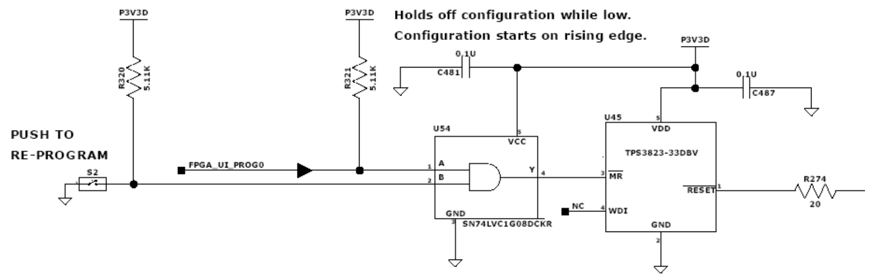

Figure J.5: Measurement board PROG_B circuit

J.13.7 Reach Technologies Display

The measurement board contains an RS-232 interface to a Reach Technologies Display for displaying waveforms and configuring the instrument. The MicroBlaze design will use an EDK XPS UART Lite peripheral, which will be configured as shown in Table J.13.7.

C_BAUDRATE	115200
C_DATA_BITS	8
C_USE_PARITY	0
C_ODD_PARITY	0

The desired terminal settings are shown in Table J.13.7.

Bits per second:	115200
Data bits:	8
Parity:	None
Stop bits:	1
Flow Control:	None

The Reach Technologies Display connects to the Xilinx Spartan-3A FPGA pins shown in Table J.57.

Table J.57: CP2102 I/O descriptions

CP2102 Pin Name	**FPGA Pin Name**	**Pin**	**Description**
TXD	FPGA_RS232_RX	V12	RS-232 Receive Data to FPGA.
RXD	FPGA_RS232_TX	U12	RS-232 Transmit Data from FPGA.

J.14 Board Reset Push-Button

There are several methods to reset the measurement board. A Texas Instruments processor supervisory circuit (TSP3823-33DBV) is used to provide both a power-up and manual reset. During power-on, $\overline{RESET}$ is asserted when supply voltage +3.3 V becomes higher than 1.1 V. Thereafter, the supply voltage supervisor monitors +3.3 V and keeps $\overline{RESET}$ active as long as +3.3 V remains below the threshold V_{IT}. An internal timer delays the return of the output to the inactive state (high) to ensure proper system reset. The delay time, t_d, starts after +3.3 V has risen above the threshold voltage V_{IT}. When the supply voltage drops below the threshold voltage V_{IT}, the output becomes active (low) again. The threshold voltage of the TPS3823-33DBV is 2.93 V.

The TPS3823-33DBV device incorporates a manual reset input, $\overline{MR}$. A low level at $\overline{MR}$ causes $\overline{RESET}$ to become active. The measurement board does not take advantage of the watch dog circuit available in the TPS3823-33DBV. A truth table for the TPS3823-33DBV is shown in Table J.58.

Table J.58: TPS3823-33DBV truth table

TPS3823-33DBV Truth Table		
INPUTS		**OUTPUTS**
$\overline{MR}$	$\mathbf{V}_{DD} > \mathbf{V}_{IT}$	$\overline{RESET}$
0	0	0
0	1	0
1	0	0
1	1	1

As +3.3 V powers-on the TPS3823 will initiate a power-on reset to the control FPGA, but the FPGA will most likely be busy in the configuration process. Therefore, the reset will be missed by the FPGA. However, an external logic AND gate is provided to logically *AND* an active-low push-button and the FPGA Configuration Done (Active High) signal (see Table J.59), so that the $\overline{MR}$ pin is toggled low then high. This results in $\overline{RESET}$ being toggled low for 1 μs, which will provide plenty of time for the FPGA to properly reset after the configuration process completes. After the measurement board has powered up and the FPGA is configured, the user can press the push-button to initiate an FPGA reset whenever desired.

Table J.59: Local reset truth table

Local Reset Truth Table		
Config Done	**Push-Button**	**Local Reset**
0	0	0
0	1	0
1	0	0
1	1	1

During normal operation, the future controller board can also initiate a board reset by toggling the controller reset pin low then high. The truth table for the board reset is shown in Table J.60.

Table J.60: Board reset truth table

Board Reset Truth Table		
Local Reset	**Controller Reset**	**Board Reset**
0	0	0
0	1	0
1	0	0
1	1	1

The board reset signal is fanned out to 3 devices on the measurement board:

- Control FPGA
- Power Control CPLD
- Data Path FPGA

The circuit shown in Figure J.6 allows for all three programmable devices to be simultaneously reset.

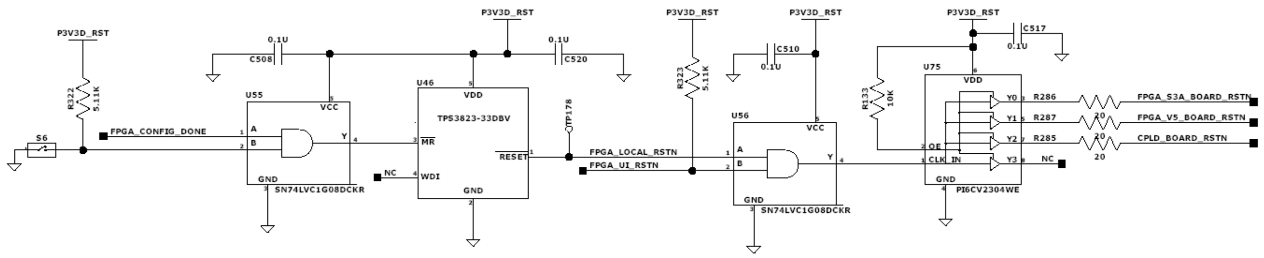

Figure J.6: Measurement board reset circuit

The reset signals connect to the Xilinx Spartan-3A FPGA pins shown in Table J.61.

Table J.61: Reset Signal descriptions

Pin Name	**Dir**	**Pin**	**Description**
FPGA_S3A_BOARD_RSTN	I	D7	Measurement Board Main Reset.
FPGA_LOCAL_RSTN	I	D6	Measurement Board Local Reset.

J.14.1 proc_sys_reset_0: Processor System Reset Controller

The *proc_sys_reset* EDK peripheral is in charge of receiving all reset and DCM locked signals in order to determine when the processor reset should be driven.

Listing J.4: Xilinx EDK MHS System Reset Instantiation

```
BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT Slowest_sync_clk = sys_clk_s
 PORT Dcm_locked = dcm0_locked
 PORT Ext_Reset_In = sys_rst_s
 PORT MB_Reset = mb_reset
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT MB_Debug_Sys_Rst = Debug_SYS_Rst
END
```

The *proc_sys_reset* EDK peripheral connects to the Xilinx Spartan-3A FPGA pins shown in Table J.62.

Table J.62: ***proc_sys_reset_0* EDK peripheral I/O descriptions**

Peripheral Name	**Pin Name**	**Dir**	**Pin**	**Default Value**	**Description**
Ext_Reset_In	FPGA_S3A_BOARD_RSTN	I	D7	1	Board Main Reset.

J.15 CLK_100MHZ_INPUT: Differential Clock Input Buffer

The *CLK_100MHZ_INPUT* EDK peripheral is an instance of the *util_ds_buf* IP and connects to the Xilinx Spartan-3A FPGA pins shown in Table J.63. The *util_ds_buf* IP is essentially an *IBUFGDS* Xilinx Spartan-3A input clock buffer primitive.

Table J.63: ***CLK_100MHZ_INPUT* EDK peripheral I/O descriptions**

***util_ds_buf* Pin #**	**FPGA Pin Name**	**Dir**	**Pin**	**Description**
IBUF_DS_P	FPGA_CLK100MHZ_P	I	AA12	100MHz Clock (Positive).
IBUF_DS_P	FPGA_CLK100MHZ_N	I	AB12	100MHz Clock (Negative).
IBUF_OUT	dcm_clk_s	O	NA	100MHz Clock.

Listing J.5: Xilinx EDK MHS System Clock Input Buffer Instantiation

```
BEGIN util_ds_buf
 PARAMETER INSTANCE = CLK_100MHZ_INPUT
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BUF_TYPE = IBUFGDS
 PORT IBUF_DS_P = fpga_0_CLK_100_P
 PORT IBUF_DS_N = fpga_0_CLK_100_N
 PORT IBUF_OUT = dcm_clk_s
END
```

J.16 dcm_module_0: Digital Clock Module

The *dcm_module_0* EDK Peripheral is an instance of the *dcm_module* IP and is used to generate the clocks shown in Table J.64. The *util_ds_buf* IP is essentially a Xilinx Spartan-3A digital clock manager (*DCM*).

Table J.64: ***dcm_module_0*** **EDK Peripheral I/O Descriptions**

util_ds_buf **Pin #**	**FPGA Pin Name**	**Dir**	**Description**
CLKIN	dcm_clk_s	I	100 MHz Clock.
CLK0	DDR_SDRAM_mpmc_clk_s	O	DDR SDRAM 100 MHz Clock, 0°.
CLK90	DDR_SDRAM_mpmc_clk_90_s	O	DDR SDRAM 100 MHz Clock, 90°.
CLKDV	sys_clk_s	O	MicroBlaze 50 MHz System Clock.
CLKFB	DDR_SDRAM_mpmc_clk_s	O	DCM Feedback CLock Clock.
LOCKED	dcm0_locked	O	DCM Locked (Active High).

Listing J.6: Xilinx EDK MHS DCM Instantiation

```
BEGIN dcm_module
 PARAMETER INSTANCE = dcm_module_0
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_CLKDV_DIVIDE = 2.0
 PARAMETER C_CLKIN_PERIOD = 10.0
 PARAMETER C_EXT_RESET_HIGH = 0
 PARAMETER C_CLKIN_BUF = FALSE
 PARAMETER C_CLKFB_BUF = FALSE
 PARAMETER C_CLK0_BUF = TRUE
 PARAMETER C_CLK90_BUF = TRUE
 PARAMETER C_CLKDV_BUF = TRUE
 PORT RST = net_vcc
 PORT CLKIN = dcm_clk_s
 PORT CLK0 = DDR_SDRAM_mpmc_clk_s
 PORT CLK90 = DDR_SDRAM_mpmc_clk_90_s
 PORT CLKFB = DDR_SDRAM_mpmc_clk_s
 PORT CLKDV = sys_clk_s
 PORT LOCKED = dcm0_locked
END
```

Glossary

ADC :

An acronym used to refer to a *analog-to-digital converter*, which converts a analog voltages to binary values.

AsAP :

Acronym for *Asynchronous Array of Simple Processors.* A 2D-mesh parallel array architecture designed for power efficiency while executing computationally intensive applications.

AsAP Version 1 :

This is the first implementation of the `AsAP` architecture which has 36 processing elements arranged in a 6x6 array with one input in the upper-left corner and one output which can be any one of the right edge processors. This version of the architecture uses nearest neighbor communication exclusively.

AsAP Version 2 :

This is the second implementation of the `AsAP` architecture which has an array of size 13x13 with a few of the lower processors replaced by hardware-based accelerators. For this work the array is assumed to be homogeneous with a size of 16x16. This version of the architecture introduces a routing overlay network and also allows the input processor to be any one of the left edge processors.

CW :

Acronym for *continuous waveform*, which is an RF signal of constant amplitude and frequency.

Core Laminate :

An insulating material with copper affixed to both sides [30]. The majority of the copper is etched away during the printed circuit board manufacturing process. The remaining copper makes up the signal traces and power planes.

Core PCB Construction :

Core construction combines all core laminate materials using a prepreg during the multilayer lamination process [31].

DAC :

An acronym used to refer to a *digital-to-analog converter*, which converts binary values to analog voltages.

dBFS :

A measure of a DAC output signals power level in decibels relative to full scale. A power level of 0 dBFS represents the maximum possible level of a device. The output of DAC devices are commonly scaled to 0 dBFS, −6 dBFS, and −12 dBFS when measuring SFDR.

DDR :

An acronym used to refer to a data transfer mechanism known as *double data rate*, which transfers data on both the rising and falling edges of a clock signal.

DDS :

An acronym used to refer to a *direct digital synthesizer*, which is made up of a phase accumulator and phase to amplitude converter to generate periodic waveforms. For example, sine, cosine, square and triangle waveforms.

DSP :

An acronym used to refer to a *digital signal processor*, or the act of digitally processing a signal.

DUT :

An acronym used to refer to a *Device Under Test* in a measurement system.

ENOB :

Effective number of bits is a measure in units of bits of a converter's performance as compared to the theoretical limit based on quantization noise.

Foil PCB Construction :

Construction using a core laminate material for all internal layers [31]. The outer layer is made up of a copper foil.

HD_2 :

Second harmonic distortion, or HD_2, is the ratio of the amplitude of the second harmonic to the amplitude of the fundamental tone. On a dB scale, HD_2 increases linearly with a slope of one in terms of the output power.

HD_3 :

Third harmonic distortion, or HD_3, is the ratio of the amplitude of the third harmonic to the amplitude of the fundamental tone. On a dB scale, HD_3 increases linearly with a slope of two in terms of the output power.

Input/output block (IOB) :

A collection or grouping of basic elements that make up the input and output functions of FPGA devices. For example, the `IOB` of a Xilinx Virtex-5 FPGA is made up of an input buffer, a tri-state output buffer, an inverter, and a pad [17].

Instrument Height :

Test and measurement equipment height is generally identified as a multiple of `U`s. Each `U` is equivalent to 1.75 inches. The actual height of an instrument which is 1U tall is 1.719 inches.

Instrument Width :

Test and measurement equipment is typically designed to fit in a 19 inch rack mounted system. A piece of equipment is generally less than 19 inches and is mounted in the rack using ear brackets on both sides.

IMD_3 :

Third-order two-tone intermodulation distortion is a metric used to describe the distortion performance of a transmitter or receiver when multiple signal tones are present in the data stream. It is measured by driving two spectrally pure sine waves through the DUT at frequencies f_1 and f_2, usually relatively close together. The amplitude of each tone when summed together will be approximately 6 dB below full-scale of the converter in order to avoid clipping. It is typically specified in dBc relative to the value of either of the two input tones [11].

Nyquist Frequency :

The sample rate divided by two ($\frac{F_s}{2}$) is known as the Nyquist Frequency.

Nyquist Sampling Theorem :

The Nyquist sampling theorem states that in order to perfectly reconstruct a signal the system must sample at a rate greater than 2B, where B represents the highest frequency in the original signal.

Nyquist Zones :

The frequency range from DC (0 Hz) to $\frac{F_s}{2}$ is called the first Nyquist zone. The second Nyquist zone is known as the frequency range from $\frac{F_s}{2}$ to F_s.

One-hot encoding :

One-hot encoding refers to a group of bits where only a single bit is set to a logic high. All remaining bits are set to a logic low. This type of encoding is commonly used to represent the state transitions of a finite state machine. An example of a one-hot encoded group of 3 bits is shown in Table J.65.

001
010
100

Table J.65: One-hot encoding example

Prepreg Laminate :

An insulating material which is inserted between the etched printed circuit board layers [30]. This material acts as the glue that bonds the multiple layers of the PCB.

QDR-II SRAM :

A type of synchronous random-access memory (RAM), which provides a DDR data interface capable of simultaneous data reads and writes, hence the use of the term quad-data-rate (QDR). The simultaneous read and write interface is made possible by two separate 36-bit data interfaces. The address bus is shared between the read and write interfaces.

RBW :

The analog or digital intermediate frequency (IF) section consists of resolution bandwidth filters, which follow the IF gain amplifier of a spectrum analyzer [12].

SFDR :

Spurious-Free Dynamic Range is a measure of a signal tone relative to the largest spurious signal present in the signal bandwidth being measured. It is commonly used to specify the performance of DAC and ADC devices.

SPAN :

Spectrum analyzer use the term SPAN to define the frequency range displayed on screen.

TOI :

Third-order intercept point is a metric used to describe how well a transmitter or receiver performs with closely spaced signals. It is calculated using the IMD_3 and fundamental tone power parameters and is specified in terms of dB.

VBW :

Spectrum analyzers use video filters to help discern signals which are close to the noise level. The video filter will effectively smooth or average the displayed signal on screen. It is a low-pass filter that comes after the envelope detector of a spectrum analyzer, and determines the bandwidth of the video signal that will eventually be digitized to yield amplitude data. The corner frequency of the video filter can be varied to increase or decrease the video bandwidth [12].

X2Y capacitor :

An X2Y capacitor is a type of capacitor developed by a company named X2Y Attenuators, LLC.

Bibliography

[1] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, M. Singh, and B. Baas. An asynchronous array of simple processors for DSP applications. In *IEEE International Solid-State Circuits Conference (ISSCC)*, volume 49, pages 428–429, 663, February 2006.

[2] Zhiyi Yu, Michael Meeuwsen, Ryan Apperson, Omar Sattari, Michael Lai, Jeremy Webb, Eric Work, Tinoosh Mohsenin, and Bevan Baas. Architecture and evaluation of an asynchronous array of simple processors. *Journal of VLSI Signal Processing Systems*, 53(3):243–259, March 2008.

[3] H. Zhang, V. Prabhu, V. George, et al. A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing. *IEEE Journal of Solid-State Circuits (JSSC)*, 35(11):1697–1704, November 2000.

[4] D. N. Truong, W. H. Cheng, T. Mohsenin, Z. Yu, A. T. Jacobson, G. Landge, M. J. Meeuwsen, A. T. Tran, Z. Xiao, E. W. Work, J. W. Webb, P. Mejia, and B. M. Baas. A 167-processor computational platform in 65 nm CMOS. *IEEE Journal of Solid-State Circuits (JSSC)*, 44(4):1130–1144, April 2009.

[5] Xilinx Inc. Virtex-5 FPGA PCB designer's guide. Website, 2009. http://www.xilinx.com/support/documentation/user_guides/ug193.pdf.

[6] Texas Instruments. High-speed 16-bit 1gs/s digital-to-analog converter dac5682z data sheet. Website, 2009. http://focus.ti.com/lit/ds/symlink/dac5682z.pdf.

[7] P. P. Vaidyanathan. *Multirate Systems and Filter Banks*. Prentice-Hall, Upper Saddle River, NJ, 1993.

[8] Texas Instruments. Ultra-wideband current-feedback operational amplifer opa695 data sheet. Website, 2009. http://focus.ti.com/lit/ds/symlink/opa695.pdf.

[9] Agilent Technologies Inc. *What is the difference between an equivalent time sampling oscilloscope and a real-time oscilloscope? (Application Note 1608)*. Palo Alto, CA, USA, 2008.

[10] Anritsu. Intermodulation distortion measurements using the 37300 series vector network analyzer. Website, 2000. http://www.us.anritsu.com/downloads/files/11410-00257a.pdf.

[11] Walt Kester. Intermodulation distortion considerations for adcs. Website, 2008. http://www.analog.com/static/imported-files/tutorials/MT-012.pdf.

[12] Agilent Technologies Inc. *Spectrum Analyzer Basics (Application Note 150)*. Palo Alto, CA, USA, 2005.

[13] Maxim Integrated Products. Application note 3716: Folded-frequency calculator. Website, 2005. http://pdfserv.maxim-ic.com/en/an/AN3716.pdf.

[14] A.T. Jacobson, D.N. Truong, and B.M. Baas. The design of a reconfigurable continuous-flow mixed-radix FFT processor. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1133–1136, May 2009.

[15] Texas Instruments. Wideband fixed-gain operational amplifer ths4302 data sheet. Website, 2006. http://focus.ti.com/lit/ds/symlink/ths4302.pdf.

[16] Texas Instruments. Wideband low-noise low-distortion fully-differential operational amplifer ths4509 data sheet. Website, 2009. http://focus.ti.com/lit/ds/symlink/ths4509.pdf.

[17] Xilinx Inc. Virtex-5 FPGA user guide. Website, 2009. http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

[18] Xilinx Inc. Virtex-5 FPGA xtremedsp design considerations user guide. Website, 2009. http://www.xilinx.com/support/documentation/user_guides/ug203.pdf.

[19] Istvan Novak. Comparison of power distribution network design methods: Bypass capacitor selection based on time domain and frequency domain performances. In *TecForum at DesignCon*, TF-MP3, 2006.

[20] Stephen H. Hall, Garrett W. Hall, and James A. McCall. *High-speed digital system design: a handbook of interconnect theory and design practices.* John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158-0012, 2000.

[21] V. Ricchiuti. Power-supply decoupling on fully populated high-speed digital PCBs. *IEEE Transactions on Electromagnetic Compatibility*, 43(4):671–676, November 2001.

[22] Rogers Corporation. RO4000 high frequency laminate with ticer foil. Technical report, Rogers Corporation, 2007. http://www.rogerscorp.com/documents/726/acm/RO-laminates-data-sheet-and-fabrication-guidelines-RO-B.aspx.

[23] Isola Group. Isola FR408 product bulletin. Technical report, Isola Group, 2009. http://www.isola-group.com/images/file/DSFR408010510doc.pdf.

[24] Kester Peter Biocca. Developing a reliable lead-free smt assembly process. Website, 2007. http://www.kester.com/en-US/documentation/Developing%20a%20Reliable%20Lead-free%20SMT%20Process%20.pdf.

[25] H. Johnson and M. Graham. *High-Speed Digital Design: A Handbook of Black Magic*. Prentice-Hall, Upper Saddle River, NJ, 1993.

[26] Xilinx Inc. Virtex-5 FPGA data sheet: DC and switching characteristics. Website, 2009.

[27] Xilinx Inc. Memory interface generator user's guide. Website, 2009.

[28] Texas Instruments. Intelligent temperature monitor and pwm fan controller. Website, 2007. http://focus.ti.com/lit/ds/symlink/amc6821.pdf.

[29] Texas Instruments. Two degree c accurate digital temperature sensor with spi interface. Website, 2005. http://focus.ti.com/lit/ds/symlink/tmp125.pdf.

[30] H. Johnson and M. Graham. *High-Speed Signal Propagation: Advanced Black Magic*. Prentice-Hall, Upper Saddle River, NJ, 2002.

[31] Merix. Designing balanced printed circuit boards. Technical report, Merix, 2009. http://www.merix.com/files/Designing%20Balanced%20PCBs.pdf.

Index