A new programmable low noise all digital phase-locked loop architecture

by

Justin L. Gaither

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Marwan Hassoun, Major Professor
Doug Jacobson
Robert Weber
Gurpur Prabhu

Iowa State University

Ames, Iowa

2005

Graduate Collage
Iowa State University

This is to certify that the master's thesis of
Justin L. Gaither
has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

Table of Contents

List of Figures

List of Tables

Acknowledgements

I wish to give special thanks to Brian Brunn for his assistance and mentorship while I worked on this project, as well as Xilinx, Inc for supporting the development, design and testing.

I would also like to recognize my wife Patricia and my children Brittany and Hanna for their sacrifices and encouragement while I worked to complete my degree.

Abstract

In the electronics industry today almost without exception there are phase-locked loops (PLL) implemented within each system and often within each integrated circuit (IC). In fact, most PLL's are implemented monolithically within ICs without any or with very few external components. Additionally, most are implemented as Analog PLL's utilizing only a digital phase detector. This is also evident in the majority of recent publications which focus on PLL structures with on-chip voltage controlled oscillators using charge pumps and ring or LC oscillators. However, the problem with most on-chip VCO's is that they are far noisier than the external crystal types. The noise in the integrated oscillators forces designers to use larger loop bandwidths than would be required with less noisy VCO's; subsequently they have poor noise filtering capabilities. Additionally, analog PLL's are usually fixed in nature. Loop components such as charge-pumps and loop filters are implemented as analog components with little or no flexibility. The focus of this thesis is the design and implementation of a very low cost, low noise Programmable All Digital PLL (ADPLL) which utilizes a low cost digital to analog converter (DAC), a voltage controlled crystal oscillator (VCXO), and a field programmable gate array (FPGA). The use of FPGA technology for digital design implementation is universal in the industry and provides benefits far beyond the implementation of ADPLL's. In fact, in almost every system today, an FPGA already exists. Therefore, the inclusion of a DPLL within existing system components would be at little or no cost. The implementation of the PLL digitally not only allows us to implement it within an FPGA, but also allows us to adapt and configure the PLL for many applications and tune it for best performance. Digital circuits also have increased noise margin and are not affected by the same noise issues associated with Analog PLL's such as temperature, voltage and noise coupled from other signals or circuits. The DPLL developed is flexible and can be configured to operate as a clock and data recovery circuit (CDR), clock multiplier, clock synthesizer, or noise filtering PLL. Using an external VCXO provides a very low noise basis for the PLL and such that we can implement very low bandwidths without sacrificing the quality of its output. In this thesis we will present the theory, architecture, design, hardware and implementation of the ADPLL in addition to the results of the testing of the prototype ADPLL that was built.

Chapter 1 Introduction

1.1 Introduction

This chapter provides preliminary information and background on Phased-Locked Loop (PLL) design, architectures, and components. Furthermore the specific implementation that this thesis contributes will be introduced. Theoretical analysis of the implementation will also be discussed.

1.2 Overview of Various PLL Architectures and their Applications

Phase-Locked Loops are used in many applications, and are as wide spread as any other circuit type[1]. In the earliest history they were used to sync the horizontal and vertical sweeps in television[1]. Today, they are used as clock multipliers in high performance microprocessors such as Intel Pentium 4. As such their benefits and properties have been studied and documented in countless journal papers, books, and articles. In fact there exists so much information about PLL's it can be overwhelming and difficult to isolate the specific information a researcher may be searching.

A basic Phase-Locked Loop consists of three basic components, a phase detector, a loop filter, and a voltage or current controlled oscillator.

1. The phase detector can be considered the brain. It makes the decisions about the behavior of the loop. Often it is a non-linear device whose output contains the phase or frequency difference between its reference and the controlled oscillator.
2. The loop filter can be considered the muscle. It controls how much impact the decision of the brain or phase detector has on the controlled oscillator. Its properties are what keep the loop stable and affect the performance.
3. The voltage or current controlled oscillator is the heart. It is central to the loop and its periodic nature keeps the loop ticking. The frequency of its output is dependent upon a much lower frequency and lower amplitude signal.

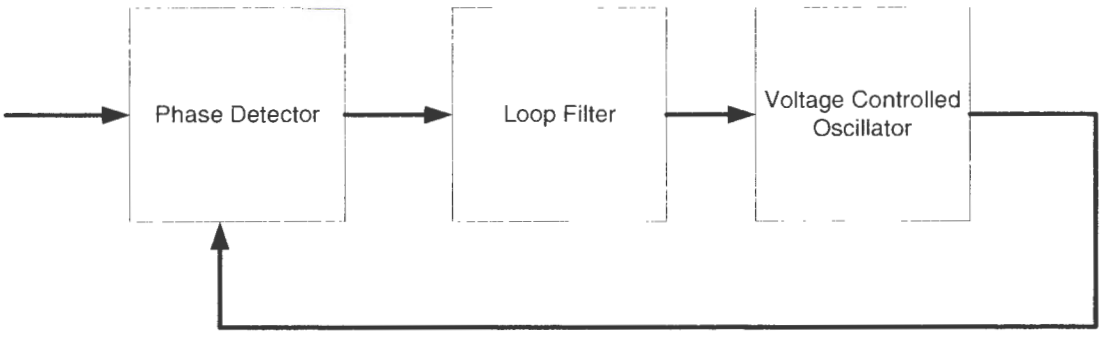

Figure 1 : Simple Phase-Locked Loop Diagram

Figure 1 shows how these components connect in a feedback architecture to create a simple phase-locked loop. However, as with any feedback architecture there must exist a balance within the loop to maintain stability and performance. The classic analysis of the PLL requires certain assumptions be made such as linearity[1]. Both the phase detector and the controlled oscillator have non-linear operating points. Luckily they also have linear or near-linear operating points when differences in phase are small. This allows analysis to be performed in the frequency domain using Laplace transformations.

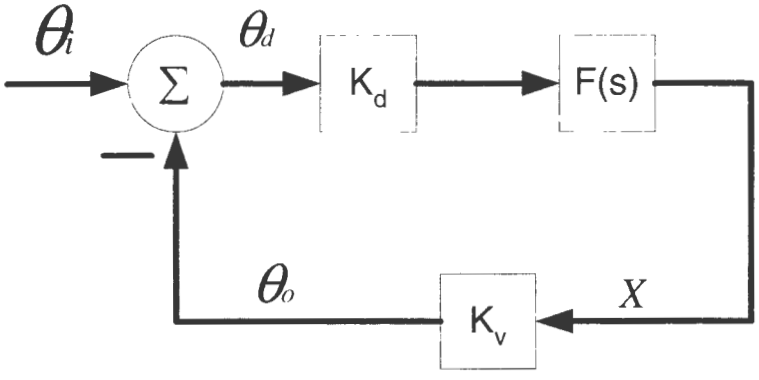

Figure 2 : Conceptual diagram of Linear PLL

Figure 2 shows a conceptual diagram of a linear PLL[1]. This linear PLL is described in the frequency domain by the equation (1).

$$T(s) = \frac{\theta_o(s)}{\theta_i(s)} = \frac{s}{s + K_d K_v F(s)} \quad (1)$$

The earliest PLL's, such as the one shown in Figure 1, were completely analog. The phase detectors where mixers, the loop filters consisted of active amplifiers and low pass filters, and the voltage controlled oscillator consisted of crystals and amplifiers. Engineers have evolved the PLL into many different variants. The classic digital PLL (DPLL) is mostly analog, but replaces the phase detector with a digital implementation [1][4][7][16][18]. Next, the All Digital PLL (ADPLL) which replaced the loop filter with a digital equivalent and the voltage or current controlled oscillator with a numerically controlled oscillator[1][5]. There also exists a Software PLL which utilizes a microprocessor and Analog to Digital Converters (ADC) to over-sample the signals and use floating point or integer math to perform the phase detection and the loop filter portions of the loop[1]. Let's examine some of these architectures in more detail.

1.2.1 Classic Digital PLL

The classic digital PLL's used mostly analog components, however the phase-detector is replaced with digital gates, and often there exists a divider between the VCO and the phase detector. This enables the VCO to operate at higher rate than the reference and still maintain synchronization. There also exists a Charge Pump which is used to convert the digital outputs to an analog current or voltage[16]. This is a common architecture used today for both integrated and discrete PLL's[18]. The key benefits of this type of approach are its simplicity and more predictable linear behavior. However, this type of PLL is not flexible and adding programmability usually results in degraded performance because of extra loading on analog circuits.

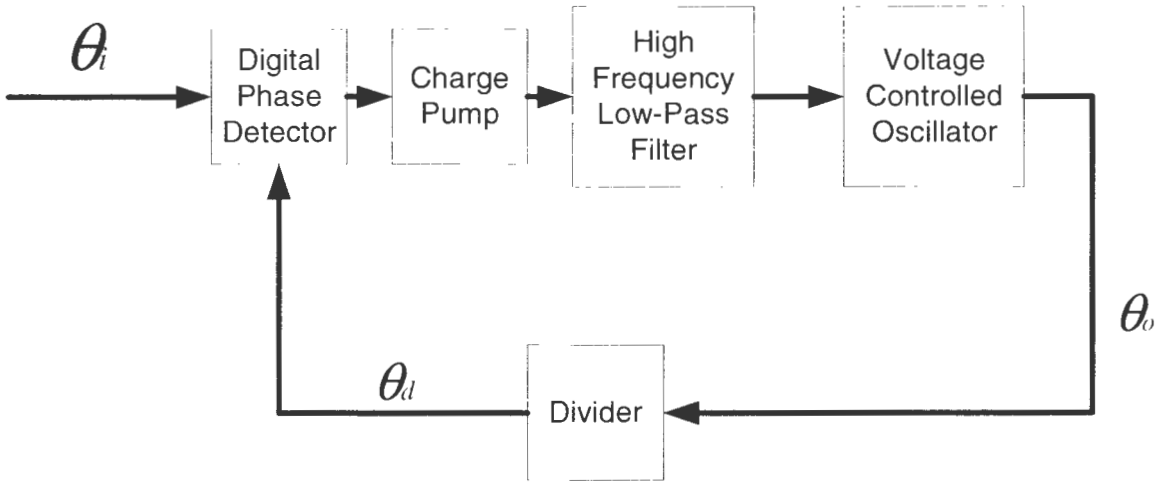

Figure 3 : Digital PLL Diagram

1.2.2 All-Digital PLL

The All-Digital PLL extends the digital circuits to include digital loop filter and numerically controlled oscillator. Often the digital loop filters include digital counters, multipliers, dividers, or other DSP type structures[7][14]. The numerically controlled oscillator takes as an input some number of binary bits of data and produces a square wave output at a specific frequency corresponding to the value of the input[5].

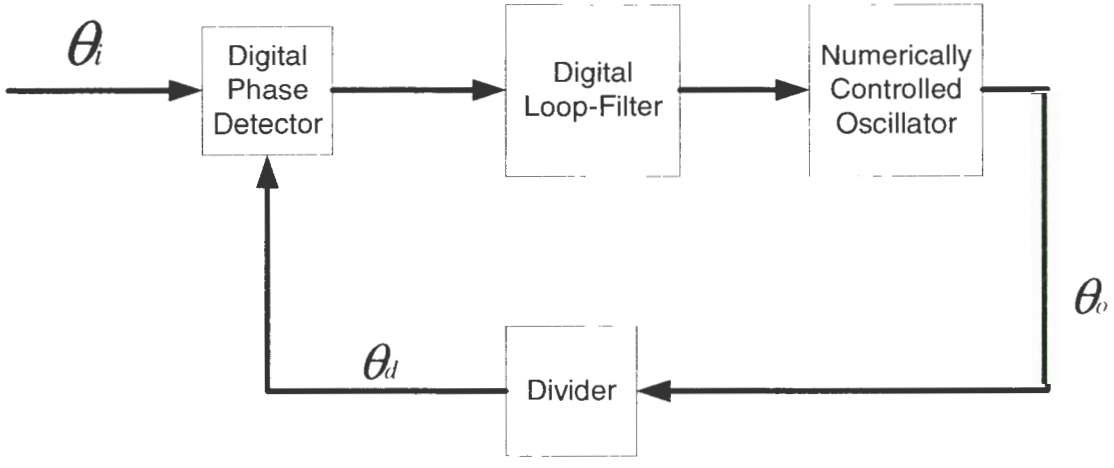

Figure 4 : All-Digital PLL Diagram

The primary benefits of this type of loop is that the majority of its components are digital. This provides simplicity in implementation and portability to may applications. Additionally digital circuits are easier to translate to different processes and scale well with Moore's Law. Programmability is also more easily added because digital circuits are not as susceptible to degradation because of loading. However, the ADPLL also behaves in a more non-linear manner. This results in the need for more difficult analysis techniques.

1.2.3 Software PLL

Software PLL's usually uses sampled data and standard mathematical functions for phase detector and loop filters[1]. This type of PLL uses a microprocessor with arithmetic computation units which operate at rates greater than the oscillator and reference. The benefits of such implementations are very low bandwidth operation, flexibility of loop parameters, and potentially very low noise. However a SPLL must operate at much slower rates because of the limits of the microprocessor.

1.3 Overview of Various PLL Components

1.3.1 Digital phase detectors

The phase detector is a critical component of any PLL and in modern PLL architecture the digital phase detector is by far the most prevalent. This section briefly discusses four common types of digital phase detectors.

XOR Based Phase Detector:

The simplest digital phase detector consists of only an XOR gate[1].

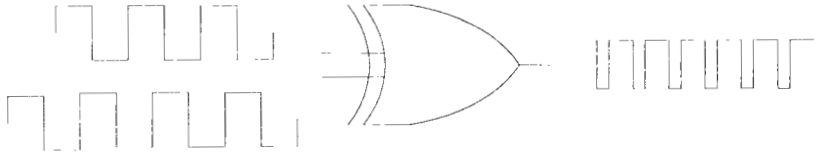

Figure 5 : XOR Based Phase Detector

The output is a series of pulses with varying pulse widths. When locked the output has a 50% duty cycle. This type of phase detector integrates well with analog loop filters and is commonly used in the classic digital PLL's. The loop filter acts to create a RMS voltage from the series of pulses. In locked condition the reference and the oscillator are 90 degrees out of phase.

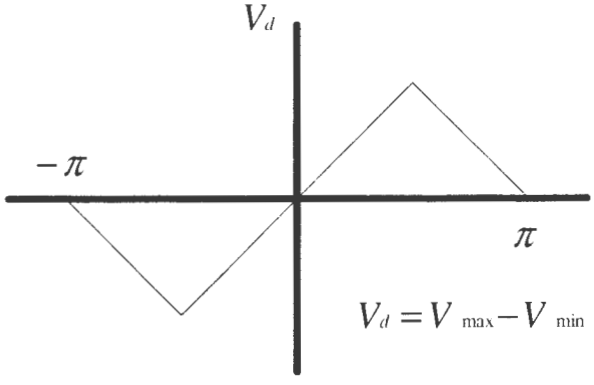

Figure 6 : XOR PD Response

It has a response as shown in Figure 6. The linear operating region is $-\pi/2$ to $\pi/2$.

Phase Frequency Detector:

The Phase-Frequency Detector has 2 outputs which indicate Up and Down changes in frequency required. This detector expands the linear region to 4π as shown in Figure 8. Also its outputs are dependent only on the rising edges of the inputs. This eliminates dependencies on the duty cycle of the inputs. This phase detector is widely used in fully integrated classic digital PLL's because its outputs allow more direct connection to charge pump circuits[1][16-18] and its large linear region.

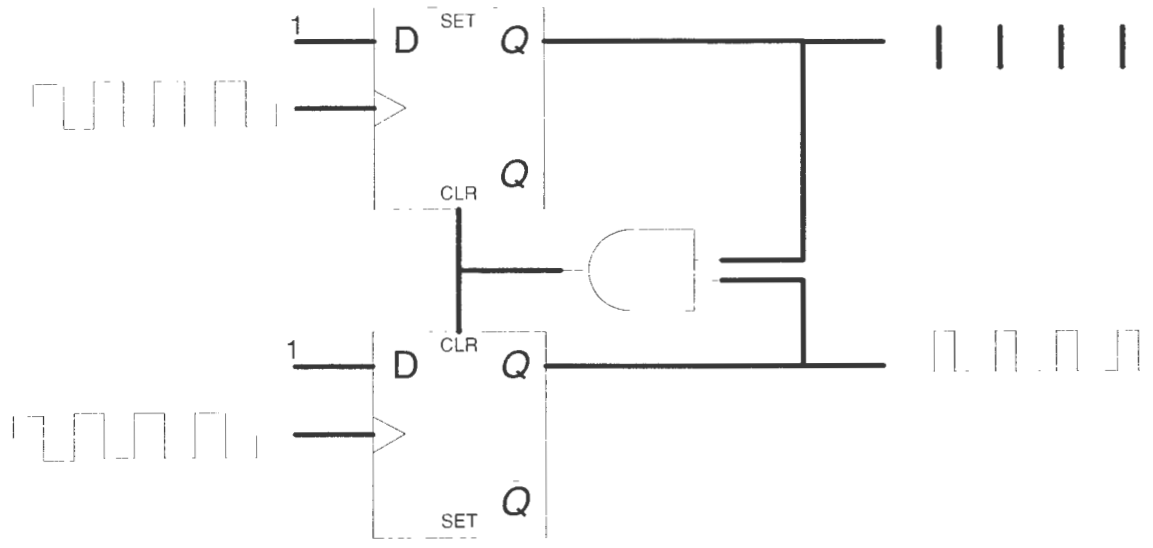

Figure 7 : Phase-Frequency Detector

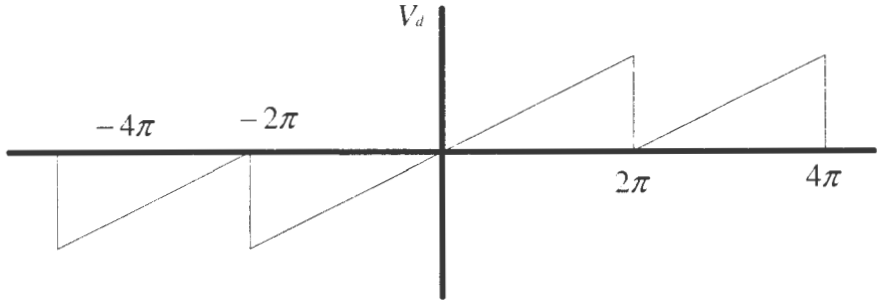

Figure 8 : PFD Response

Alexander (Bang-Bang) Phase Detector:

This phase detector is widely used in clock data recovery applications because it produces a sample of the data as well as phase error information. Additionally, it recognizes when there has been no transitions[1][2-4][13-14]. This is important so that no adjustments to the loop are made without transitions. Additionally, the outputs are synchronous digital signals and integrate well with all digital PLL's. The phase error information is not dependent on amplitude, duty cycle, or frequency of the phase detector outputs as they are with the other digital phase detectors discussed.

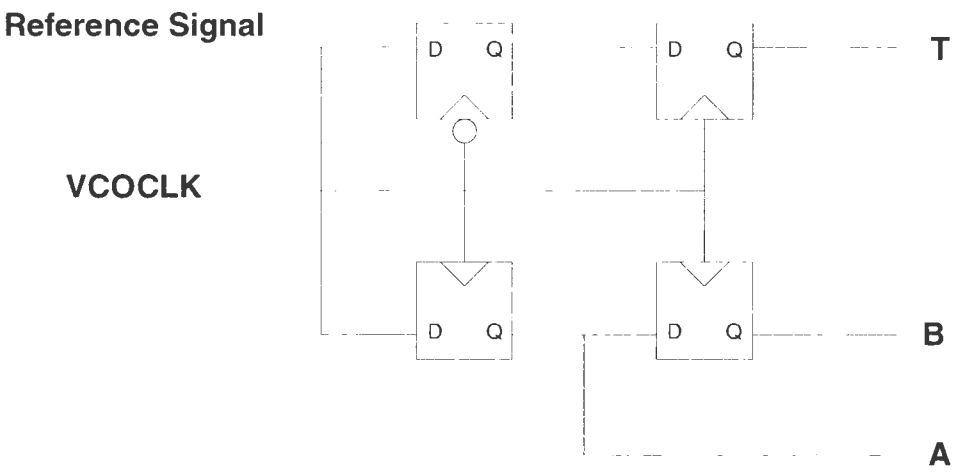

Figure 9 : Alexander (Bang-Bang) Phase Detector

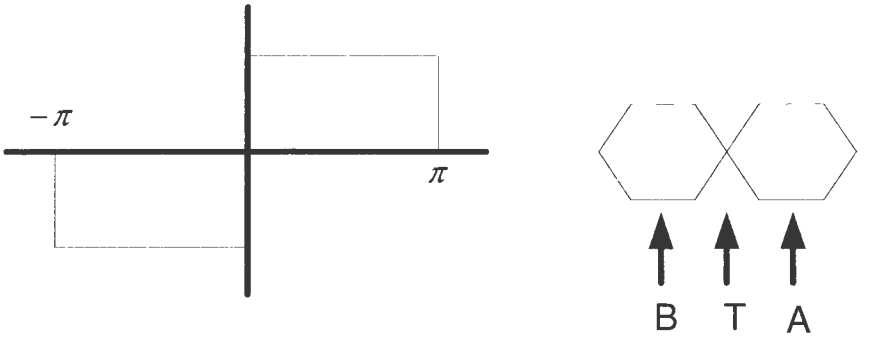

Figure 10 : Bang-Bang PD Response

1.3.2 Charge Pumps and Loop Filters

A Charge Pump is used in many classic digital PLL's, especially the fully integrated. Its purpose is to convert the digital signals to analog current or voltage level suitable for a current or voltage controlled oscillator. Gardner[16] was one of the first to produce a paper describing charge pump operation. The charge pump operates by turning on or off one or more current sources. Figure 11 illustrates a common integrated charge pump architecture. Analog loop filters can be grouped as either active or passive. A passive loop filter would consist of a single pole low pass filter constructed from a resistor and a capacitor similar to Figure 11; while an active loop filter would contain an amplifiers and gain control. Loops

greater than 2nd order are rarely used because of stability issues[1]. A Digital loop filters would consist of accumulators, integrators and gain control[1][7][14].

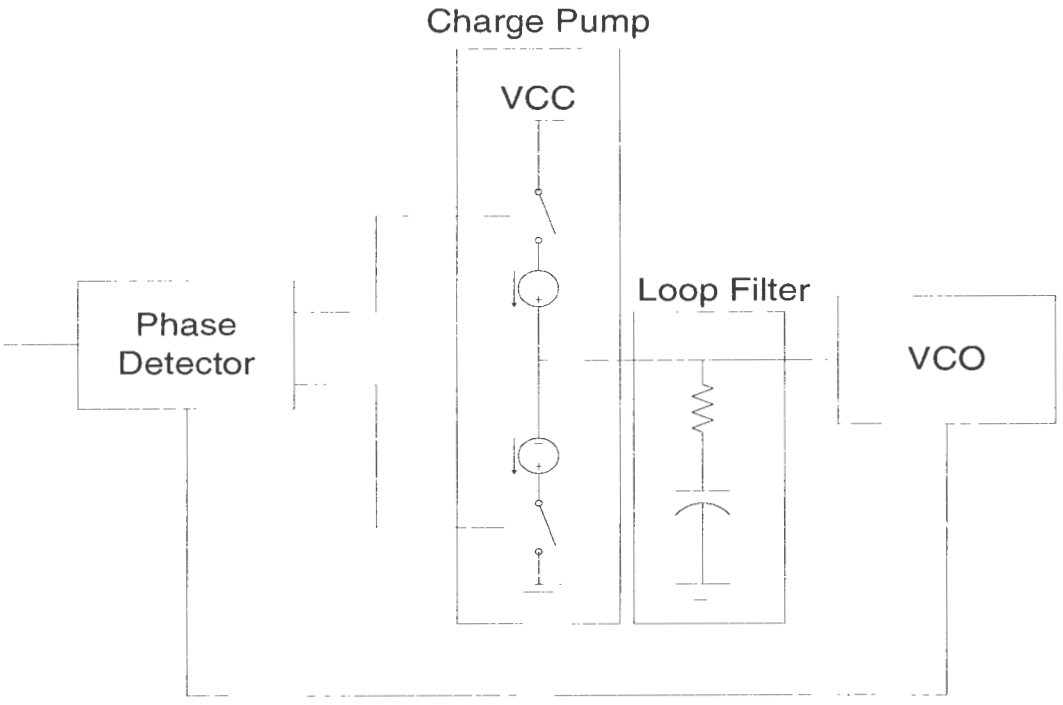

Figure 11 : Example Charge Pump and Loop Filter

1.3.3 Controlled Oscillators

There are many types of controlled oscillators depending on whether they are integrated inside of an IC or external. The basic integrated versions are the ring oscillator and the LC oscillator[8][9]. A ring oscillator consists of a series of buffers or inverters forming a feedback loop. The delay of each buffer is controlled via the control voltage or current. In most cases the delay is adjusted using a voltage control capacitor or varactor. An initial signal is injected into the ring and as long as the total gain of the ring is greater than or equal to 1 it will continue to oscillate. If the individual stages or buffers of the ring oscillator are well matched, it is easy to generate multiple phases of the output clock which is an advantage in many architectures. The LC or tank oscillator consists of an inductor and capacitor connected in parallel. The tuning range of LC oscillators is usually much narrower than ring

oscillators due to the difficulty in changing the inductance value. However, LC oscillators have much lower phase noise than ring oscillators[8]. External controlled oscillators are usually based on quartz crystal technologies using either Bulk Acoustic Wave(BAW) or Surface Acoustic Wave(SAW) technologies[11][12]. There is also now published work demonstrating the use of crystal based technology in on chip or integrated applications[10]. Crystal based oscillators are among the lowest noise solutions available.

1.4 Thesis contribution

In this final section of the introduction the architecture and design which was developed for this thesis will be introduced. The design utilizes a FPGA, DAC and VCXO. The major contributions of this thesis are:

1. An all digital PLL were designed, implemented, and characterized.
2. The design is flexible and can be adapted to many applications.
3. The design is scalable to many process technologies.
4. The design can be adapted to lower or higher frequencies.
5. The design produces a very low noise clock.
6. The design implemented is relatively low cost using inexpensive components.
7. The design can be used to filter noise in other clock or data signals.
8. The design allows the bandwidth of the loop to be adjusted dynamically allowing for faster acquisition time and increased filtering capability.
9. The design can be used in clock and data recovery applications.
10. The design introduces a modified phase detector and loop filter architecture which produces a more linear response than traditional bang-bang phase detectors.
11. An automated jitter transfer test was developed to analyze the loop performance.

Figure 12 on the following page illustrates the basic architecture used.

1.4.1 Architecture and Design

An Alexander or Bang-Bang phase detector was selected for implementation for primarily two important reasons. This phase detector works very well as a clock and data recovery phase detector since it is able to ignore periods where there are no transitions[1]. Secondly, it is a synchronous digital output which can be integrated easily with digital loop filter circuits.

This allows greater flexibility in applications and reduces the need for specific divider ratios on the reference clock. Additionally, if the reference signal is intermittent, the loop will maintain its present frequency which is a benefit in many applications. One unique feature of the implemented phase detector is that it operates at a much higher sample rate than the loop filter and DAC. Figure 13 shows a Sample Clock going to the Loop Filter. This clock is an integer divide from the VCO Clock which enters the FPGA. The result is that the phase detector output becomes a 9 bit signed integer. This makes the phase detector operate more linear because the output of the phase detector is not a +1 or -1 as is the common implementation of bang-bang phase detectors. The output gain of any bang-bang phase detector will be dependent on the jitter or phase noise present on the reference signal and the clock inputs[2][3][14].

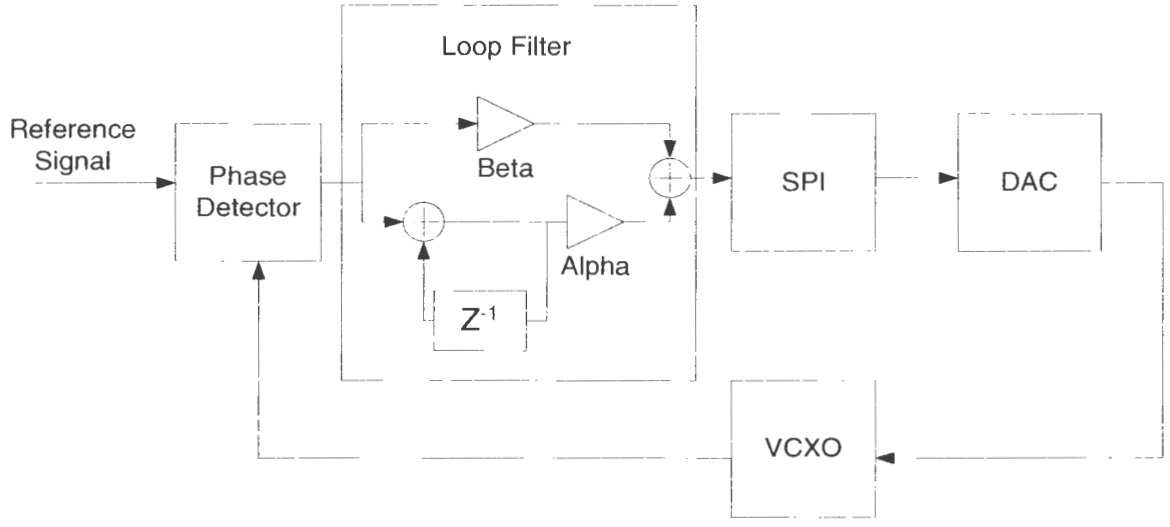

Figure 12 : Simplified Design Architecture

The design implements a digital loop filter which has both first order and second order paths. The first order path contains the gain control Beta and controls the response to small changes in phase, while the second order path contains the gain control Alpha and helps to maintain stability and responds to larger frequency and phase steps and maintains the DC offset required by the VCXO. The SPI interface is a serial to parallel circuit which receives the 16bit data from the loop filter and creates a serial stream of bits for the DAC. A survey

of available DAC devices was performed. There are two basic classes of DAC's available off the self. The first is the inexpensive, low frequency type which operate at update rates less than 1Msample/second. These DAC's are available with resolutions between 10 to 16 bits. The cost of these DAC's is between $1 and $8. The second class is the more expensive high frequency type which operate at rates greater than 1Msample/second. These DAC's have parallel interfaces and use large packages. The resolutions available is between 12 and 24 bits and are tailored for wireless applications. The cost of these DAC's is usually greater than $20. The design implemented uses the inexpensive, lower frequency DAC which are available from a variety of vendors. It will be shown that there is only a very small penalty in added quantization noise resulting from the use of the slower more inexpensive DAC.

1.5 Analysis of implemented loop parameters

1.5.1 DAC Resolution and Sample Rate

To determine the effect of DAC resolution and update rate on the overall jitter of the system Equation (2) should be considered.

$$J_Q(s) = \left(\frac{1}{1 - \frac{R * K_{dac} * K_v}{10^6}} \right) - \left(\frac{1}{1 + \frac{R * K_{dac} * K_v}{10^6}} \right) * \frac{1}{F_s} \qquad (2)$$

Where R is the assumed orbit radius[3], K_v is the VCXO gain in parts per million (ppm), and F_s is the update rate of the loop filter and DAC. The DAC gain is defined by Equation (3).

$$K_{dac} = \frac{V\max - V\min}{2^N} \qquad (3)$$

For this implementation we are using a 12bit DAC resulting in K_{dac} equal to 0.8mV/bit. Fs is equal to 1.15Mhz, K_v is 390ppm, and an estimated R = 3. Therefore we could expect J_Q approximately equal to 1.6ps p-p.

1.5.2 Loop Filter Gain

Due to the non-linear characteristics of this type of ADPLL, the analysis has not been widely published. Nicola Da Dalt published a time domain analysis[3] of a very similar PLL. His paper determined that there existed a minimum value of Beta for a given D and Alpha. D is the delay of the system.

$$\beta_{\min} = \frac{\alpha * (1 + 2 * D)}{2} \qquad (4)$$

Additionally Da Dalt determined that there existed an optimal value of Beta such that minimum jitter would be possible due to minimization of the orbits[3].

$$\beta_{opt} = (1.3846 + 1.8846 * D) * \alpha \qquad (5)$$

It must be noted that the references to Beta in the design and measurement sections is somewhat different than Da Dalt's and the introduction chapter of the thesis. Da Dalt refers to Beta and Alpha as pure gain. In the design of this ADPLL Alpha and Beta are related to gain via Equations (6).

$$\beta_{gain} = 2^{-Beta} : \alpha_{gain} = 2^{-Alpha} \qquad (6)$$

1.6 Conclusion

In this chapter the basic set of PLL architectures where discussed. The components of these PLL's where also described and compared. Finally, the specific design implemented in this thesis was also introduced as well as supporting analysis of jitter performance and stability.

Chapter 2 Implementation

2.1 Introduction

This chapter describes the detailed implementation of the all digital PLL. Each module or block of the design will be described including input and output signal names and functionality. Additionally the supporting circuits used for control and status monitoring and user interface which was used during characterization will also be described.

2.2 Design

The design was implemented with digital circuits designed using Verilog HDL. The design was compiled, synthesized, and fitted into a Virtex-II ProX FPGA. The FPGA was connected to external components as shown in Figure 13. The external components consist of a low cost 12 bit digital to analog converter (DAC) Analog Devices AD5320 and a voltage controlled crystal oscillator (VCXO) Crystek CVPD-034 operating with center frequency of 156.25Mhz. The ADPLL design consumes less than 178 slices, which is 1% of the V2PX20 FPGA used and ~13% of the smallest Virtex-II Pro device that Xilinx makes, the 2VP2. A slice is a standard logic block which all Xilinx FPGA's are built. This design can easily fit into a device that is also supporting other functions without requiring extra expense or larger device. This design could also be implemented in low cost CPLD technology such as the Xilinx CoolRunner II product family if an FPGA was not already present on the circuit card. Figure 23 provides a detailed diagram of the circuits inside the FPGA including supporting circuits to allow testing. Appendix A contains the Verilog HDL source code for the FPGA circuits. The top level verilog HDL source file is available in Appendix A and is named "fpga_top.v".

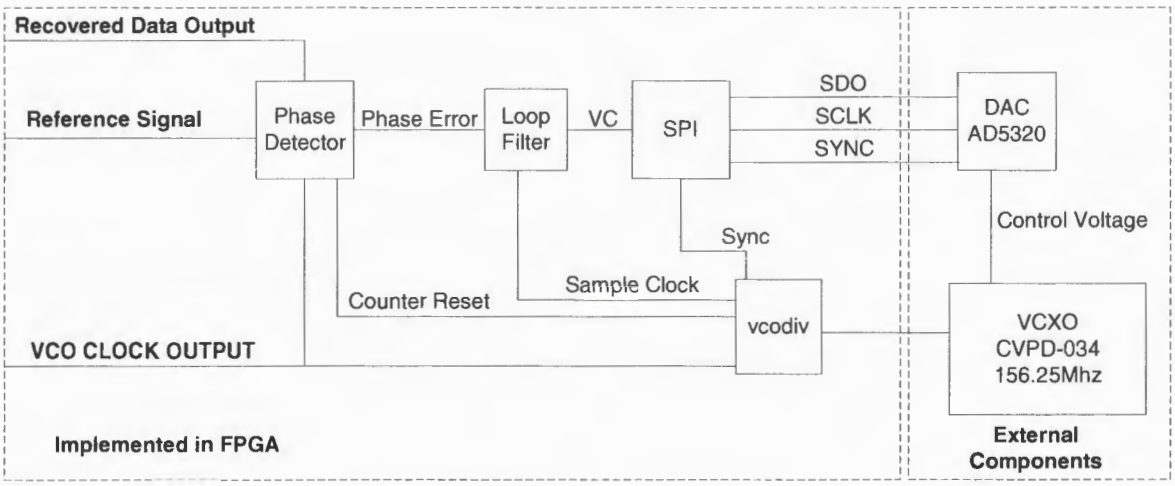

Figure 13 : Simplified System Level Diagram

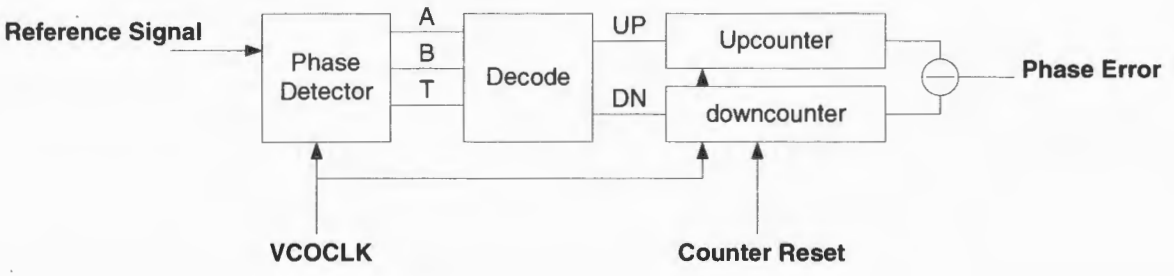

Figure 14 : Accumulating Bang-Bang Phase Detector

2.2.1 Phase Detector

An Alexander phase detector as depicted in Figure 9 was augmented with a decoder and up and down counters. This was done to create the accumulating bang-bang phase detector depicted in Figure 14. The accumulating bang-bang phase detector allowed the phase detector to operate at a higher rate than the loop-filter and DAC. This is important when a low speed DAC converter is used. The Phase Error is represented by a 9 bit signed integer instead of the typical single signed bit of an Alexander phase detector. The decode operation for the phase detector is illustrated in Table 1. Both Invalid and No transition cases do not cause the up and down counters to increment. In a low transition case this allows the VCXO to maintain a constant frequency. This approach does not have the same limitations as an

analog or classic digital PLL. A PLL with an analog loop filter and a charge pump would tend to drift downward in frequency in the absence of transitions due to leakage currents and loss. The up and down counters are reset whenever the phase error signal is sampled by the loop filter. The Verilog HDL source file is in Appendix A and is named "pd.v".

Table 1 : BBPD decode table

A	T	B	Description
0	0	0	No transitions
0	0	1	Down
0	1	0	Invalid
0	1	1	UP
1	0	0	UP
1	0	1	Invalid
1	1	0	Down
1	1	1	No transitions

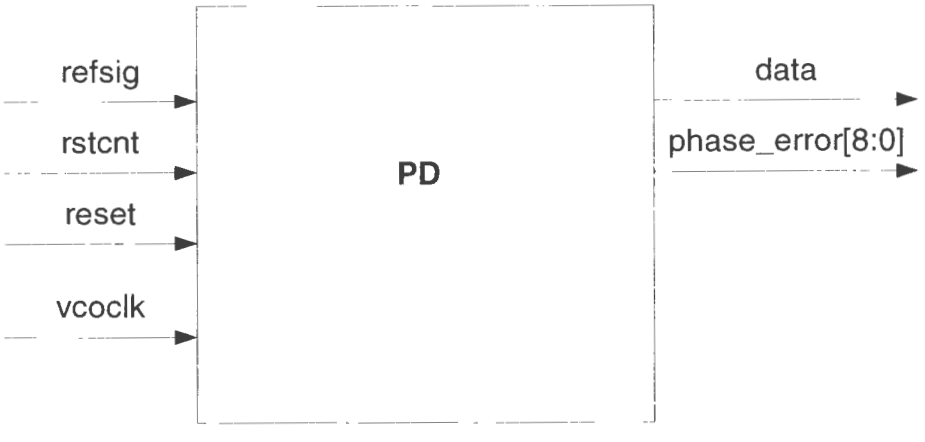

Figure 15 : Accumulating Phase Detector Input Output Diagram

Table 2 : Accumulating Phase Detector Input Output Description

Signal Name	Description
refsig	Reference signal may be a clock or data signal
rstcnt	Counter Reset signal
reset	Resets the phase detector flip-flops
vcoclk	clock signal from the VCO or VCXO
data	sampled data output synchronous to vcoclk
phase_error[8:0]	9 bit signed integer representing the phase error

2.2.2 Loop Filter

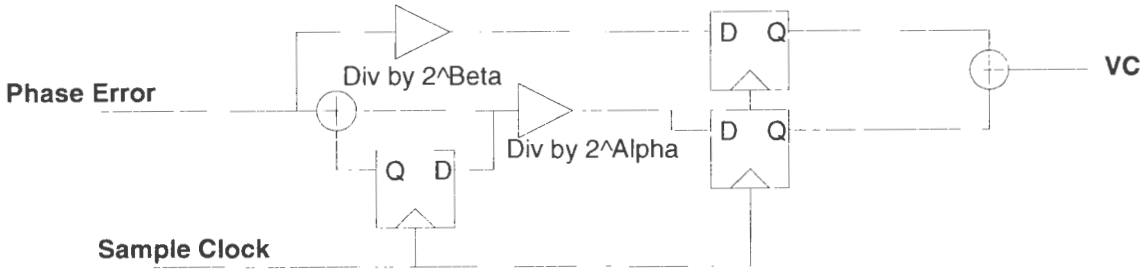

Figure 16 : Digital Loop Filter Implementation

The loop filter consists of a first order path which is just a gain stage and a second order path which is an integrator/accumulator followed by a gain stage. The first order gain is controlled via input beta and divides the input value by 2^{beta}. The second order gain is controlled by input alpha and divides the accumulated value by 2^{alpha}. Because it divides only by multiples of 2, the dividers can be simplified to binary right shifts. This is important for implementations using standard digital gates because it reduces the complexity and size of the dividers. However, in applications where multipliers are available such as the DSP blocks in the Xilinx Virtex 4 FPGA, it would be better to use true multiplies in order to give better control of loop parameters.

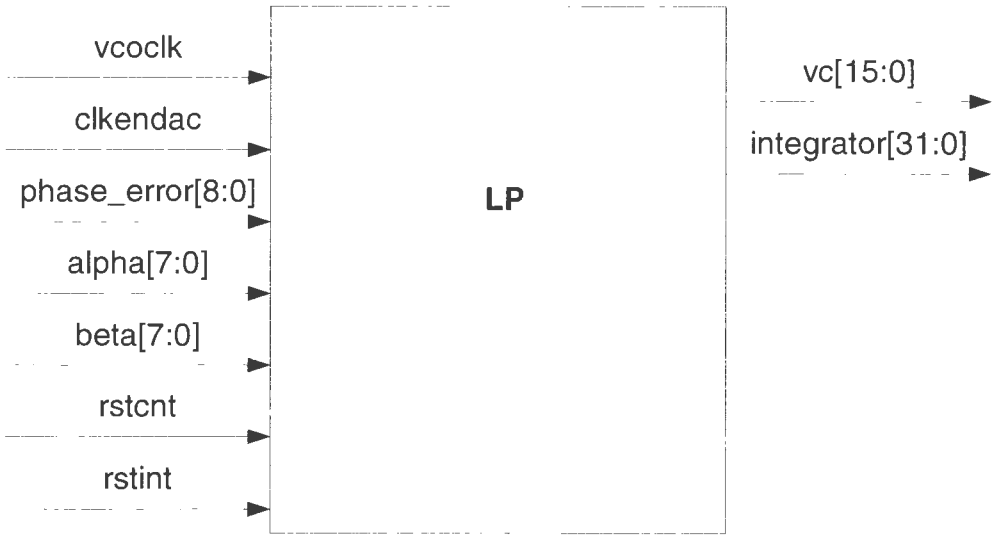

Figure 17 : Loop Filter Input Output Diagram

Table 3 : Loop filter Input Output Description

Signal Name	Description
vcoclk	clock signal from VCO or VCXO
clkendac	clock enable signal used for all registers
phase_error[8:0]	9 bit signed integer representing phase error
alpha[7:0]	controls 2nd order loop gain
beta[7:0]	controls 1st order loop gain
rstcnt	identifies when to sample the phase_error signal
rstint	Reset the integrator registers
vc[15:0]	16 bit unsigned integer output representing the Control Voltage
integrator[31:0]	32 bit unsigned output of the integrator value used for monitor and debug

In this implementation the Sample Clock is the VCO output clock, however it is controlled by the signal clkendac which acts as a clock enable. This results in an effective clock which is divided from the VCO clock. The rate of this Sample Clock is controlled by the vcodiv circuit and the input control signal divcnt. The signal divcnt should be set so that the period of the Sample Clock is greater than the minimum period allowed by the DAC; in this case 50ns. The Verilog HDL source file for the loop filter is in Appendix A and is named "lp.v".

2.2.3 SPI – Serial Peripheral Interface

The Serial Peripheral Interface is a standard interface which provides the communication of the 12 bit DAC value through a 3 wire serial interface. Most microcontrollers and processors support similar interfaces. Using this interface allows flexibility in the DAC design; the same pin out and package can be used for many different DAC models allowing pin for pin compatibility for 8bit to 16 bit DAC's. Additionally using a serial interface reduces the size of the external DAC package. This is important so that the design can maintain only a small amount of space on the printed circuit board. The Verilog HDL source file for the serial peripheral interface is in Appendix A and is named "spi.v".

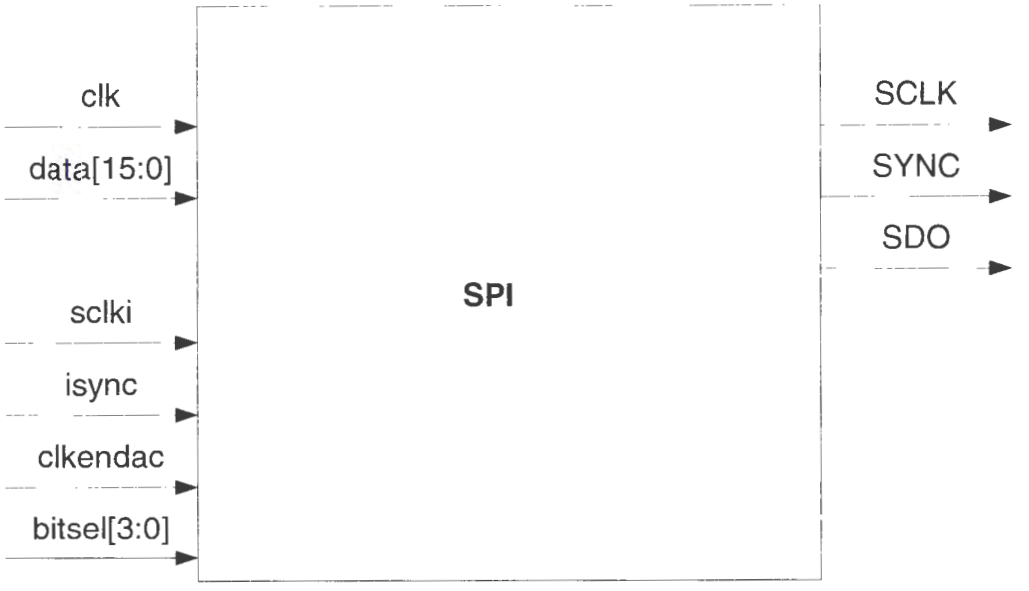

Figure 18 : SPI Input and Output Diagram

Table 4 : SPI Input and Output Description

Signal Name	Description
clk	clk input used for all registers
clkendac	clock enable signal used for all registers
data[15:0]	data to be sent to DAC
sclki	input clock for DAC interface
isync	input sync signal for DAC interface
bitsel[3:0]	identifies which bit of data goes to DAC
SCLK	Buffered sclki driven to DAC
SYNC	buffered isync driven to DAC
SDO	Buffered data[bitsel] driven to DAC

2.2.4 VCO Divider (vcodiv)

This block is responsible for creating the clocks, clock enables and sync pulses used by the other circuits in the design.

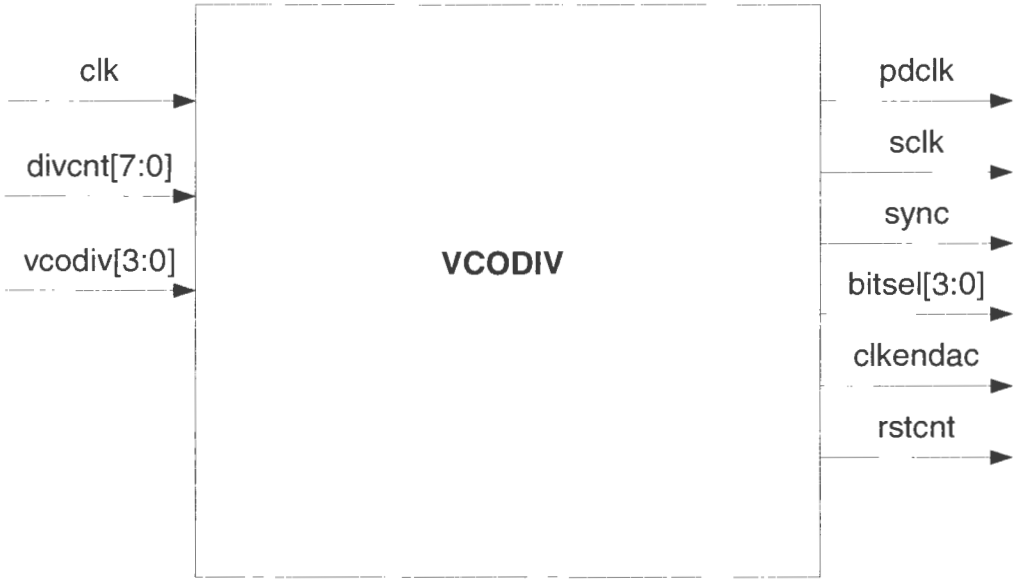

Figure 19 : VCODIV Input/Output Diagram

The outputs of this block synchronize the sampling of the accumulating phase detector, reset of the counters, calculations in the loop filter and the transmission of the DAC value thru the SPI to the DAC.

Table 5 : VCODIV Input and Output Description

Pin Name	Description
clk	This is the input clock from the VCXO
divcnt[7:0]	Divider value for the Sample Clock
vcodiv[3:0]	Divider value for the pdclk
pdclk	clk divided by vcodiv
sclk	clk divided by divcnt driven out of the FPGA to DAC
sync	pulse occurs 1 every 17 cycles is used by SPI block
bitsel[3:0]	identifies the bit which should be send out the SPI block
clkendac	clock enable signal for loop filter, spi blocks pulse identifies the rising edge of sclk
rstcnt	Counter Reset signal used to reset the up and down counters in the phase detector

The inputs divcnt and vcodiv control the rate of the output clock and clock enable signals, clkendac. The pdclk signal is the clk signal divided by vcodiv. This output clock can be selected to drive the phase detector VCOCLK signal if the VCXO is too fast for the FPGA fabric. This would be the case if the VCXO where operating at greater than 200Mhz. However this operation was not used or required during the characterization of this design. The clkendac signal is a pulse with duration equal to the period of the signal clk. It is used as a clock enable signal by the loop filter and SPI interface so that only a single global clock signal vcoclk is used by FPGA logic. The signal divcnt is used to determine the rate of the clock enable. The signal sclk is a 50% duty cycle version of this which is driven out of the FPGA to the DAC. The Verilog HDL source file is provided in Appendix A of this thesis and is named "vcodiv.v".

2.2.5 Digital to Analog Converter

The DAC used for this implementation is an Analog Devices AD5320 device[15]. This device has several important advantages. First, it is a low-cost DAC with a volume price less than $2. Second, it is very small package which saves space on the PCB board.

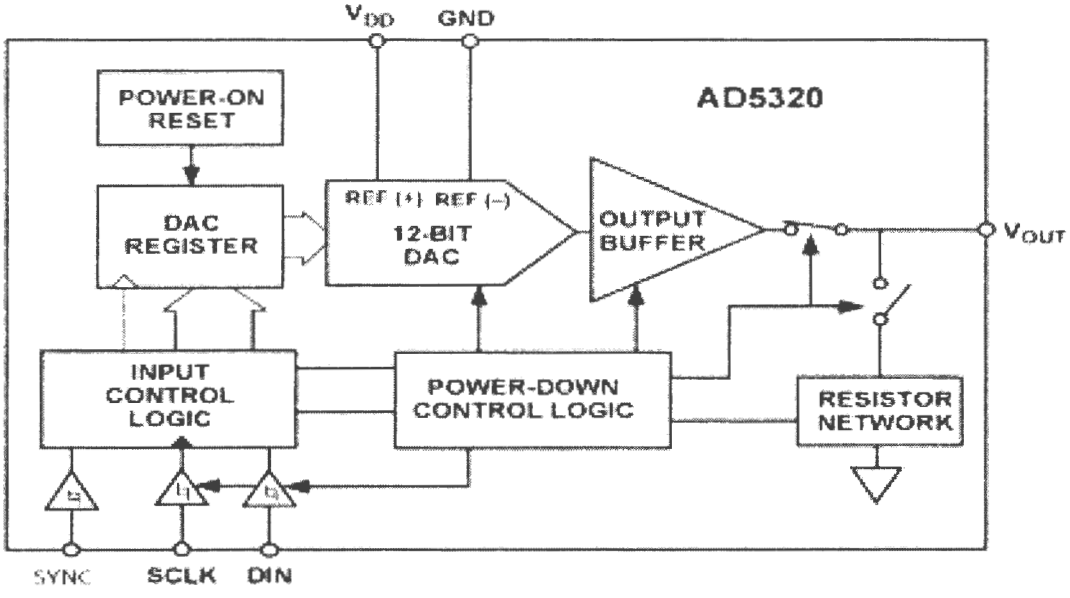

Figure 20 : AD5320 Diagram

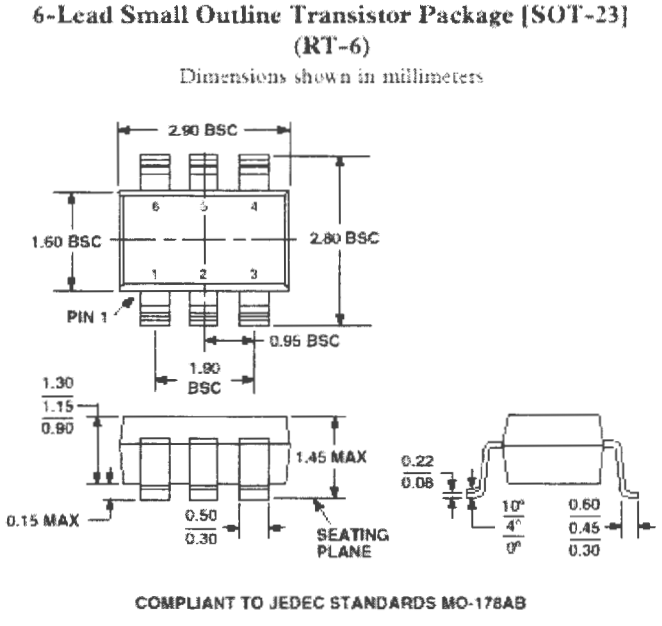

Figure 21 : AD5320 Package Diagram

2.2.6 Voltage Controlled Crystal Oscillator

The VCXO used for this implementation was produced by Crystek, Inc. The specifications as stated in the product data sheet are illustrated in Table 6[14]. The K_v or gain associated with the this VCXO was not provided in the datasheet. However, it was measured to be approximately 390ppm.

Table 6 : VCXO specifications

Frequency Range:		77.760MHz to 200MHz
Temperature Range:		0°C to 70°C
(Option M)		-20°C to 70°C
(Option X)		-40°C to 85°C
Storage:		-55°C to 120°C
Input Voltage:		3.3V ± 0.3V
Control Voltage:		1.65V ± 1.65V
Settability At Nominal:		1.65V ± 0.25V
Input Current:		50mA Typ, 88mA Max
Output:		Differential LVPECL
	Symmetry:	45/55% Max @ 50% Vdd
	Rise/Fall Time:	1ns Max @ 20% to 80% Vdd
	Pulling Range:	±50ppm APR Min. (std)
	Linearity:	± 10% Max
	Logic: Terminated to Vdd-2Vinto	50 ohms
	Temp. 0°C to 85°C	"0" = 1.490 Min, 1.680 Max
		"1" = 2.275 Min, 2.420 Max
	Temp. -40°C to 0°C	"0" = 1.470 Min, 1.745 Max
	Enable/Disable Time	"1" = 2.215 Min, 2.420 Max
		200ns Max
Jitter:	12KHz to 80MHz	0.5psec Typ., 1psec RMS Max
Phase Noise:	10Hz	-60dBc/Hz Typical
(Ref: 122.88MHz)	100Hz	-95dBc/Hz Typical
	1KHz	-120dBc/Hz Typical
	10KHz	-140dBc/Hz Typical
	100KHz	-145dBc/Hz Typical
Aging:		<5ppm 1st/yr, <2ppm every year thereafter

2.2.7 ChipScope Pro

ChipScope Pro is a tool offered by Xilinx as a way to control and monitor internal circuits inside the FPGA. Two ChipScope Pro blocks were added to the design in order to facilitate the control and monitoring of the parameters listed in Table 7. Figure 22 provides a screen capture of the ChipScope Pro control panel as it was used with this design.

Table 7 : Programmable settings and monitors

Signal Name	Description
divcnt[7:0]	Controls the divider which creates the Sample clock it should be set so that the period of the sample clock is greater than the minimum period of the DAC
vcodiv[3:0]	Controls the divider of the clock going to the phase detector. Not usually used unless VCO is too fast for the FPGA logic
BUFGMUXSEL	Selects the divided vco clock or the vco clock directly
alpha[7:0]	controls the 2^{nd} order loop gain
beta[7:0]	controls the 1^{st} order loop gain
rstint	resets the integrator registers
integrator[31:0]	monitors the integrator value
phase_error[8:0]	monitors the phase_error value

The Chipscope Pro blocks contribute significantly to the total area of the design used for characterization. However, in a real system design these blocks would not be necessary because either the design would be fixed to certain parameters or the would exist some other method to control loop parameters. The logic blocks which were added to the design are ICON and VIO blocks. The ICON block is the control portion of ChipScope and VIO is a synchronous input and output block. The control and status signals are connected to the VIO block to allow access via this graphical user interface. The ChipScope blocks do not have source files associated with them. They get merged during the FPGA compile process. Figure 23 shows how these blocks are connected to the rest of the ADPLL design.

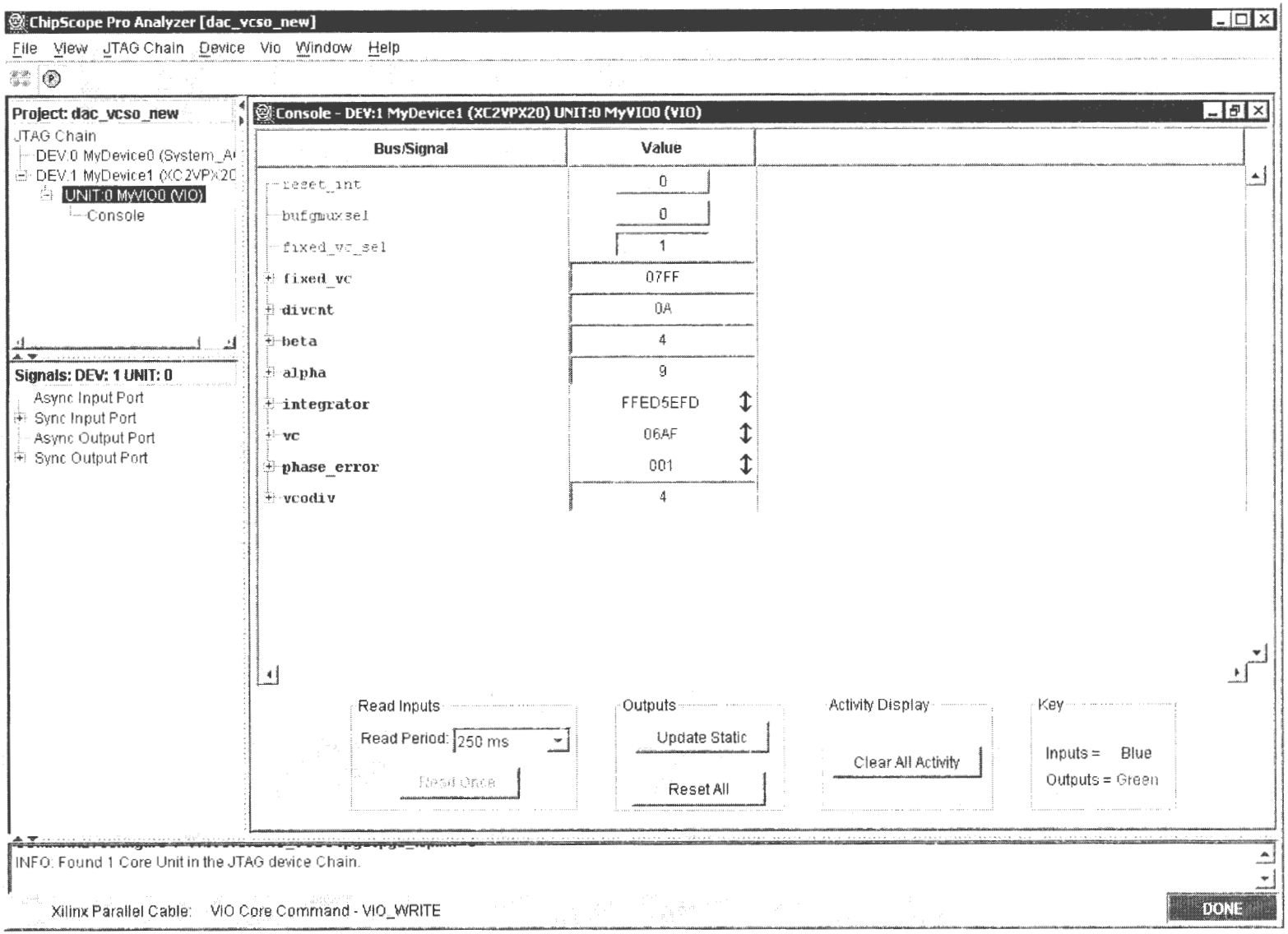

Figure 22 : ChipScope Pro Screen Capture

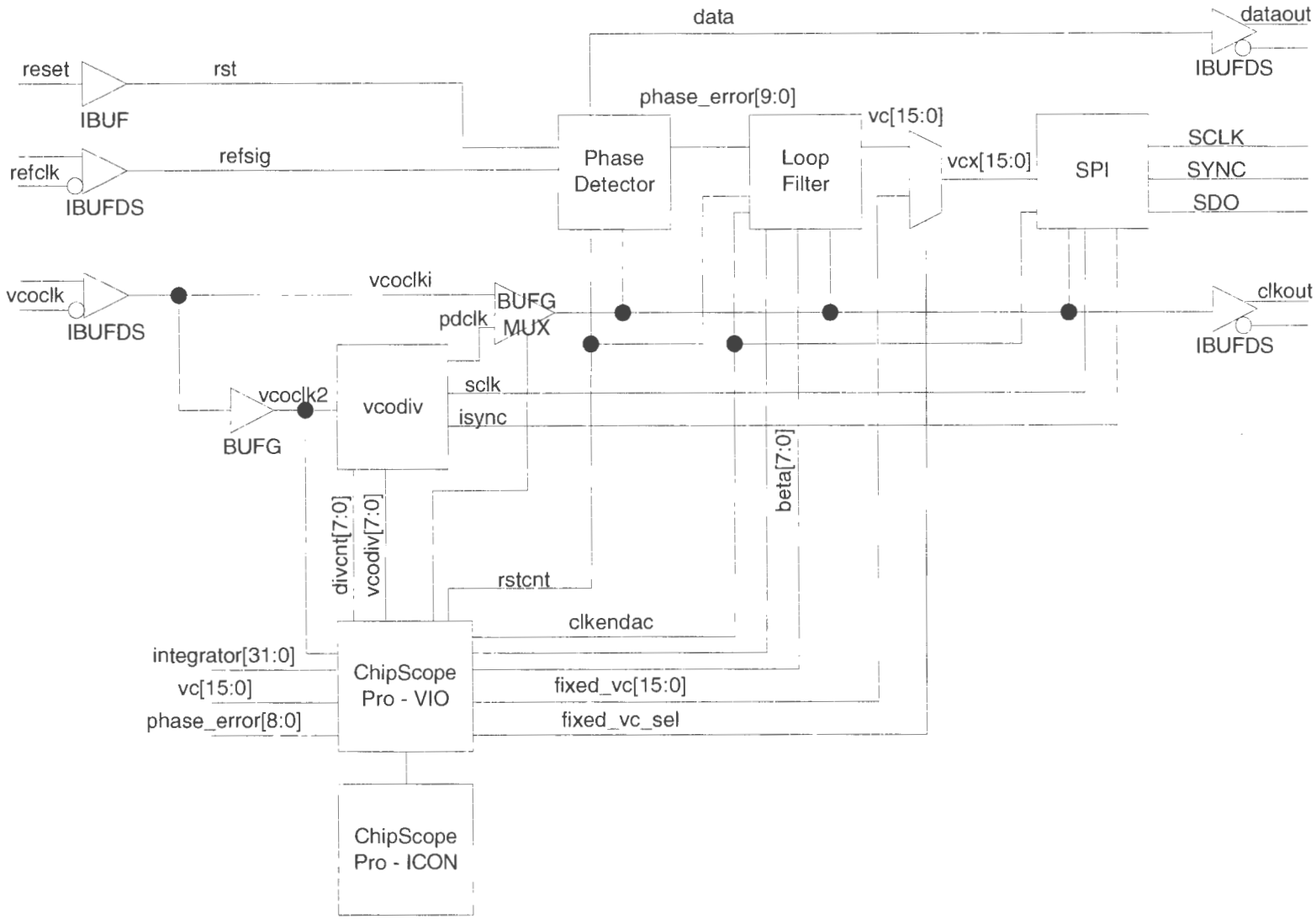

Figure 23 : Detailed Diagram of FPGA

2.3 FPGA compile and P&R

The entire design including Chipscope Pro cores was processed through the Xilinx ISE version 6.3.03i tools set. This included using the Xilinx Synthesis Tool (XST) to synthesize the verilog HDL files to EDIF format. Xilinx translate tool was used to combine the EDIF files from the synthesized design and the Chipscope Pro cores and the Coregen adder circuits into a complete design file. This design file is then processed by the Xilinx map tool which maps gates to FPGA slices. This design is then processed by the Xilinx par tool which generates a completely placed and routed design. The final step is to convert this design to a bit-stream file which can be programmed into the FPGA via the JTAG programming port. The design consumed 1226 Slices or 12% of the target device XC2VPX20. The majority of this was consumed by the Chipscope Pro cores which contain a lot of internal memory cells. When the DPLL is compiled without the ChipScope Pro blocks and all of the control signals such as alpha[7:0], beta[7:0], divcnt[7:0], etc are brought to input pins the ADPLL alone consumes only 178 Slices or 1% of the 2VPX20 FPGA. The design will operate at up to 172Mhz according to the static timing analysis performed by the Xilinx tools. All data was taken using a 156.25Mhz VCXO and 78.125Mhz Reference Signal. The printed circuit board used to test the design is a MK20XFP board which was co-developed by Xilinx, Inc and M6 Research, Inc. contains a picture of the board used.

2.4 Conclusion

In this chapter of the thesis the ADPLL design was described in detail. Each block associated with the design was describe and the input and output signals identified. Additionally, the ChipScope Pro interface was shown and the control and status signals where described. Finally, all verilog HDL source files for each module is provided in Appendix A of this thesis.

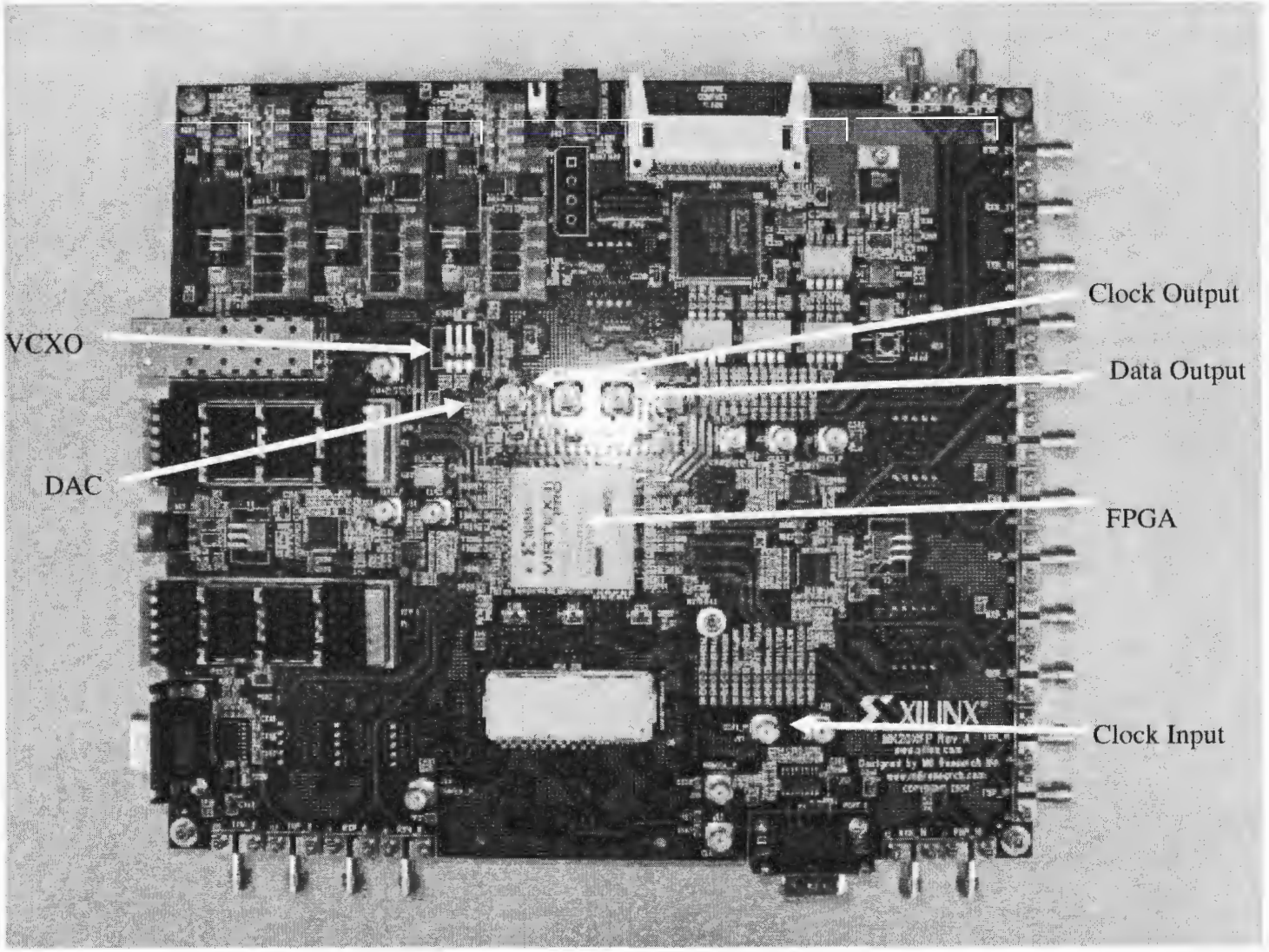

Figure 24 : Picture of MK20XFP Board

Chapter 3 Measured Results

3.1 Introduction

This chapter of the thesis is intended to present all data taken on the implementation as it was tested on the MK20XFP board as shown in Figure 24 utilizing the Xilinx Virtex-II ProX FPGA, Analog Devices AD5320 DAC, and Crystek 156.25Mhz VCXO. The individual parameters of the design were varied in order to analyze their impact on performance, bandwidth, and stability. Additionally, the test setups and automation software which was developed in order to acquire the data will also be described.

3.2 Measurement Setup

There were two basic test setups used to analyze the performance of the implemented ADPLL circuit. The first was a reference source and a spectrum analyzer which was used for jitter transfer, jitter generation, and frequency step response. The second was a Bit Error Rate Tester (BERT) using reference clock sources which was used only for jitter tolerance testing. This setup is a 12Gbps BERT test system developed by Hewlett-Packard before the company split off into Agilent. It has the ability to operate from 10Mhz to 12Gbps. Test Setup #1 was used for the majority of the testing including Jitter Transfer, Jitter Generation and Frequency Step Response. Test Setup #2 was used only for the Jitter Tolerance Test.

Table 8 : Equipment List for Test Setup #1

Manufacturer	Model Number	Description
Agilent	33250	Function / Arbitrary Waveform Generator, 80 MHz
Agilent	E4407B	E4407B ESA-E Series Spectrum Analyzer 100 Hz to 26.5 GHz
Agilent	infinuim	4Gs/s, 2.5Ghz Digital Storage Oscilloscope

Table 9 : Equipment List for Test Setup #2

Manufacturer	Model Number	Description
Agilent	83752A	Signal generator
HP	70004A	Display
HP	70843B	Error Performance Analyzer
HP	3325B	Signal generator

3.3 Labview Automation

In order to facilitate repeatable and numerous measurements, National Instruments Labview software was used to develop an automated Jitter Transfer Measurement using a test setup #1 as described in Table 8. Using the Labview software two virtual instruments (VI's) were developed. The first, Jitter_Transfer3.vi, configures the reference source to generate a reference clock at defined frequency with a defined amount of modulation. Then using the spectrum analyzer it measures the modulation amplitude present on the signal under test. This is repeated for a number of different Modulation frequencies. Labview VI's are constructed graphically first by constructing a Front Panel which consist of controls and indicators. The Front Panel represents the Graphical User Interface of the instrument. Figure 25 shows the Front Panel for the Jitter_Transfer3.vi Block. The operation of the program behind the Front Panel is represented by the Block Diagram. The Block Diagram is formed by connecting different controls and indicators from the Front Panel represented by icons to functions and sub-vi's using wires. Figure 26 shows the Block Diagram for the Jitter_Transfer3.vi. As can be seen from the Front Panel this VI receives a list of modulation frequencies, modulation frequency, center frequency, averages, and reference divide ratio. It outputs 3 vectors which are also plotted on the graph on the front panel. The vectors are modulation gain, input modulation amplitude which is calculated using Equation (7), and measured output modulation amplitude. As can be seen there is very good correlation between the two graphs. This data is recreated in Figure 30.

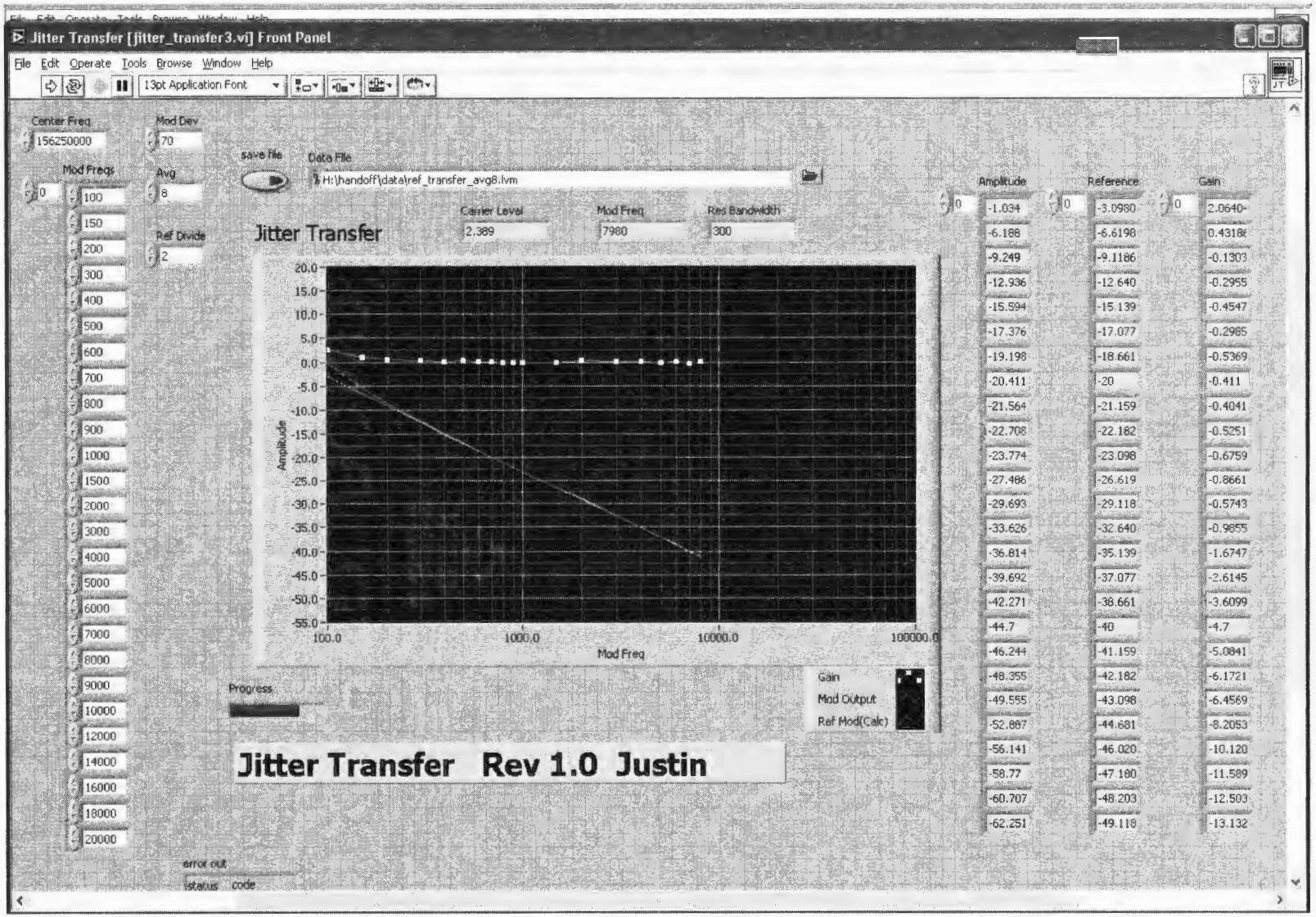

Figure 25 : Jitter Transfer3.vi Front Panel

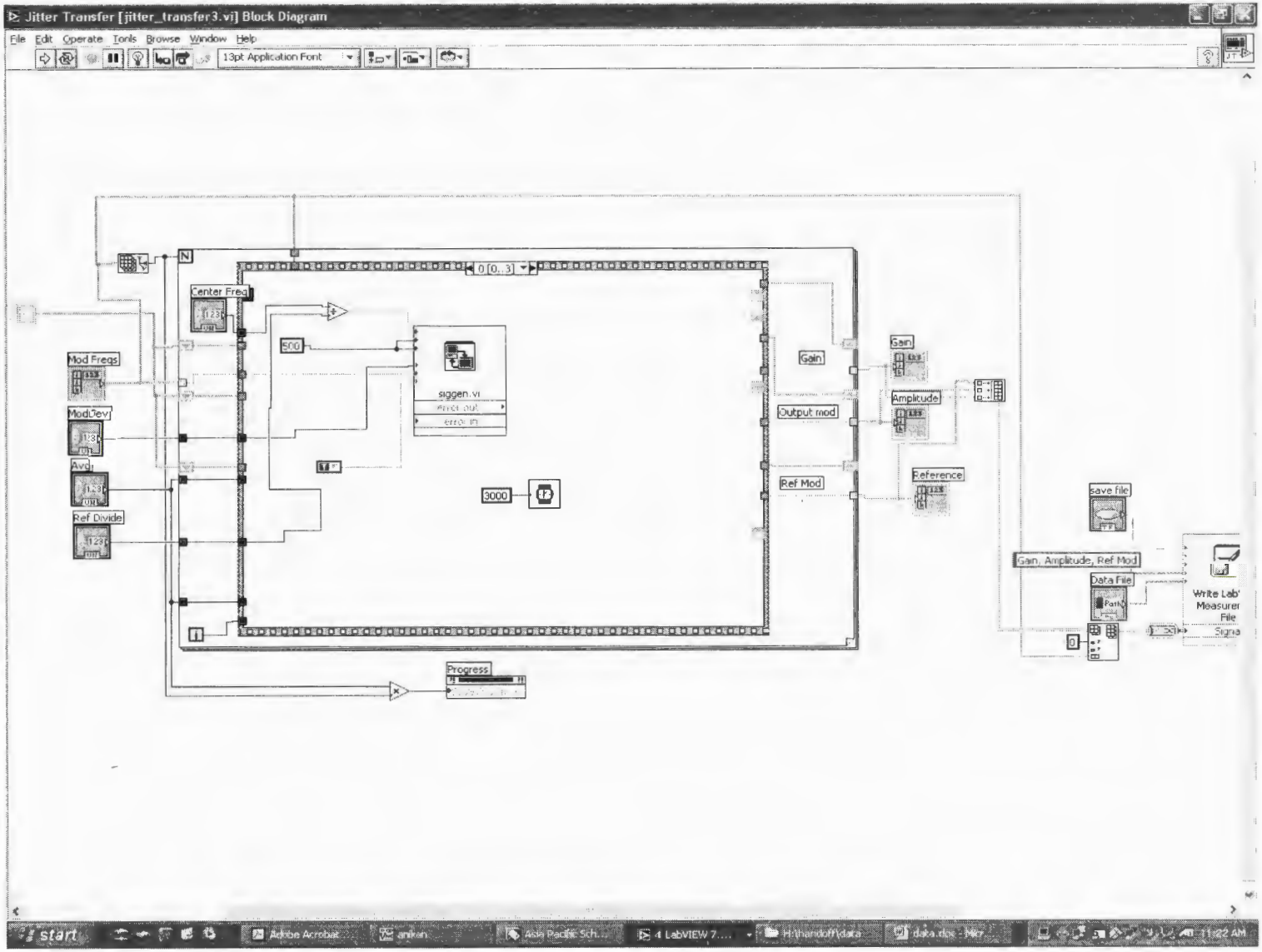

Figure 26 : Jitter Transfer3.vi Block diagram

The second VI, Multiple_jitter_transfer3.vi, calls the first VI, Jitter_Transfer3.vi, multiple times pausing between each call to allow for changing loop parameters via ChipScope Pro control panel. During each iteration of Jitter_Transfer3.vi, the data is graphed on separate amplitude and gain graphs for immediate feedback on progress and performance. Figure 27 shows the Front Panel of Multiple_jitter_transfer3.vi which receives the same basic inputs as Jitter_Transfer3.vi and passes them directly to Jitter_Transfer3.vi. The output is a series of graphs. The top graph shows data collected from each iteration through Jitter_Transfer3.vi and the comparison of the effects of the different parameter changes. This data is also saved to a data file which was later imported into a spreadsheet which recreated the graphs for inclusion in this thesis. The top graph shows the jitter transfer or modulation gain, the second graph shows just the modulation amplitude that was measured. Figure 28 shows the Block Diagram for the Multiple_jitter_transfer3.vi. Central in the VI diagram is the icon for the Jitter_Transfer3.vi sub-vi. It is shown along with the function block which pops-up a dialog box to request the loop parameter change. The Labview software uses the General Purpose Instrument Bus (GPIB) to communicate with both pieces of equipment.

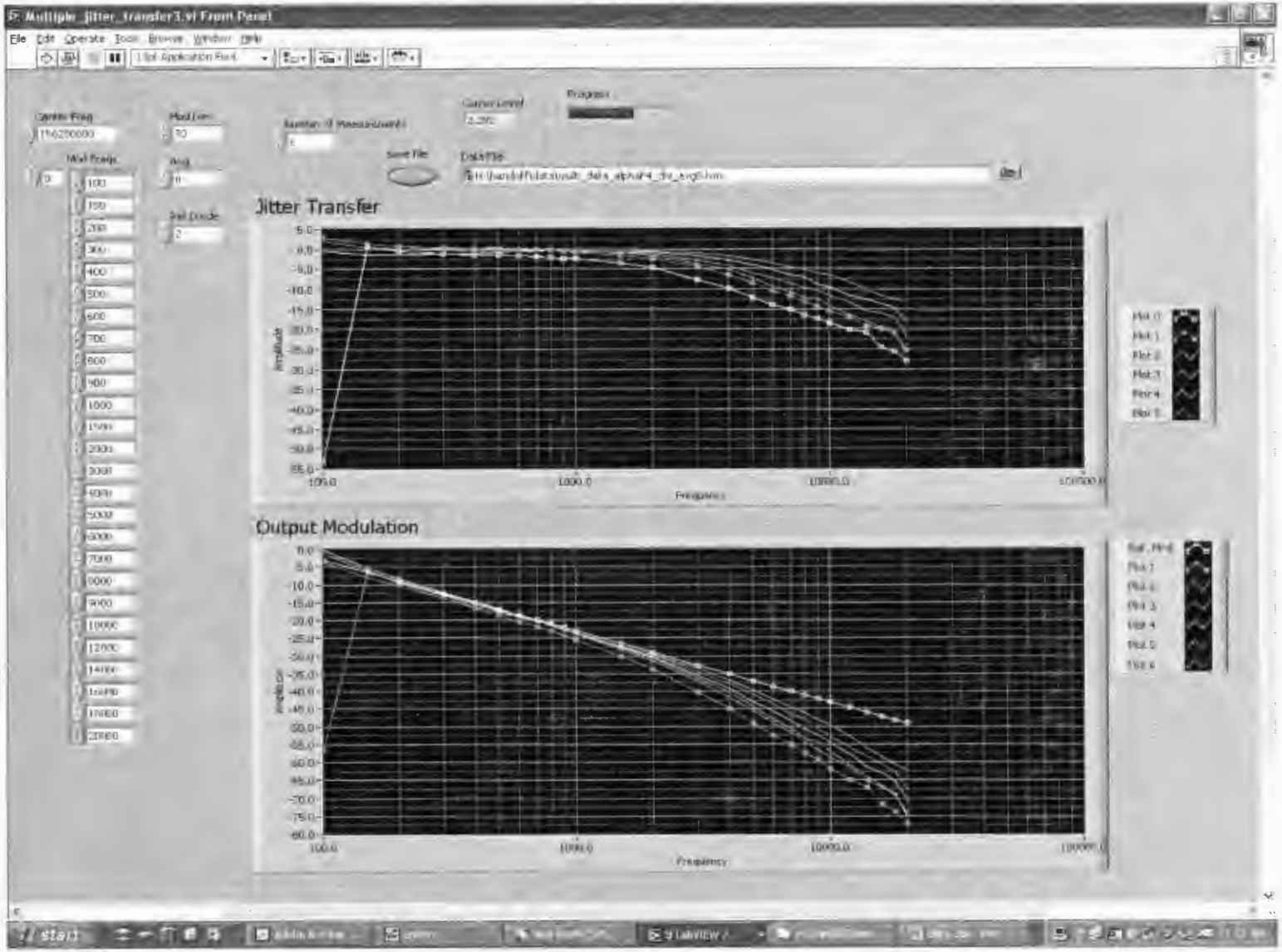

Figure 27 : Multiple_jitter_transfer3.vi Front Panel

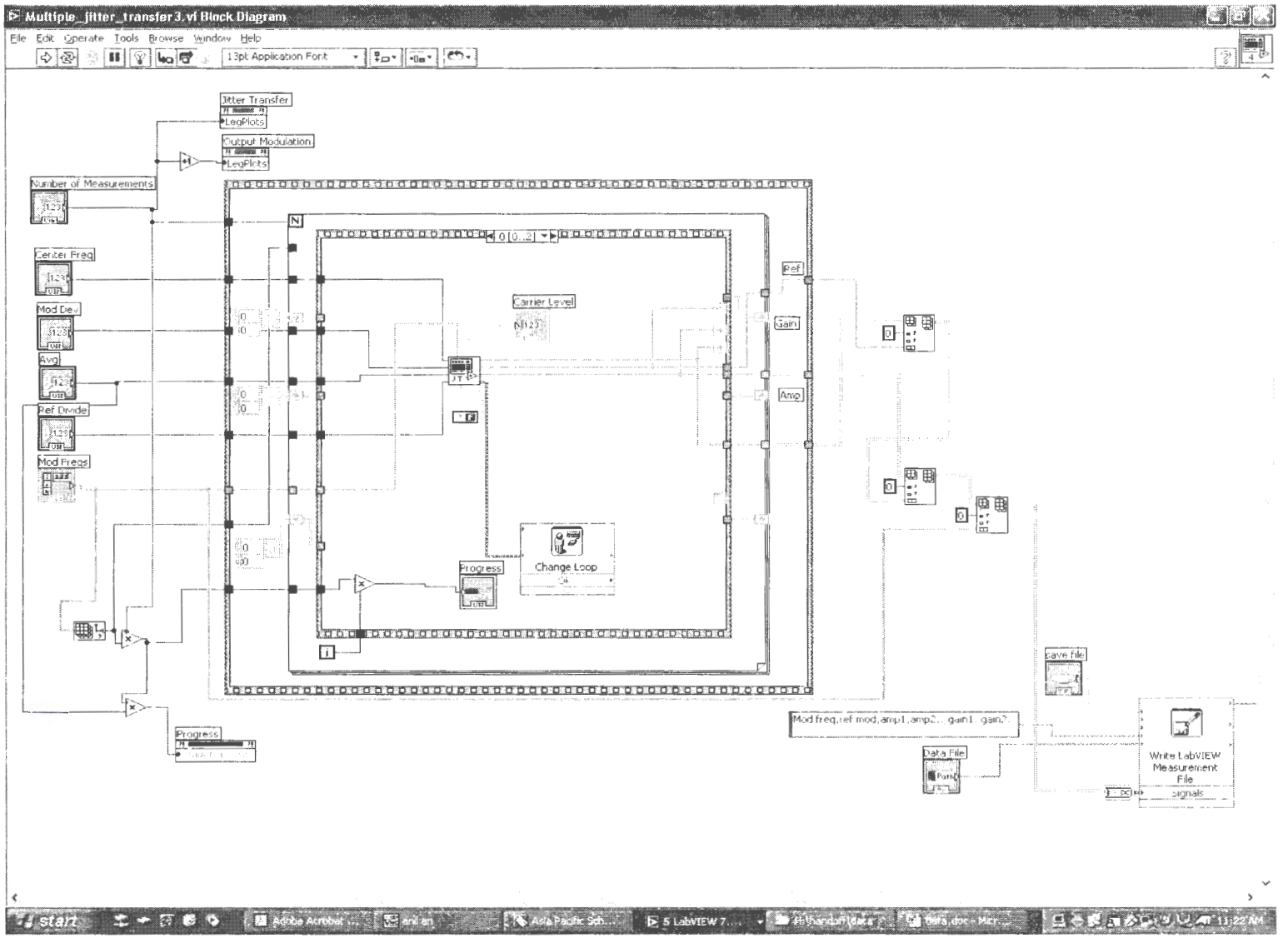

Figure 28 : Multiple_jitter_transfer3.vi Block Diagram

3.4 Jitter Transfer

Using the Labview VI's some calibration measurements were taken in order to get an idea of the amount of noise present in the VCXO and the FPGA output drivers. A baseline noise measurement was taken with the DAC set to a fixed value of 0x6F0. This is shown in Figure 29. This graph establishes the noise floor of the DAC and FPGA at about -60dBc to -70 dBc.

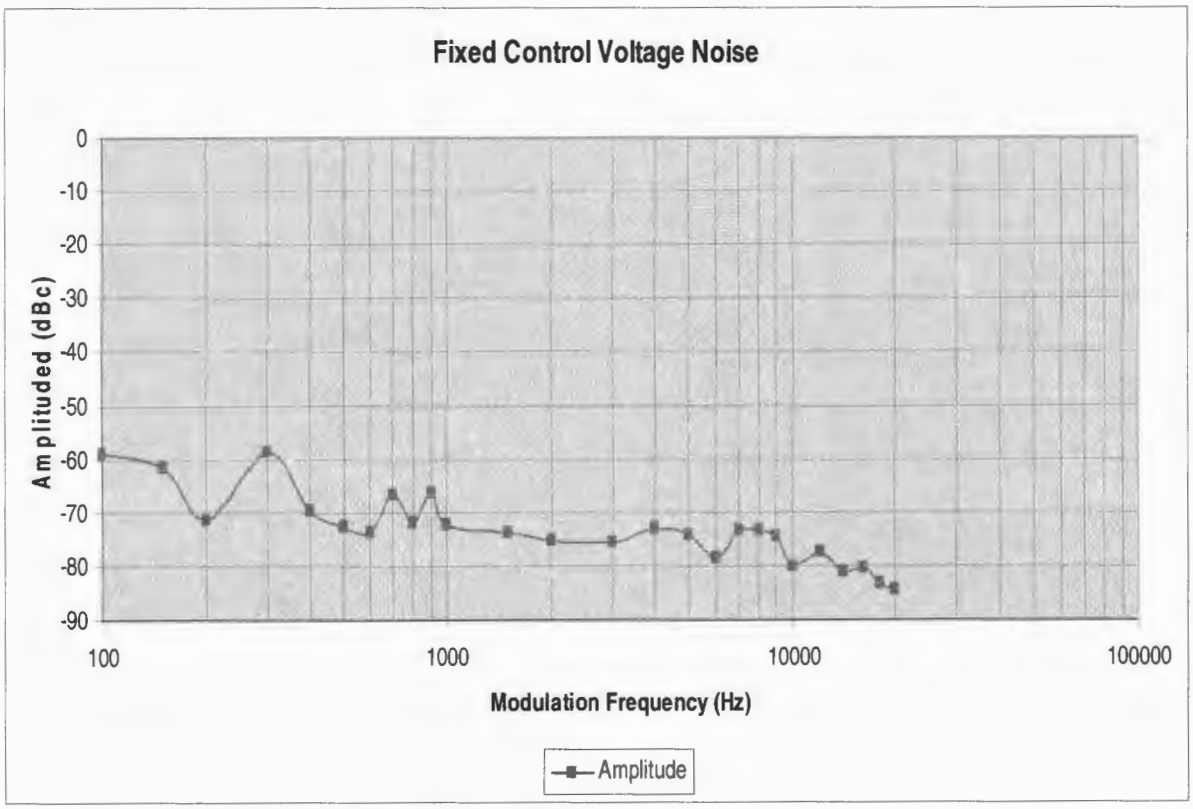

Figure 29 : Fixed Control Voltage Noise Plot

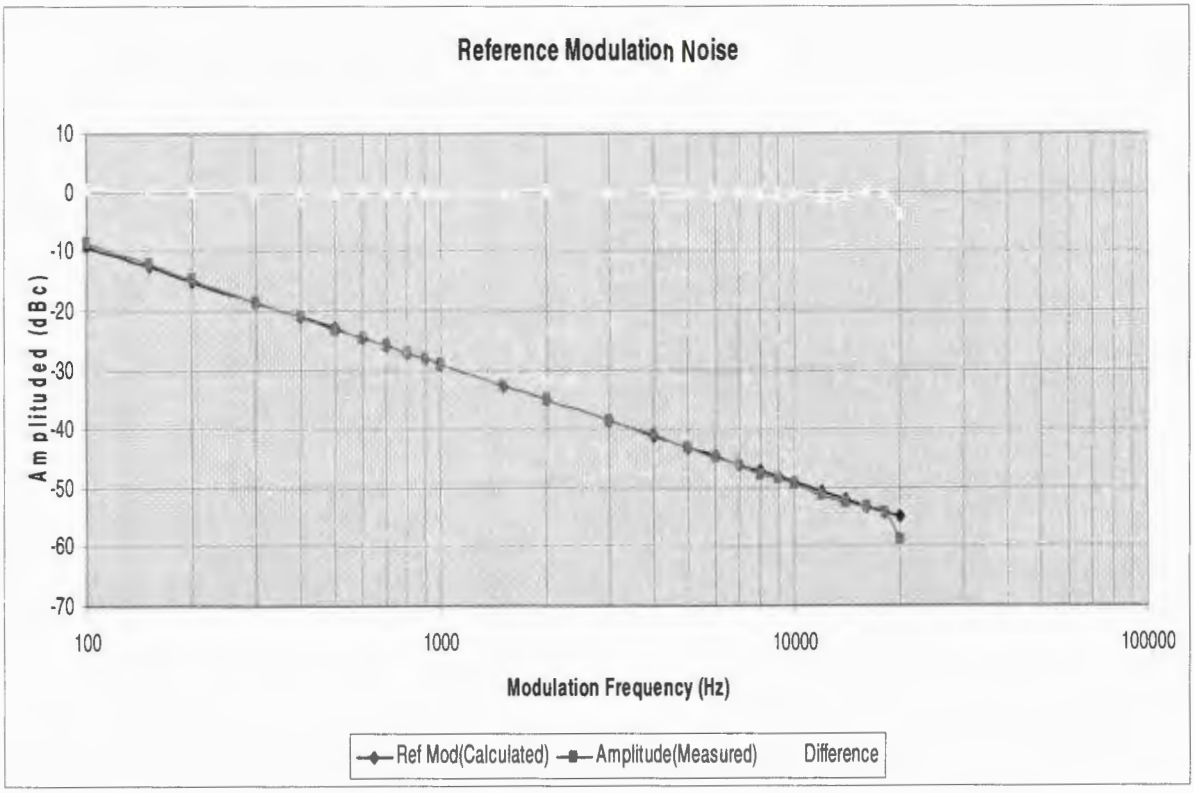

Figure 30 : Reference Modulation Plot

Next a measurement was taken to correlate the calculated value of Reference modulation amplitude relative to the actual measured value using the equipment. Amplitude of the reference modulation, A(dBc) is calculated by the following equation.

$$A(dBc) = 20 * \log\left(\frac{Fdev * Nref}{F\,\mathrm{mod} * 2}\right) \qquad (7)$$

Where Fdev is the frequency of the deviation, Fmod is the modulation frequency and Nref is the reference divide ratio. The reference divide ratio is usually 2, because the VCXO frequency is twice the reference frequency. Notice the two values correlate very well with only minor differences due to noise.

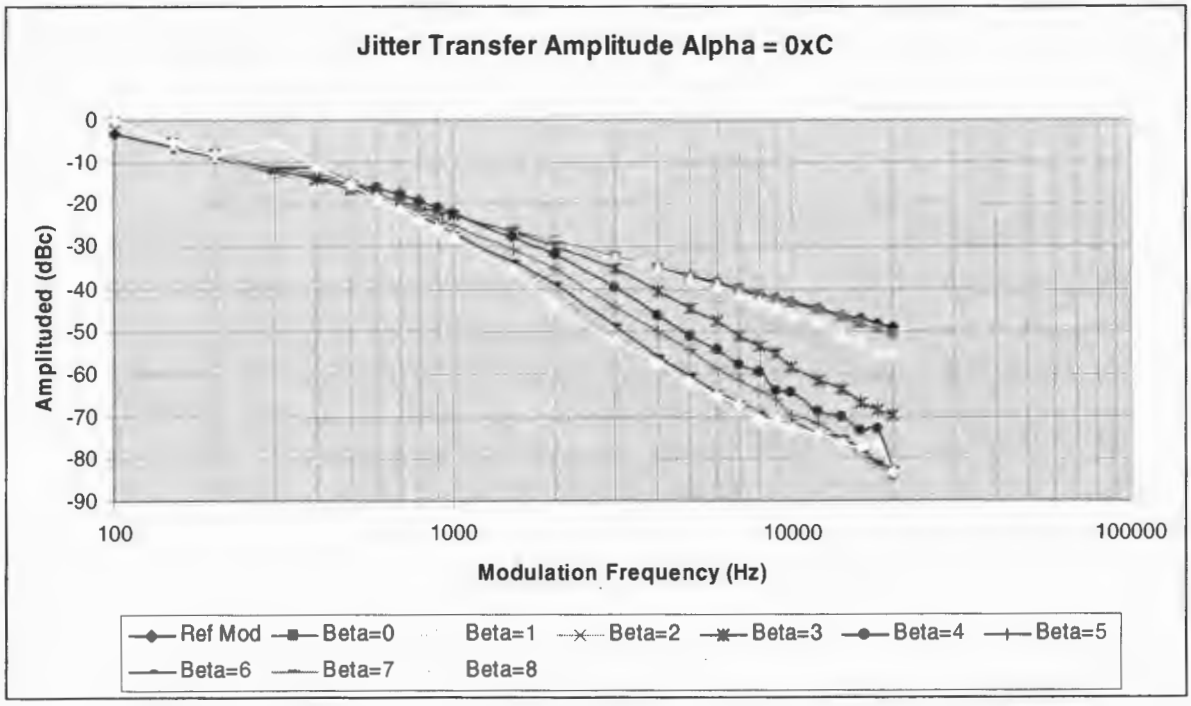

Figure 31 : Jitter Transfer Amplitude Alpha=0xC

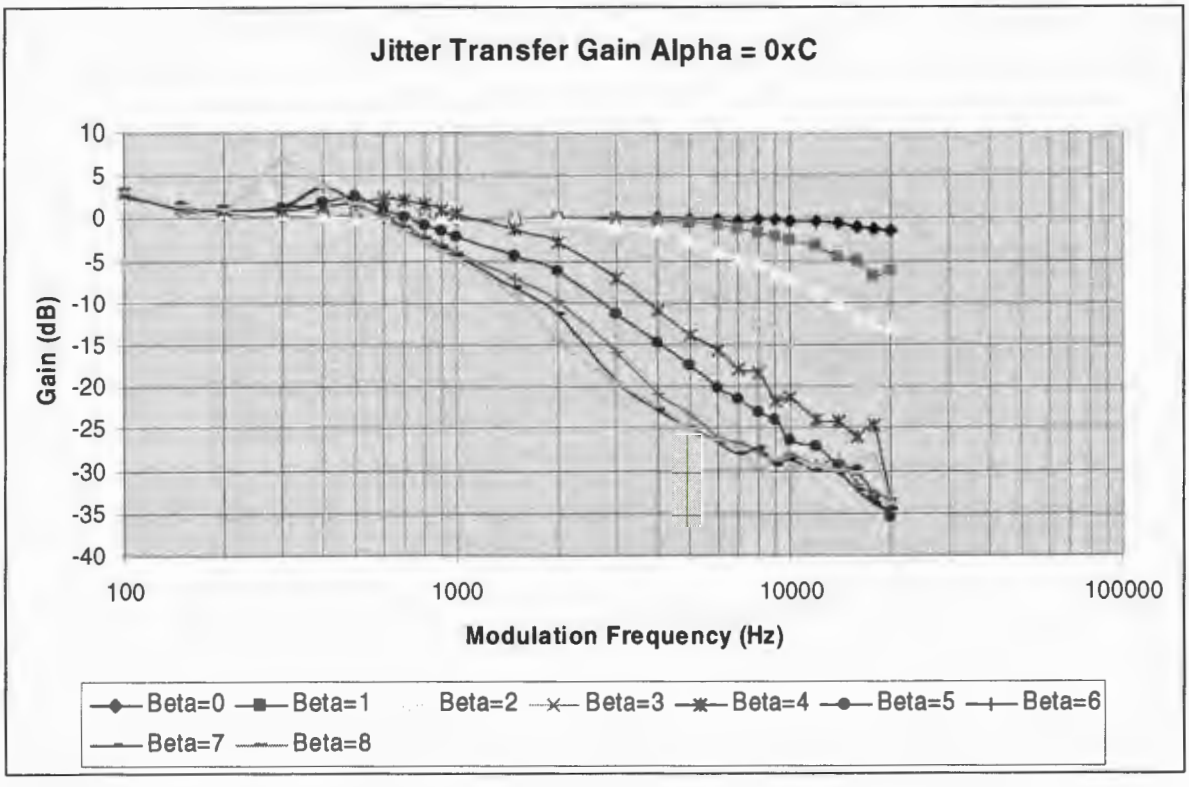

Figure 32 : Jitter Transfer Gain Alpha=0xC

In order to analyze the effects of the parameter Beta a fixed value for Alpha was chosen such that all possible values of Beta would result in stable operation of the ADPLL. Figure 31 and Figure 32 illustrate these measurements. Gain is calculated by subtracting the calculated reference modulation amplitude from the measured output amplitude for each modulation frequency. The data points taken at 200Hz and less should be considered to be marginal measurements since these measurements are taken near the limits of the spectrum analyzers specified operating range. Some false or invalid measurements were observed in this range during the course of testing. Additionally, measurements of amplitude less than -65dB are not as accurate since this is the approximate noise floor of the test system. With this in mind we can see that as Beta increases the closed loop bandwidth of the system decreases. Where as a Beta = 0 results in a bandwidth of approximately 10Khz; a Beta = 4 results in a bandwidth of 1.5Khz. It is also noticed that as the value of beta approaches the value of alpha the modulation gain starts to show peaking or positive gain.

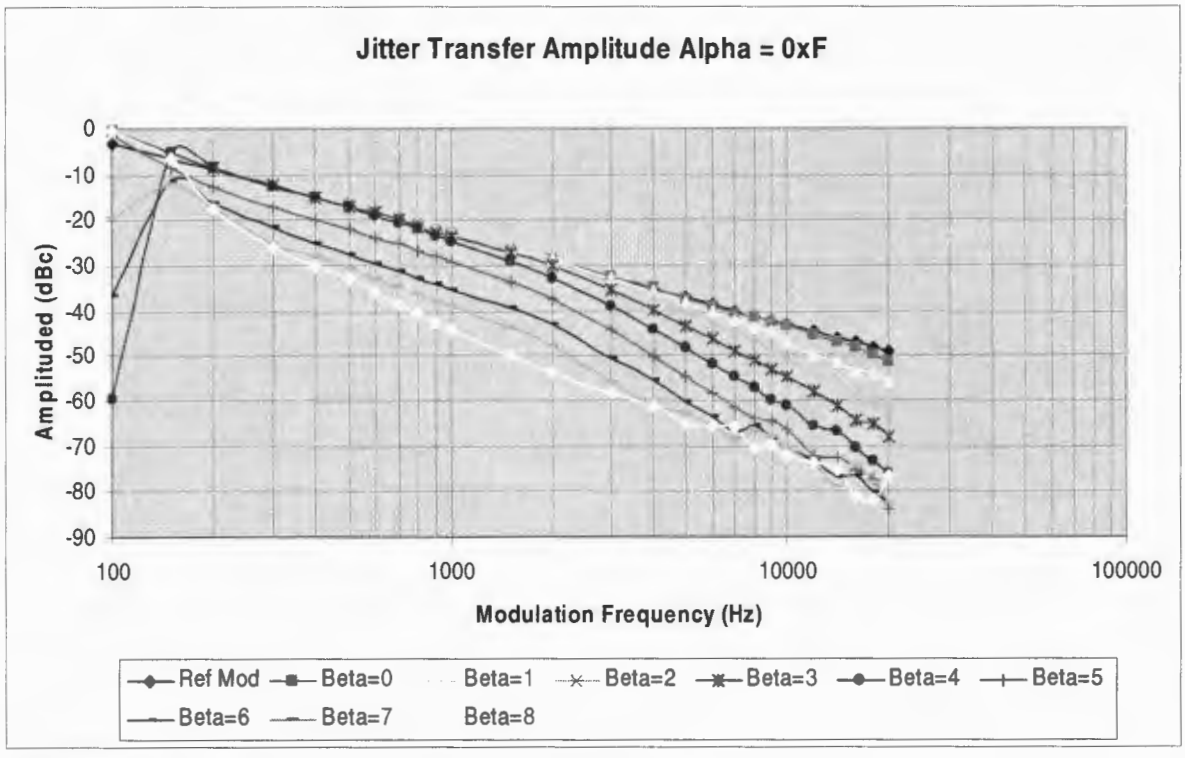

Figure 33 : Jitter Transfer Amplitude Alpha=0xF

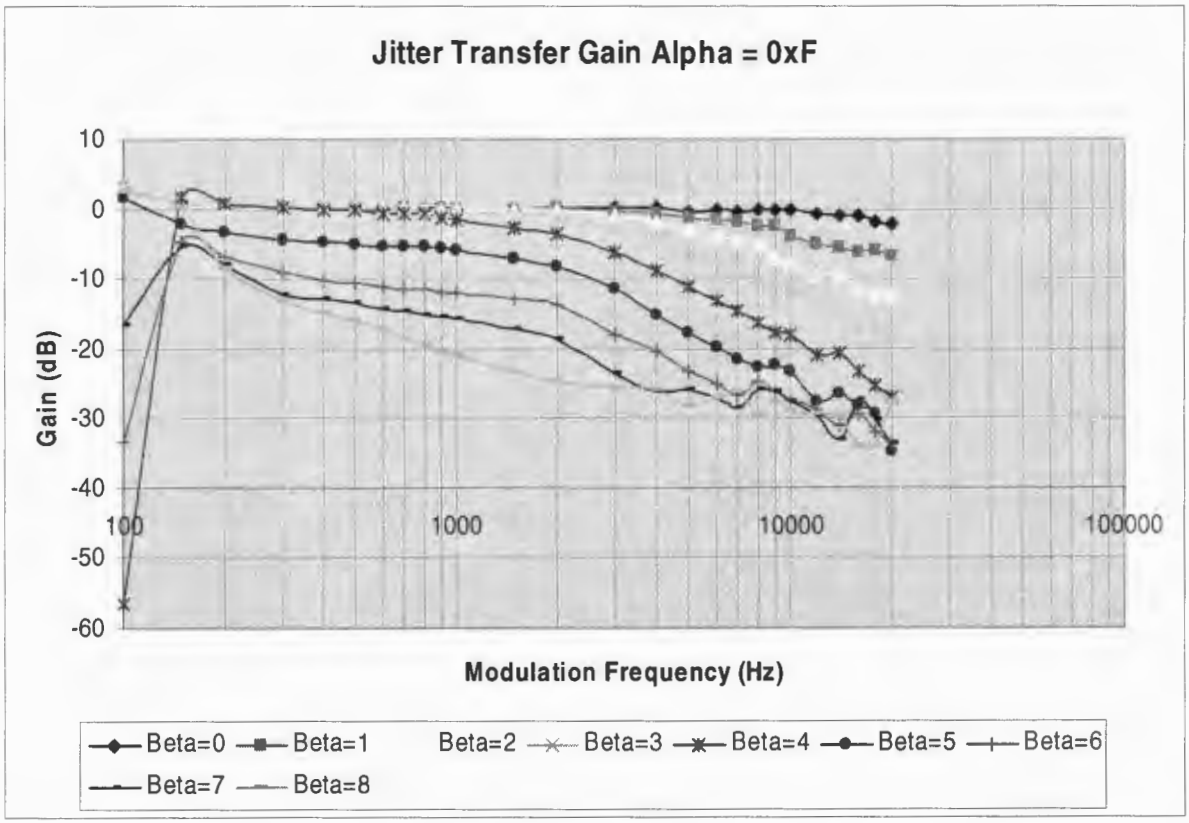

Figure 34 : Jitter Transfer Gain Alpha=0xF

These measurements where repeated with an Alpha=0xF which is its maximum value. It is shown that increasing Alpha also reduces closed-loop bandwidth of the system. This is shown by comparing Figure 34 and Figure 32. In fact it can be seen that with Alpha = 0xF and Beta > 4 the closed loop bandwidth of the system is less than what can be measured by the equipment used or less than 100hz.

In order to investigate this issue further data was taken with a fixed value of Beta=5 and Alpha was varied from 0x9 to 0xF. This data is shown in Figure 35 and Figure 36.

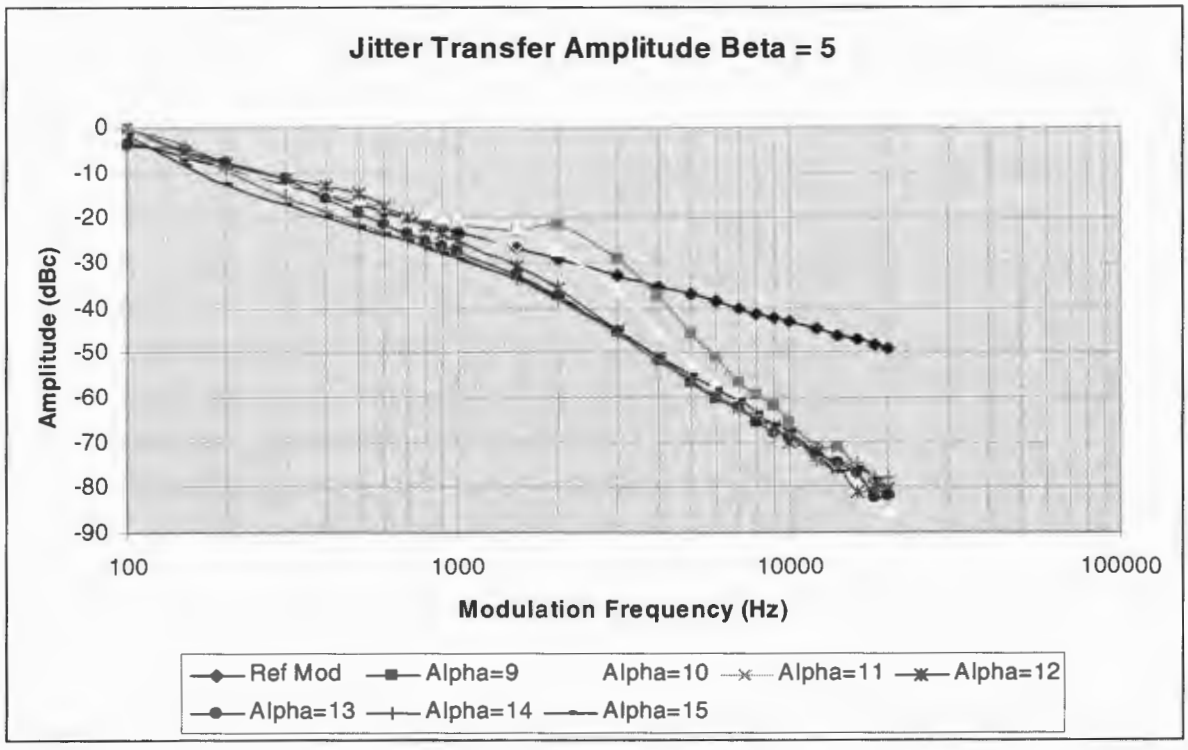

Figure 35 : Jitter Transfer Amplitude Beta=5

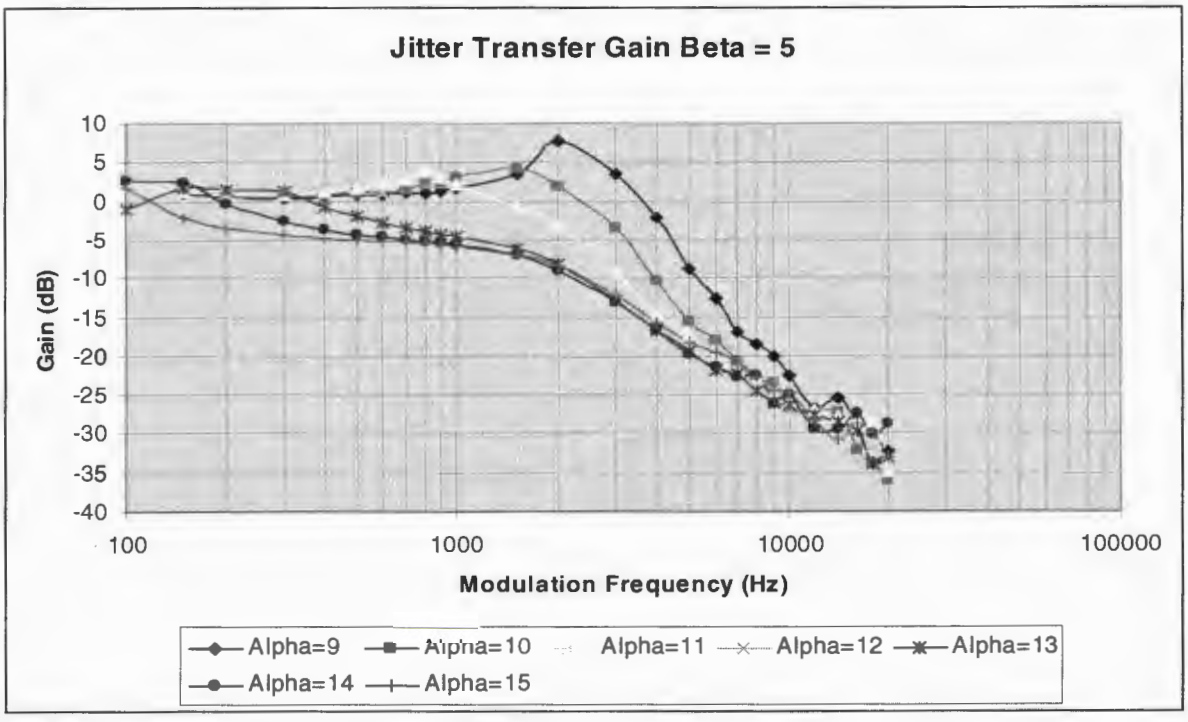

Figure 36 : Jitter Transfer Gain Beta=5

It is important to point out the increased peaking of the jitter transfer function as Alpha approaches Beta. In fact peaking is evident whenever Alpha < Beta + 8. It was also observed that the ADPLL appeared stable anytime Alpha < Beta + 3.

Next a value of Alpha and Beta were chosen for good stability and peaking performance and the divcnt parameter was changed from a value of 8 to a value of 0x18. This value did affect the closed loop bandwidth slightly; however the impact was significantly less pronounced than the effect of Alpha and Beta on closed-loop bandwidth. This can be seen in Figure 38. This affect is most likely due to the increased phase detector gain resulting from reduced update rate of the loop filter.

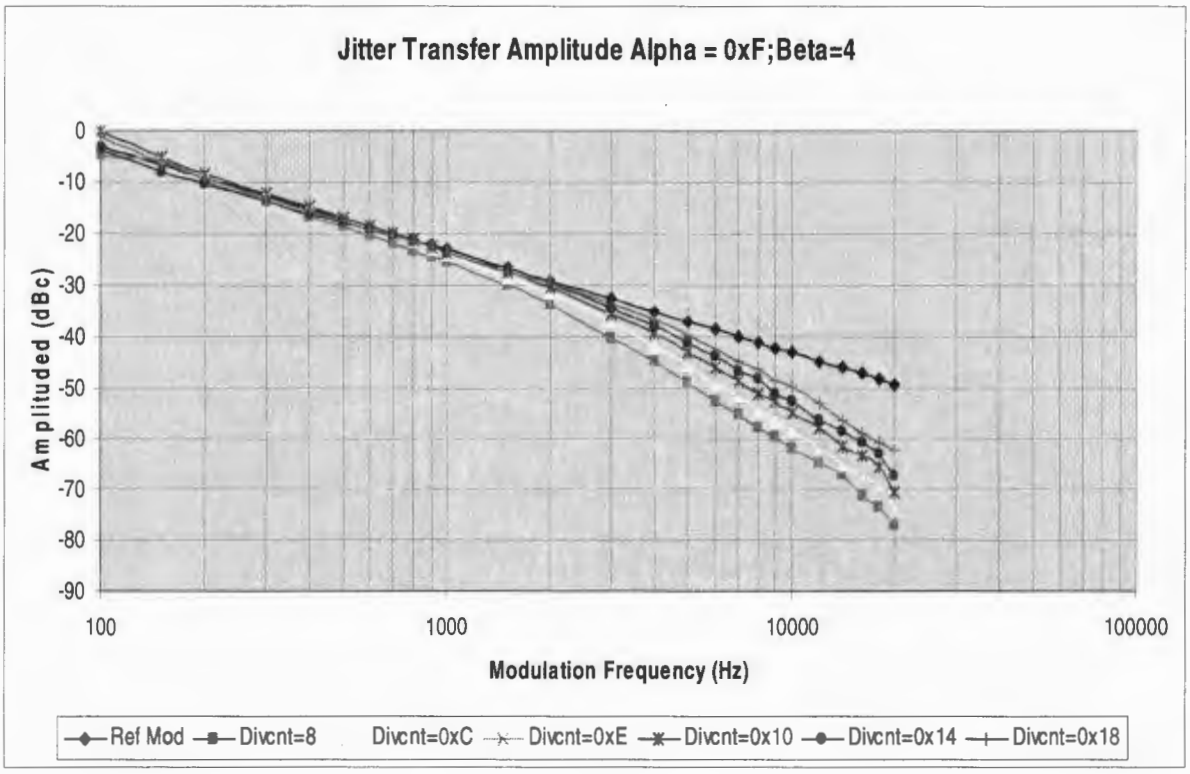

Figure 37 : Jitter Transfer Amplitude Alpha=0xF;Beta=4

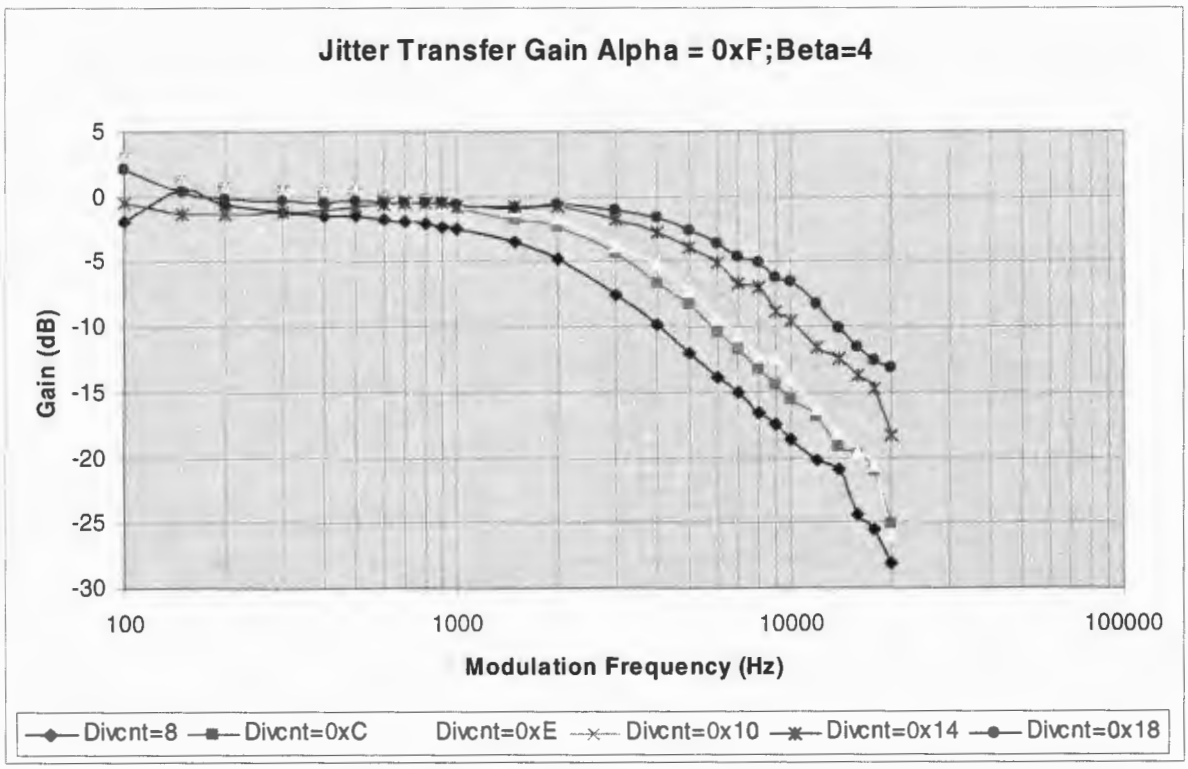

Figure 38 : Jitter Transfer Gain Alpha=0xF;Beta=4

3.5 Jitter Generation

Jitter generation was evaluated using the phase noise measurement capability of the spectrum analyzer in test setup #1. The bandwidth of consideration is from 10Khz to 10Mhz. The lower limit of 10Khz is the limit of the equipment. First the Reference Source was measured with no modulation present. The value of 0.4299 Degrees shown in Figure 39 is the total noise present in the band of interest. This equals 7.64ps RMS jitter from Equation (8).

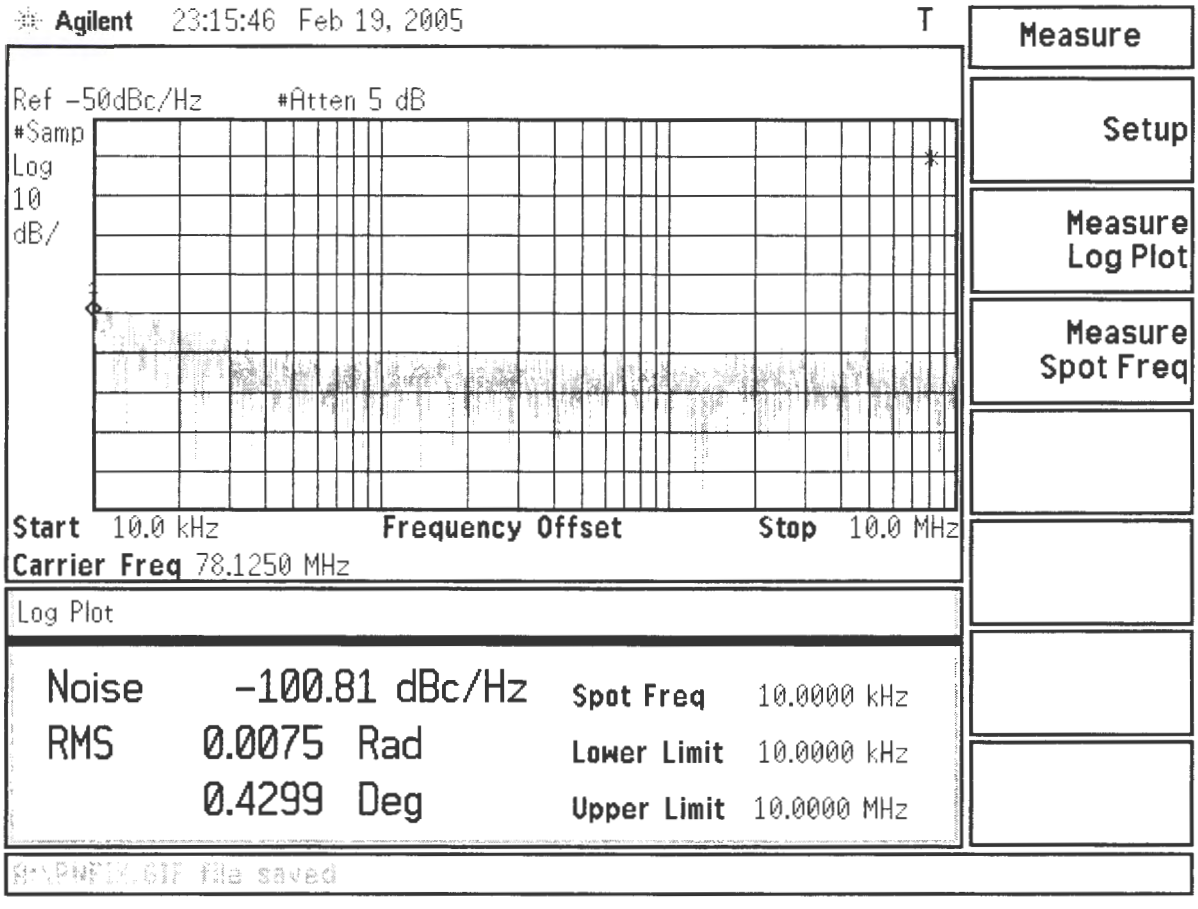

Figure 39 : Phase Noise Measurement Reference

The system was measured first with a fixed DAC value so that the VCXO and the drivers of the FPGA could be measured. Using Equation (8), a value of 2.22ps RMS jitter was calculated for the fixed DAC value. It is important to point out that when measured directly at the output of the VCXO this value is approximately 1ps RMS. The difference can be attributed to the noise present in the FPGA

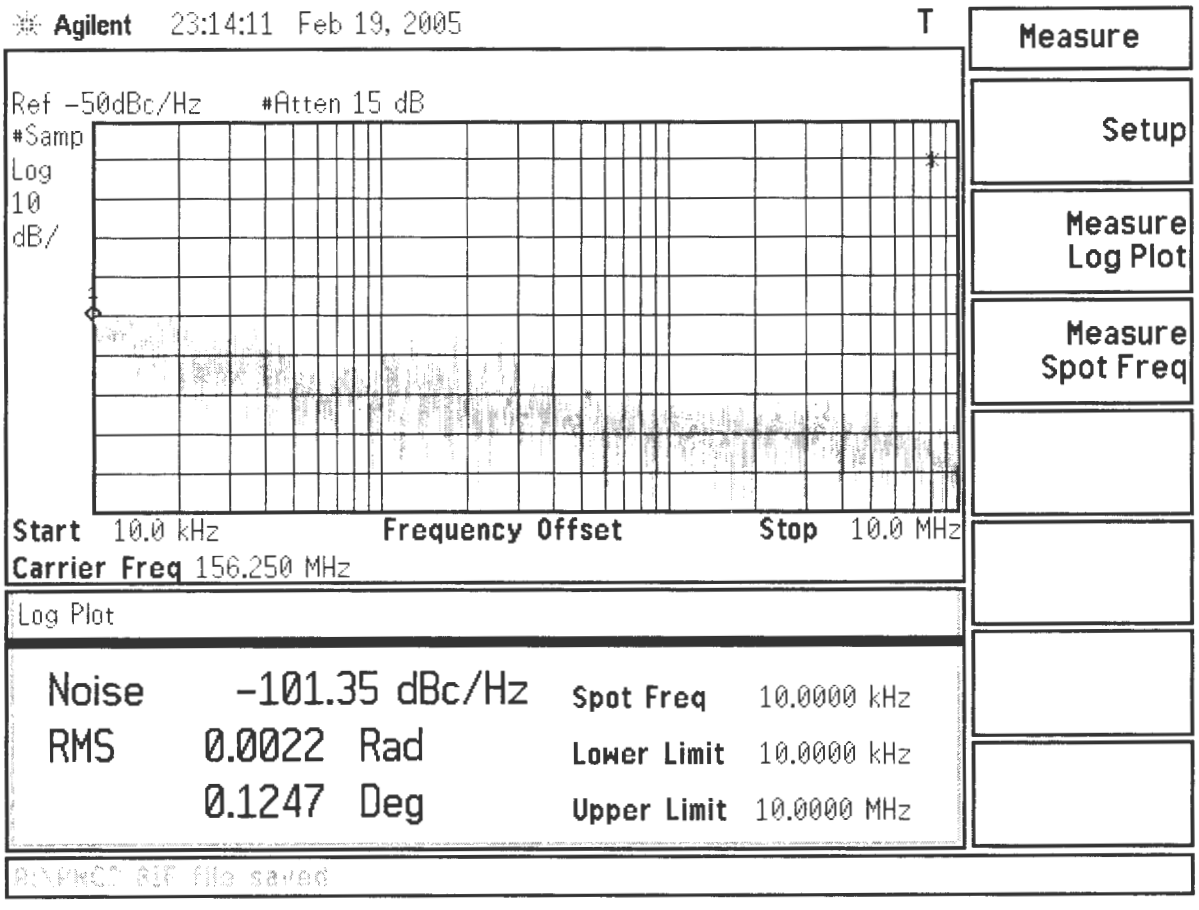

Figure 40 : Phase Noise Measurement Fixed DAC value

Next two sets of values for Alpha and Beta were chosen to illustrate the worst and best case phase noise possible in the system. The first measurement shown in Figure 41 represents the worst case phase noise. It has the widest closed-loop bandwidth and significant peaking. Using Equation (8) the peak to peak jitter is 12.84ps RMS. The second measurement shown in Figure 42 represents the best or more common case with more narrow bandwidth and minimal peaking. Its peak to peak jitter is 2.27ps RMS. This is better than the reference source and comparable to the fixed DAC value. It was observed that if the measurement was taken directly at the output of the VCXO the peak to peak jitter is less than 2ps RMS. Therefore it is felt that much of the phase noise present in the measurements is caused by the output and input drivers of the FPGA.

.

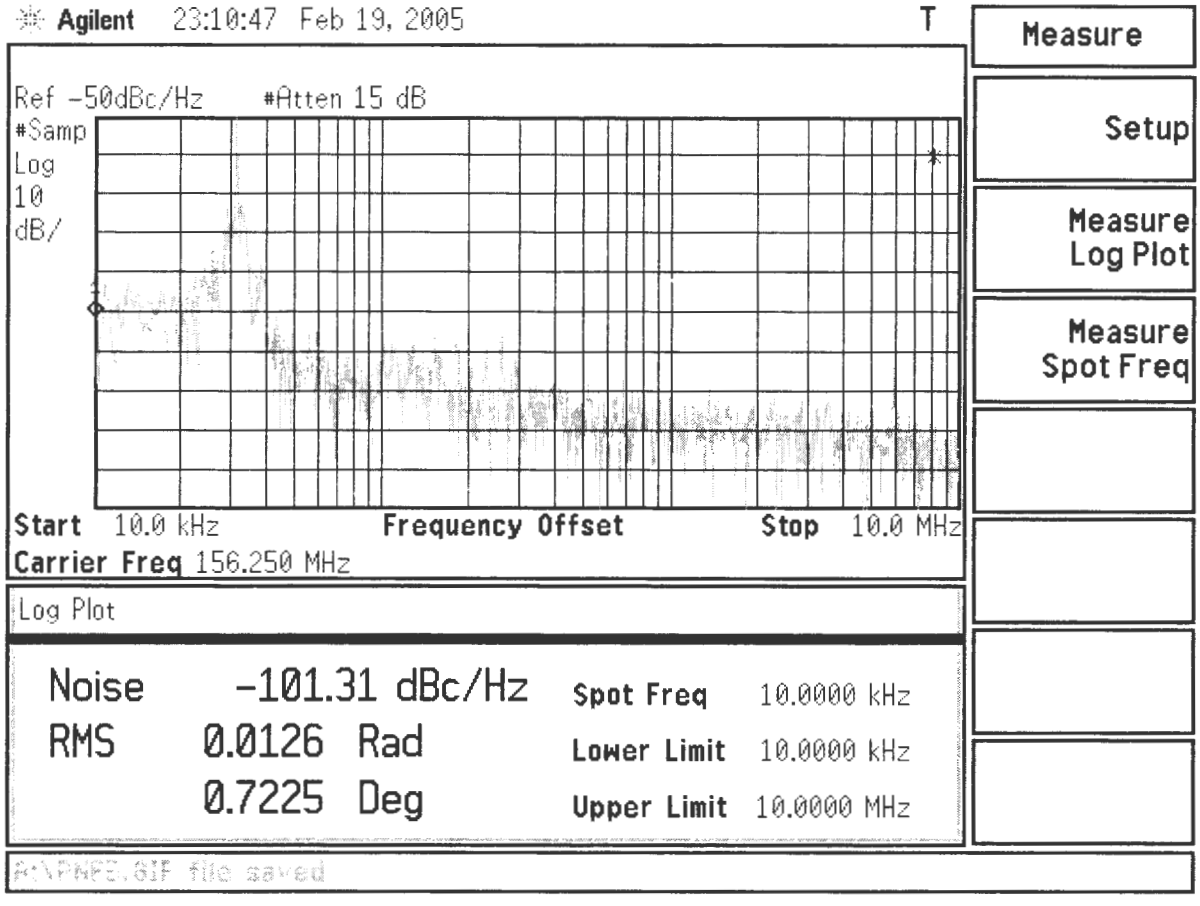

Figure 41 : Phase Noise Measurement Alpha=3;Beta=0

$$PN_{RMS} = \frac{1}{F_{vcxo}} * \frac{1}{360} * RMS_{\deg} \quad (8)$$

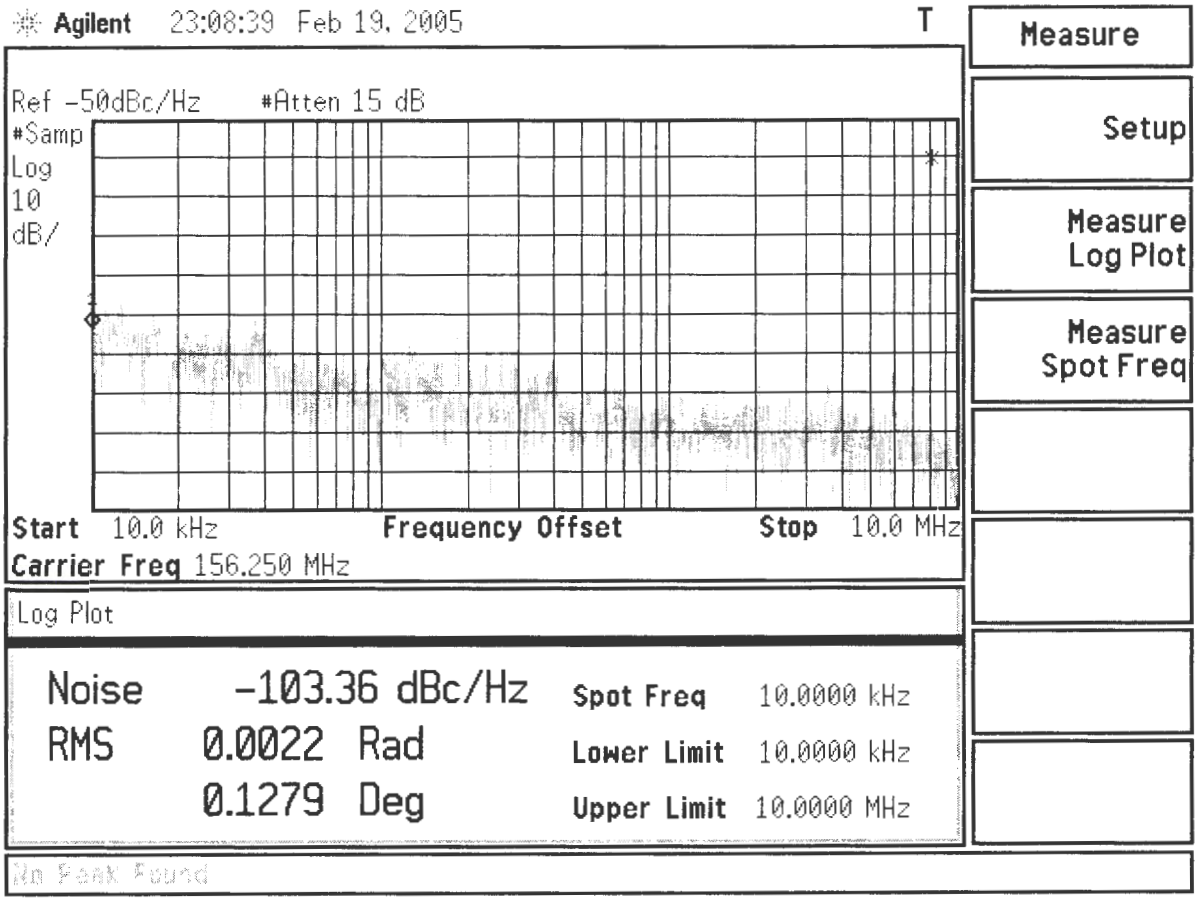

Figure 42 : Phase Noise Measurement Alpha=0xF;Beta=5

When we compare the measured values of jitter generation to the published papers this ADPLL exceeds the performance of many systems. The integrated ring oscillator based PLL's range from 14.6ps [7] to 15.9ps [8]. Published integrated LC oscillators operating at much higher frequencies range from 3.6ps [8] to 4.77ps [2]. It is expected that the performance of this system would be improved with higher frequencies.

3.6 Frequency Step Response

Frequency step response is an important measurement to monitor for PLL's. Test setup #1 was used to generate frequency step responses with different PLL parameter values. The oscilloscope captures are shown in Figure 43 to Figure 46. It should be noted that this is a very large step response and tends to push the ADPLL into non-linear operating points as can be seen by the triangular oscillations in the screen captures. Lee [2] gives a good analysis of this behavior which is caused by slewing. The slewing results because of the maximum rate at which the DAC value can be updated and the behavior of the bang-bang phase detector.

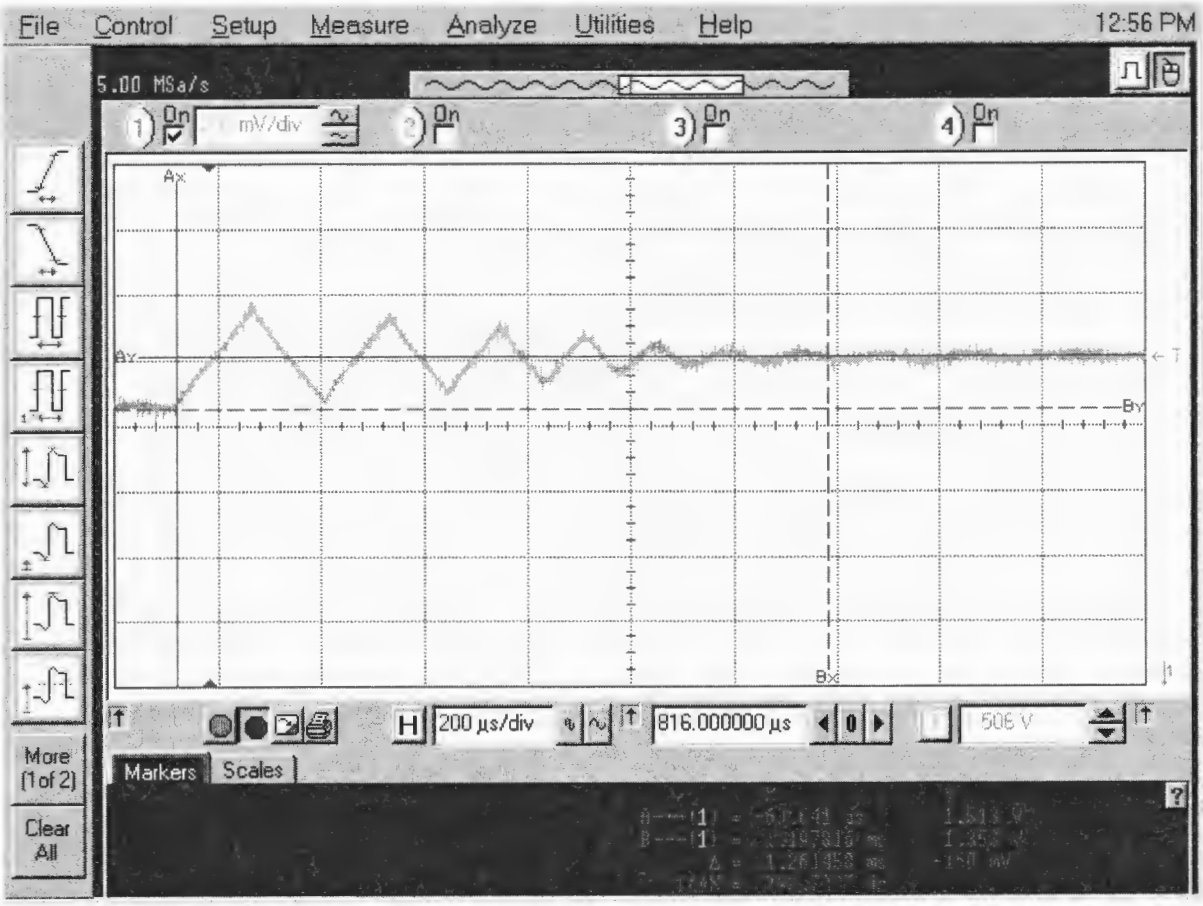

Figure 43 : 1000Hz Frequency Step Response Alpha=6;Beta=3

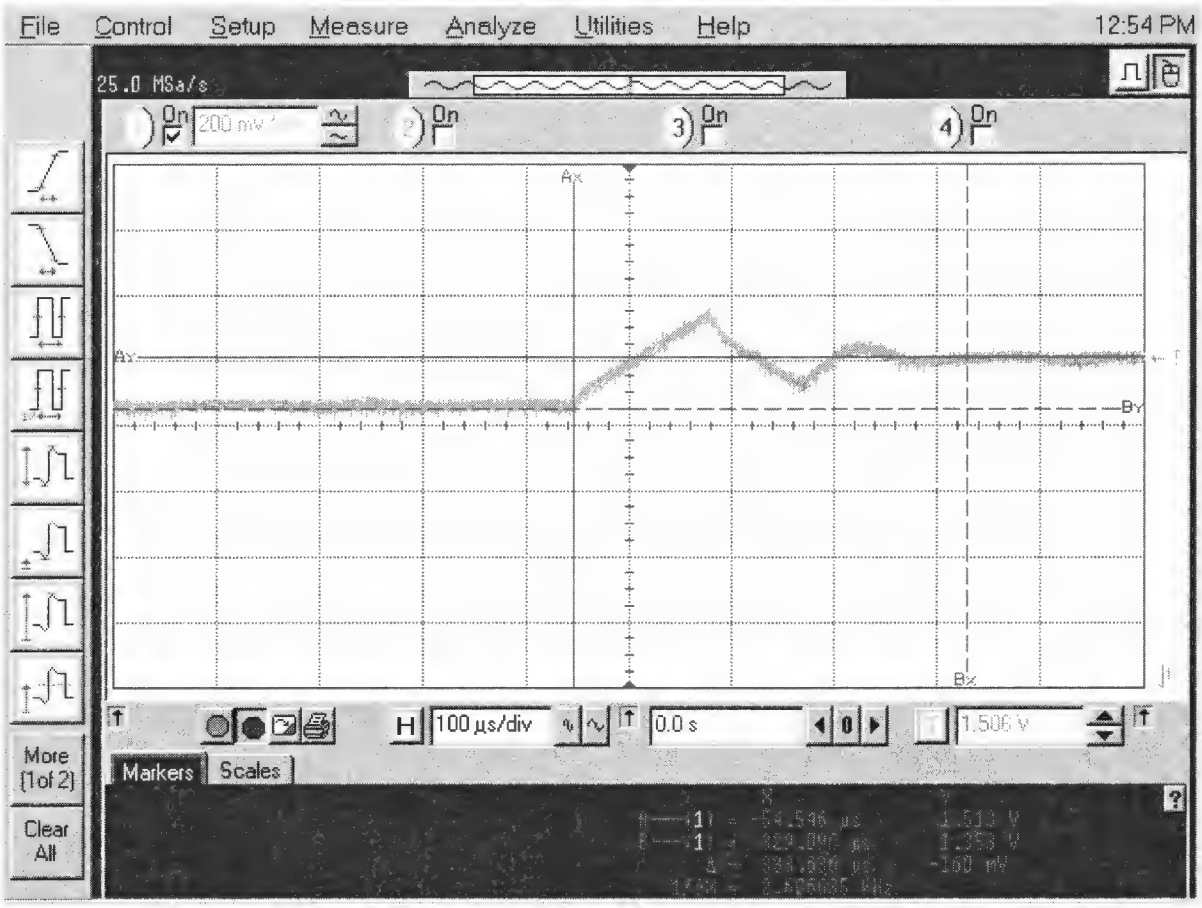

Figure 44 : 1000Hz Frequency Step Response Alpha=6;Beta=2

It can be seen that lock time increases dramatically when the value of beta is reduced. A smaller value of beta allows the DAC value to be increased faster because the gain of the first order path is much larger. Figure 46 shows an event which results from the large frequency step induced and the very low gain of both first and second order paths. This is not likely to occur in a real system.

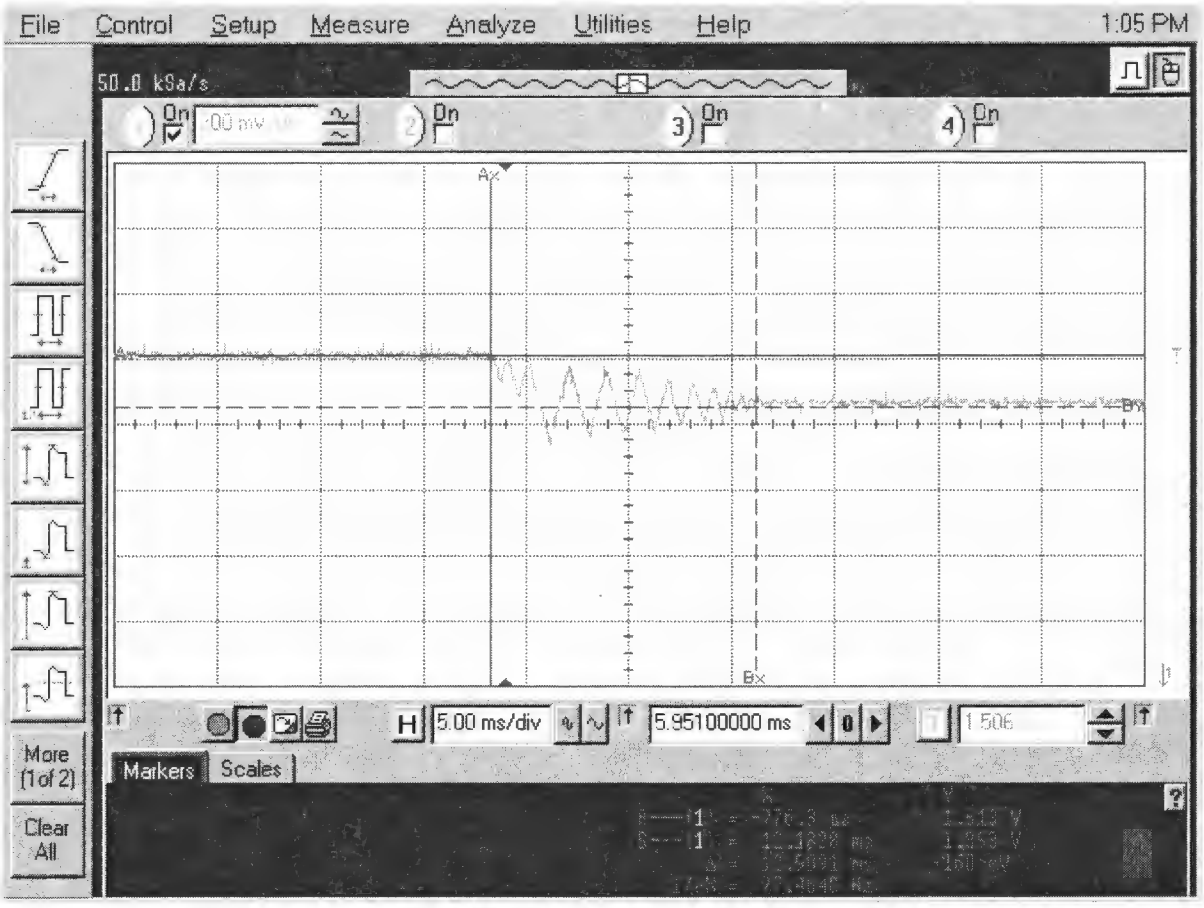

Figure 45 : Negative 1000Hz Frequency Step Response Alpha=9;Beta=5

The ADPLL responded by tracking the wrong direction resulting in an out of lock condition. The result is that the DAC value saturates. Eventually the loop forces the DAC value which is a 12 bit unsigned integer value to rollover. The loop then continues to track down until lock is acquired. This event is repeatable and only occurs with a positive step of 1Khz and with large values of Alpha.

Another issue that was observed is that as the value of Alpha is increased, the loop will loose lock momentarily when the value of alpha is changed. This occurs only when alpha is greater than 0x9. This is an issue because the expectation was that the parameters could be modified dynamically in order to produce fast lock times and then increased in order to have lower bandwidths. Changes to beta do not cause the loop to loose lock.

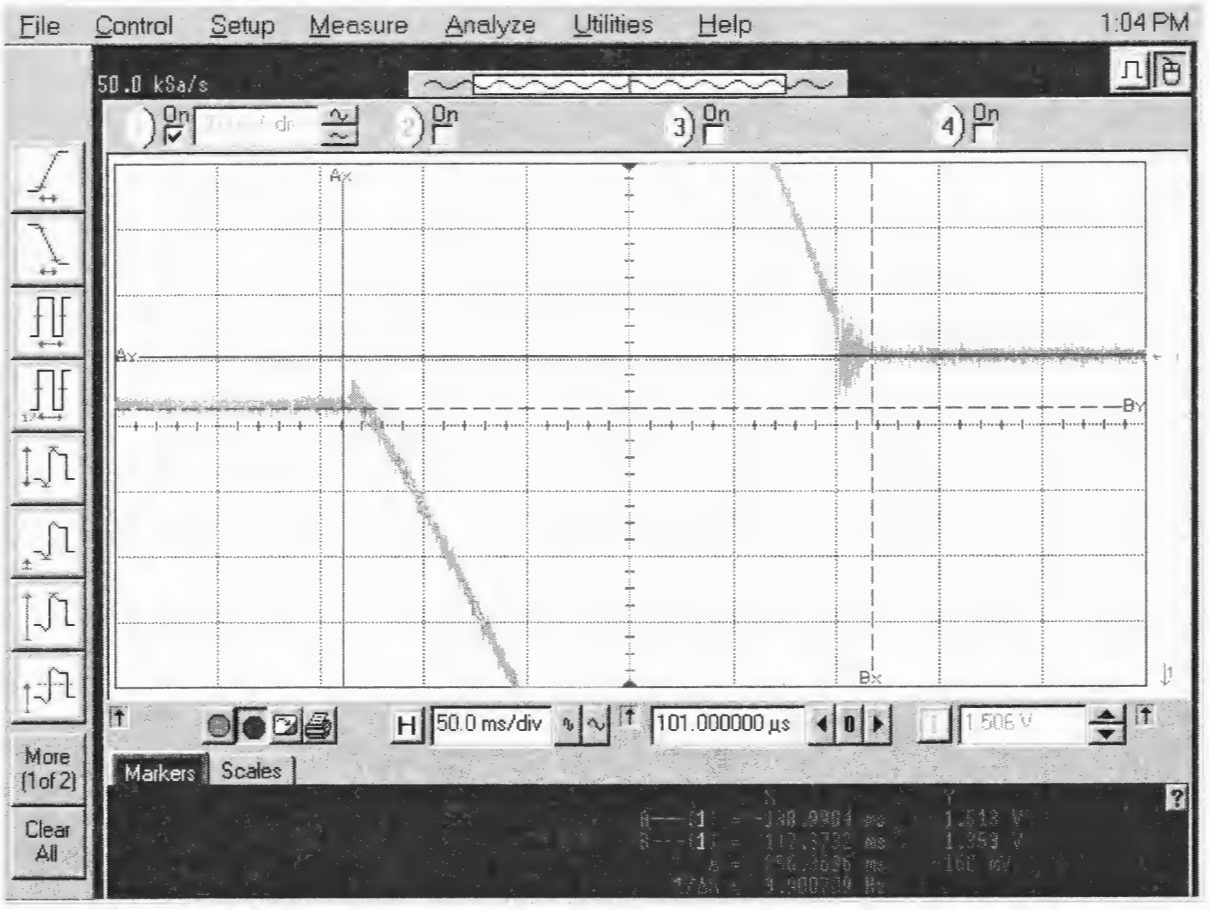

Figure 46 : Positive 1000Hz Frequency Step Response Alpha=9;Beta=5

3.7 Data Recovery Jitter Tolerance

Jitter tolerance is the ability of a clock and data recovery (CDR) system to receive data with significant amounts of jitter and still be able to accurately recover the data and clock signal. To perform this test, test setup #2 was used to provide a PRBS23 pattern driven into the reference signal input. The output data was monitored for errors and the input data was modulated with sinusoidal jitter. Figure 47 represents the maximum amount of jitter which was used and error free data received. The performance of a CDR system depends on tracking the jitter that is received. Since this system has such low bandwidth it was unable to recover data with greater than 0.39UI of modulation unless the loop bandwidth was significantly higher than the modulation frequency. In fact only the widest bandwidth settings provided adequate jitter tolerance performance. This shows that for acceptable jitter

tolerance performance the bandwidth should be approximately 10x the frequency of the required rolloff point for the required jitter tolerance.

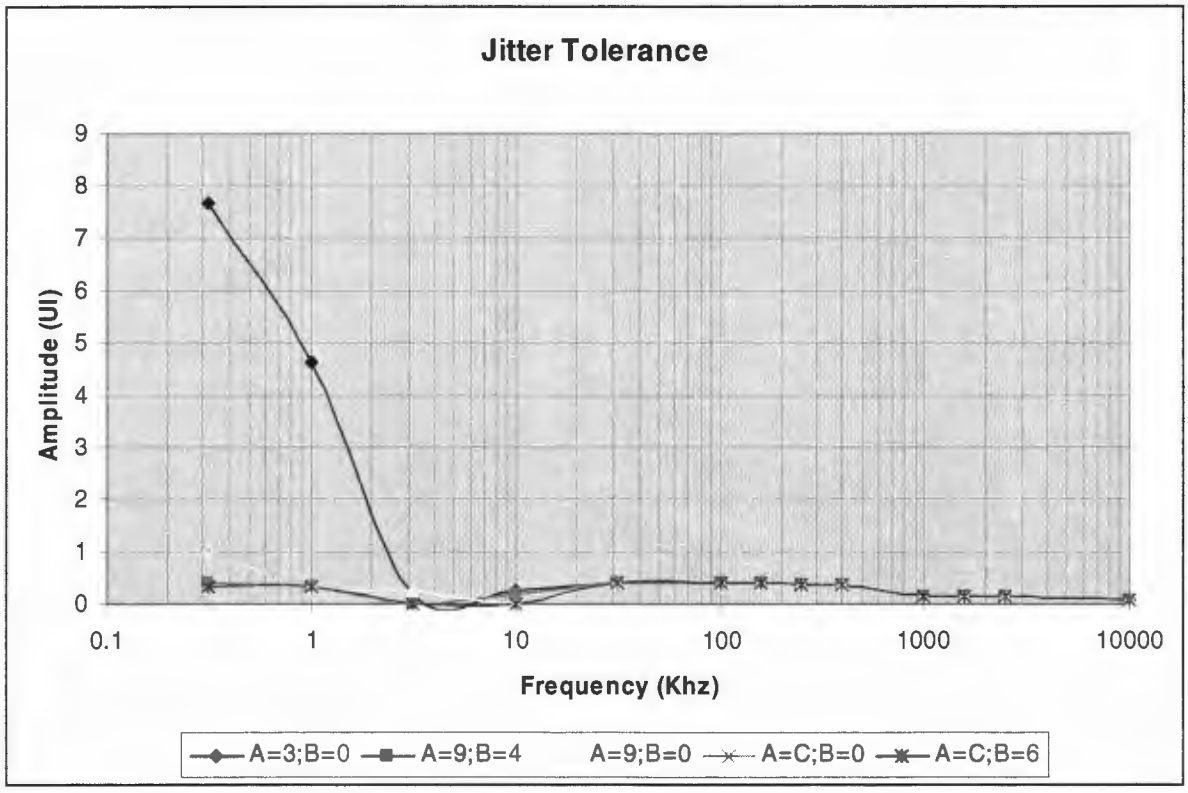

Figure 47 : Jitter Tolerance Plots

3.8 Comparison to Analytical Analysis

Measured phase noise was only slightly larger than the calculated quantization jitter calculated previously. In fact if measured directly at the output of the VCXO it is nearly equal. It was observed that the ADPLL became unstable when the value of Beta approached Alpha within a value of 3. When comparing this with Da Dalt's[3] analysis of minimum Beta we see very good correlation. Consider Table 10 which used a value of D=2 in Equation (4). The measured value column represents the maximum value of Beta where the ADPLL was observed to be stable and able to lock. The second column represents the value of α as it is used in Da Dalts equations. The third column is the calculated value of β from

DaDalts equations. The fourth column represents the value of Beta which is calculated from the third column. Notice that columns 4 and 5 are consistently different by a value of 2.

Table 10 : Comparison of minimum Beta

Alpha	2^-Alpha	Bmin	Beta-min Calculated	Measured Beta-min
7	0.007813	0.0195313	5.68	4
8	0.003906	0.0097656	6.68	5
9	0.001953	0.0048828	7.68	6
10	0.000977	0.0024414	8.68	7
11	0.000488	0.0012207	9.68	8

According to Da Dalt's [3] equation for optimal value of Beta equation (5), if we specify a value of Alpha equal to 0xC; the optimum value of Beta would be approximately 8. However from Figure 32 we see excessive peaking with any value of Beta greater than 3. However, consider that Da Dalt was examining jitter based on the trajectory and orbit of the closed loop system only. The peaking in the jitter transfer map has a different cause and effect. Therefore a trade-off must be made regarding the effects of the peaking versus effects of larger orbits or less than optimum Beta in order to choose the best performing loop parameters.

3.9 Conclusion

In this chapter data was presented which showed the implemented design operates well with a variety of loop parameter values. Further the ADPLL performs better than published integrated solutions with respect to jitter generation. This is due in no small order to the low noise performance of the crystal based VCXO. Further it is shown that the ADPLL operates effectively as a noise reducing filter and clock multiplying circuit due to the ability to produce very low bandwidth loops.

Chapter 4 Conclusion

In this thesis an all digital PLL based on DAC and VCXO technology was analyzed, designed, and implemented using FPGA technology. The DPLL was characterized with different programmable settings and shown to function well in different applications. As part of the characterization of the ADPLL an automated jitter transfer program was developed to analyze the loop performance. This automation is important in order to produce accurate and repeatable characterization data. The ADPLL was shown to have flexibility in loop bandwidth and was stable across a wide range of parameter settings. The design was shown to operate as both noise filtering, clock multiplying and data recovery applications. The data recovery application's performance was limited by the narrow bandwidth of the implemented design. The performance of the system would benefit with the use of a faster DAC and wider bandwidth. The ADPLL implemented is more suitable for jitter filtering where it was shown to reduce input jitter significantly and have very low jitter generation capability. The Verilog HDL code provided in the thesis can be synthesized to any digital technology using standard industry CAD tools. In this thesis we used a Xilinx Virtex-II ProX FPGA; however the design could be re-processed for the latest Virtex 4 FPGA or designed into an Application Specific Integrated Circuit (ASIC). Additionally, the design is not limited to the particular DAC and VCXO used in this implementation. Most VCXO or VCSO oscillators share the same footprint and pin-out. Therefore, changing operating rate of the design would be straightforward. Further, there exist many DAC's which operate with the same SPI interface as used in this design. Changing the DAC used may require new PCB design due to footprint and pin-out differences between these DAC's. Furthermore, the circuit is not limited to the operation speed of the DAC or digital circuits. It is possible to divide a very high speed VCSO using an external component before entering the FPGA without dramatically affecting performance.

The cost of the implemented design is estimated at less than $25 which includes the cost of the DAC and VCXO. It is assumed that the FPGA or ASIC would be a pre-existing component which could be expanded to support the added functionality. For comparison purposes, this is a reduction of at least 50% over existing solutions such as Texas Instruments

CDC7005 device and supporting components and over 100% versus existing Integrated Circuit Systems (ICS) devices.

In the thesis an innovative accumulating phase detector was introduced. This phase detector expanded the capabilities of a standard bang-bang phase detector to operate at a higher rate than the loop filter and DAC. This results in a lower update rate for the VCXO however it allows the phase detector to appear more linear than a standard bang-bang phase detector. This reduced rate for the DAC is important because most low cost DAC's have update rates far lower than the clock rates of most digital communication systems today.

4.1 Future Work

This thesis introduced an innovative solution for clocking in today's systems. The following is a list of potential future work.

1. Characterize the performance benefits of faster DAC.
2. Improve Clock and Data Recovery operation.
3. Measure low bandwidth performance using equipment capable of measuring in very low frequency band.
4. Implement loops with frequencies greater than 600Mhz.
5. Design of an automated control system which could dynamically adapt loop parameters.
6. Implementation using embedded multipliers of Xilinx Virtex 4.

Appendix A

4.2 Introduction

This Appendix contains the Verilog HDL source code for the implemented design. The function of each of these verilog modules is described in detail in Chapter 2.

4.3 Design Top-Level module : fpga_top.v

```
// FileName: fpga_top.v
//
// Author: Justin Gaither
//
// Begin Date: Tue Nov 30 15:08:16 2004
//
// Description: Top level system for All Digital PLL using external
//  DAC and VCXO
//
//
//
//                    Revision History
//-------------------------------------------------------------
// Date Modified            User Name          Full Name
//       Descrition of Changes
//-------------------------------------------------------------
// $Log: fpga_top.v,v $
// Revision 1.2  2005/01/31 22:10:57  jgaither
// Update for fixed vc, and clk and data to sma
//
// Revision 1.1  2005/01/19 20:26:49  jgaither
// Initial revision
//
//
//
//-------------------------------------------------------------
// NOTES:
//-------------------------------------------------------------
//
//
//-------------------------------------------------------------
`timescale 1ns/100ps
`ifdef TPDA
```

```
`else
 `define TPDB #0.1
 `define TPDA #0.1
`endif

module fpga_top (/*AUTOARG*/
  // Outputs
  SCLK, SDO, SYNC, clk_sel, clk_en, dataoutp, dataoutn, clkoutp,
  clkoutn,
  // Inputs
  reset, refclk_P, refclk_N, vcoclk_P, vcoclk_N
  );
  input reset;
  input refclk_P,refclk_N;
  input vcoclk_P,vcoclk_N;
  output SCLK,SDO,SYNC;
  output clk_sel,clk_en;
  output dataoutp,dataoutn;
  output clkoutp,clkoutn;


  wire [35:0] control0;
  wire [63:0] sync_in;
  wire [63:0] sync_out;

  wire [3:0] bitsel;
  wire       clkendac,isync;
  wire [15:0] vc,vcx;
  wire [8:0]  phase_error;
  reg        refsig;
  wire [31:0] integrator;

  IBUF rst_buf      (.O(rst)    , .I(reset));
  IBUFGDS refclk_buf (.O(refclki), .I(refclk_P), .IB(refclk_N));
  IBUFGDS vcoclk_buf (.O(vcoclki), .I(vcoclk_P), .IB(vcoclk_N));
//  BUFG refclk_bufg (.O(refclk) , .I(refclki));
  //BUFG clk_bufg    (.O(sclk)   , .I(sclki));
  OBUF clksel_buf    (.O(clk_sel), .I(1'b1));
  OBUF clken_buf     (.O(clk_en) , .I(1'b0));

  OBUFDS data_buf     (.O(dataoutp), .OB(dataoutn), .I(datax));
  OBUFDS clk_buf      (.O(clkoutp) , .OB(clkoutn) , .I(vcoclk));
```

```
// Can select divided pdclk or input clock
BUFGMUX vcoclk_bufg (.O(vcoclk), .I0(vcoclki), .I1(pdclk), .S(sync_out[25]));
BUFG vcoclk2_bufg (.O(vcoclk2), .I(vcoclki));


vcodiv vcodiv(
        // Outputs
        .clkendac       (clkendac),
        .rstcnt         (rstcnt),
        .bitsel         (bitsel[3:0]),
        .sync           (isync),
        .sclk           (sclk),
        .pdclk          (pdclk),
        // Inputs
        .clk            (vcoclk2),
        .divcnt         (sync_out[63:56]),
        .vcodiv         (sync_out[23:16])
        );

pd pd(
        // Outputs
        .data                   (datax),
        .phase_error                    (phase_error),
        // Inputs
        .refsig                 (refclki),
        .rstcnt                 (rstcnt),
        .vcoclk                 (vcoclk),
        .reset                  (rst)
        );


lp lp(
        // Outputs
        .vc                     (vc[15:0]),
        .integrator             (integrator),
        // Inputs
        .clk                    (vcoclk),
        .clken                          (clkendac),
        .phase_error                    (phase_error),
        .beta                   (sync_out[55:52]),
        .alpha                          (sync_out[51:48]),
        .rstcnt                 (rstcnt),
        .rstint                 (sync_out[45])
```

```
        );

  assign vcx = (sync_out[24]) ? vc : sync_out[15:0];

  spi spi(
          // Outputs
          .SYNC                 (SYNC),
          .SDO                          (SDO),
          .SCLK                 (SCLK),
          .dout                 (dout),
          .syn                  (syn),
          // Inputs
          .clk                  (vcoclk),
          .sclki                (sclk),
          .data                 (vcx[15:0]),
          .isync                (isync),
          .clkendac             (clkendac),
          .bitsel               (bitsel[3:0])
          );


  assign  sync_in = {isync,phase_error,vc,integrator};

// Chipscope Pro Cores

  //-----------------------------------------------------------------
  //
  //  ICON core instance
  //
  //-----------------------------------------------------------------
  icon1 i_icon
    (
      .control0(control0)
    );

  //-----------------------------------------------------------------
  //
  //  VIO core instance
  //
  //-----------------------------------------------------------------
  vio4 i_vio
    (
      .control(control0),
```

```
    .clk(vcoclk2),
    .sync_in(sync_in), // 64 bits
    .sync_out(sync_out) // 64 bits
  );


endmodule // fpga_top
```

4.4 Accumulating Phase Detector Module : pd.v

```
// FileName: pd.v
//
// Author: Justin Gaither
//
// Begin Date: Thu Dec 23 15:15:40 2004
//
// Description: Accumulating Bang-Bang Phase detector
//
//
//
//                    Revision History
//-----------------------------------------------------------------
// Date Modified          User Name          Full Name
//       Descrition of Changes
//-----------------------------------------------------------------
// $Log: pd.v,v $
// Revision 1.1  2005/01/31 22:10:57  jgaither
// Initial revision
//
// Revision 1.1  2005/01/19 20:26:49  jgaither
// Initial revision
//
//
//
//-----------------------------------------------------------------
// NOTES:
//-----------------------------------------------------------------
//
//
//-----------------------------------------------------------------
`timescale 1ns/100ps
```

```
`ifdef TPDA
`else
 `define TPDB #0.1
 `define TPDA #0.1
`endif

module pd (/*AUTOARG*/
  // Outputs
  data, phase_error,
  // Inputs
  refsig, rstcnt, vcoclk, reset
  );
  input refsig;
  input rstcnt;
  input vcoclk;
  output data;
  output [8:0] phase_error;
  input  reset;

  reg a,b,t,ta;
  reg up, down;
  wire data;
  reg [8:0] phase_error;
  reg [15:0] upcnt,dncnt;
  wire [15:0] new_pe;

  initial begin
    upcnt = 0;
    dncnt = 0;
    phase_error = 0;
  end

  assign data =  a;


//-----------------------------------------------------------
// Standard Bang-Bang Phase Detector
//-----------------------------------------------------------

  always@(negedge vcoclk or posedge reset)
    if(reset) begin
        ta <= `TPDB 0;
    end
```

```
    else begin
        ta <= `TPDB refsig;
    end

  always@(posedge vcoclk or posedge reset)
    if(reset) begin
        a <= `TPDB 0;
        b <= `TPDB 0;
        t <= `TPDB 0;
    end
    else begin
        b <= `TPDB refsig;
        a <= `TPDB b;
        t <= `TPDB ta;
   end

//-----------------------------------------------------------------
// Decode phase detector outputs
//-----------------------------------------------------------------

  always@(a or b or t)
    case({a,t,b})
        3'b000 : begin // no trans
          up = 0;
          down = 0;
        end
        3'b001 : begin // too fast
          up = 0;
          down = 1;
        end
        3'b010 : begin // invalid
          up = 1;
          down = 1;
          $display("Error in PFD %b %b %b %t",a,t,b,$time);
        end
        3'b011 : begin // too slow
          up = 1;
          down = 0;
        end
        3'b100 : begin // too slow
          up = 1;
          down = 0;
        end
```

```
        3'b101 : begin // invalid
          up = 1;
          down = 1;
          $display("Error in PFD %b %b %b %t",a,t,b,$time);
        end
        3'b110 : begin // too fast
          up = 0;
          down = 1;
        end
        3'b111 : begin // no trans
          up = 0;
          down = 0;
        end
    endcase // case(a,t,b)


//-----------------------------------------------------------------
// Up and Down Counters
//-----------------------------------------------------------------

  always@(posedge vcoclk) begin
    if(rstcnt) begin
        upcnt <= `TPDB 16'h0000;
        dncnt <= `TPDB 16'h0000;
        phase_error  <= `TPDB {new_pe[15],new_pe[7:0]};
    end
    else if(up & !down)
        upcnt <= `TPDB upcnt + 1;
    else if(down & !up)
        dncnt <= `TPDB dncnt + 1;
  end

  sub subtract ( // subtractor from coregen
       .A(upcnt), // unsigned
       .B(dncnt), // unsigned
       .S(new_pe));


endmodule // pd
```

4.5 Loop Filter Module : lp.v

```
// FileName: lp.v
//
// Author: Justin Gaither
//
// Begin Date: Tue Nov  9 10:04:11 2004
//
// Description:
//
//
//
//                    Revision History
//-----------------------------------------------------------------
// Date Modified            User Name          Full Name
//       Descrition of Changes
//-----------------------------------------------------------------
// $Log: lp.v,v $
// Revision 1.2  2005/01/31 22:10:57  jgaither
// Update for fixed vc, and clk and data to sma
//
// Revision 1.1  2005/01/19 20:26:49  jgaither
// Initial revision
//
//
//
//-----------------------------------------------------------------
// NOTES:
//-----------------------------------------------------------------
//
//
//-----------------------------------------------------------------
`timescale 1ns/100ps
`ifdef TPDA
`else
 `define TPDB #0.1
 `define TPDA #0.1
`endif

module lp (/*AUTOARG*/
   // Outputs
   vc, integrator,
   // Inputs
```

```
  clk, clken, phase_error, beta, alpha, rstcnt, rstint
  );
  input clk;
  input clken;
  input [8:0] phase_error;
  input [3:0] beta;
  input [3:0] alpha;
  input rstcnt;
  output [15:0] vc;
  output [31:0] integrator;
  input         rstint;

  reg [15:0]   vc;
  reg [31:0]   integrator;
  reg          load;
  reg [11:0]   b1,a1;
  wire [31:0] new_int;
  wire [11:0] new_vc,new_vcx;
  wire [11:0] vc_new;


  initial begin
    integrator = 32'h00000000;
    vc = 16'd1676;
  end

// register the control voltage(vc) and integrator
  always@(posedge clk) begin
    if (rstcnt)
        load <= `TPDB 1;
    else if(clken)
        load <= `TPDB 0;
    if (rstint)
        integrator <= `TPDB 32'h00000000;
    else if(load & clken) begin
        integrator <= `TPDB new_int;
        vc <= `TPDB vc_new;
        //$display("%d %x %x",vc,integrator,phase_error);
    end
  end

  add12 add12 ( // signed adder from coregen
       .A(b1), // signed
```

```
        .B(a1), //signed
        .S(new_vc));

  // integrator should drive to zero value
  add16 add16 ( // signed adder from coregen
        .A(integrator),  // signed
        .B(phase_error), // signed
        .S(new_int));

  // Simple way to saturate the Control Voltage similar to what an Analog filter
would do.
  assign `TPDA vc_new = (vc[11:8] == 4'hf & new_vc[11:8] == 4'h0) ? 16'h0FFF :
(vc[11:8] == 4'h0 & new_vc[11:8] == 4'hf) ? 16'h0000 : {4'h0,new_vc[11:0]};


//----------------------------------------------------------------
// Gain Stages
//----------------------------------------------------------------

  always@(posedge clk)
    case(beta)
        4'h0 : b1 = {{3{phase_error[8]}},phase_error}; // keep sign
        4'h1 : b1 = {{4{phase_error[8]}},phase_error[8:1]};
        4'h2 : b1 = {{5{phase_error[8]}},phase_error[8:2]};
        4'h3 : b1 = {{6{phase_error[8]}},phase_error[8:3]};
        4'h4 : b1 = {{7{phase_error[8]}},phase_error[8:4]};
        4'h5 : b1 = {{8{phase_error[8]}},phase_error[8:5]};
        4'h6 : b1 = {{9{phase_error[8]}},phase_error[8:6]};
        4'h7 : b1 = {{10{phase_error[8]}},phase_error[8:7]};
        4'h8 : b1 = {{11{phase_error[8]}},phase_error[8:8]};
        4'h9 : b1 = 12'h0;
        4'ha : b1 = 12'h0;
        4'hb : b1 = 12'h0;
        4'hc : b1 = 12'h0;
        4'hd : b1 = 12'h0;
        4'he : b1 = 12'h0;
        4'hf : b1 = 12'h0;
    endcase // case(beta)
  always@(posedge clk)
    case(alpha)
        4'h0 : a1 = new_int[11:0]; // keep sign
        4'h1 : a1 = new_int[12:1];
        4'h2 : a1 = new_int[13:2];
```

```
        4'h3 : a1 = new_int[14:3];
        4'h4 : a1 = new_int[15:4];
        4'h5 : a1 = new_int[16:5];
        4'h6 : a1 = new_int[17:6];
        4'h7 : a1 = new_int[18:7];
        4'h8 : a1 = new_int[19:8];
        4'h9 : a1 = new_int[20:9];
        4'ha : a1 = new_int[21:10];
        4'hb : a1 = new_int[22:11];
        4'hc : a1 = new_int[23:12];
        4'hd : a1 = new_int[24:13];
        4'he : a1 = new_int[25:14];
        4'hf : a1 = new_int[26:15];
    endcase // case(beta)


endmodule // div8
```

4.6 Serial Peripheral Interface module : spi.v

```
// FileName: spi.v
//
// Author: Justin Gaither
//
// Begin Date: Fri Nov 12 09:23:19 2004
//
// Description: Serial Peripheral Interface
//
//
//
//                    Revision History
//-------------------------------------------------------------
// Date Modified            User Name          Full Name
//       Descrition of Changes
//-------------------------------------------------------------
// $Log: spi.v,v $
// Revision 1.1  2005/01/19 20:26:49  jgaither
// Initial revision
//
//
//
```

```
//------------------------------------------------------------
// NOTES:
//------------------------------------------------------------
//
//
//------------------------------------------------------------
`timescale 1ns/100ps
`ifdef TPDA
`else
 `define TPDB #0.1
 `define TPDA #0.1
`endif

module spi(/*AUTOARG*/
  // Outputs
  SYNC, SDO, SCLK, dout, syn,
  // Inputs
  clk, data, isync, sclki, clkendac, bitsel
  );
  input clk;
  input [15:0] data;
  input        isync;
  input        sclki;
  input        clkendac;
  input [3:0]          bitsel;
  output       SYNC;
  output       SDO;
  output       SCLK;
  output       dout,syn;

  reg          syn,dout,en,busy;
  reg [3:0]    cnt;
  reg [15:0]   datav;
  reg          clko;

  OBUF sclk_buf (.O(SCLK), .I(clko));
  OBUF sdo_buf  (.O(SDO),  .I(dout));
  OBUF sync_buf (.O(SYNC), .I(syn));

  always@(posedge clk) begin
    clko <= `TPDB sclki;
    if(clkendac) begin
        if(!isync)
```

```
        datav <= `TPDB data;
      syn <= `TPDB ~isync;
      dout <= `TPDB datav[bitsel];
    end
  end


endmodule // spi
```

4.7 VCO divider Module : vcodiv.v

```
// FileName: vcodiv.v
//
// Author: Justin Gaither
//
// Begin Date: Tue Nov  9 10:04:11 2004
//
// Description: Divides the clock from vco and generates pulses and bitsel to
//   SPI interface
//
//
//                    Revision History
//-----------------------------------------------------------------
// Date Modified          User Name          Full Name
//       Descrition of Changes
//-----------------------------------------------------------------
// $Log: vcodiv.v,v $
// Revision 1.2  2005/01/31 22:10:57  jgaither
// added clk divider to pd
//
// Revision 1.1  2005/01/19 20:26:49  jgaither
// Initial revision
//
//
//
//-----------------------------------------------------------------
// NOTES:
//-----------------------------------------------------------------
//
```

```
//
//------------------------------------------------------------
`timescale 1ns/100ps
`ifdef TPDA
`else
 `define TPDB #0.1
 `define TPDA #0.1
`endif

module vcodiv(/*AUTOARG*/
  // Outputs
  clkendac, rstcnt, bitsel, sync, sclk, pdclk,
  // Inputs
  clk, divcnt, vcodiv
  );
  input clk;
  input [7:0] divcnt;
  input [3:0] vcodiv;
  output clkendac;
  output rstcnt;
  output [3:0] bitsel;
  output       sync;
  output       sclk;
  output       pdclk;

  reg [3:0]    bitsel;
  reg          rstcnt;
  wire         clkendac;
  reg [7:0]    cnt;
  reg [3:0]    vcnt;
  reg          sync;

  initial begin
    cnt = 1;
    vcnt = 1;
    bitsel = 4'hf;
    sync = 0;
    rstcnt = 0;
  end

  always@(posedge clk) begin
    vcnt <= `TPDB (vcnt < vcodiv) ? vcnt + 1 : 4'h1;
    cnt <= `TPDB (cnt < divcnt) ? cnt + 1 : 8'h01;
```

```
    if(clkendac & bitsel == 4'h3)
        rstcnt <= `TPDB 1;
    else // single clk pulse
        rstcnt <= `TPDB 0;
    if (clkendac) begin
        if(bitsel == 4'h0) begin
          bitsel <= `TPDB 4'hf;
          sync <= `TPDB 0;
        end
        else if(sync)
          bitsel <= `TPDB bitsel - 1;
        else
          sync <= `TPDB 1;
    end
  end

  assign clkendac = (cnt == divcnt);
  assign sclk = (cnt <= {1'b0,divcnt[7:1]});
  assign pdclk = (cnt <= {1'b0,divcnt[3:1]});


endmodule
```

References

1. "Phase-locked loops: a control centric tutorial" Abramovitch, D.;American Control Conference, 2002. Proceedings of the 2002, Volume: 1, 8-10 May 2002

2. "Analysis and modeling of bang-bang clock and data recovery circuits" Jri Lee; Kundert, K.S.; Razavi, B.; IEEE Journal of Solid-State Circuits, Volume: 39, Issue: 9, Sept. 2004

3. "A Design-Oriented Study of the Nonlinear Dynamics of Digital Bang–Bang PLLs " DaDalt, N.; IEEE Transactions on Circuits and Systems I: Regular Papers, [see also IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications], Volume: 52, Issue: 1, Jan. 2005

4. "A Monolithic 622Mb/s Clock Extraction Data Retiming Circuit" Lai, B.; Walker, R.C.; Solid-State Circuits Conference, 1991. Digest of Technical Papers 38th ISSCC

5. "A 5GHz CMOS digitally controlled oscillator with a 3GHz tuning range for PLL applications" Hasan, S.M.R.; Electronics, Circuits and Systems, 2003. ICECS 2003. Proceedings of the 2003 10th IEEE International Conference on, Volume: 1, 14-17 Dec. 2003

6. "A first multigigahertz digitally controlled oscillator for wireless applications" Staszewski, R.B.; Chih-Ming Hung; Leipold, D.; Balsara, P.T.; IEEE Transactions on Microwave Theory and Techniques, Volume: 51, Issue: 11, Nov. 2003

7. "A digitally controlled phase-locked loop with a digital phase-frequency detector for fast acquisition" In-Chul Hwang; Sang-Hun Song; Soo-Won Kim; IEEE Journal of Solid-State Circuits, Volume: 36, Issue: 10, Oct. 2001

8. "A performance comparison of PLLs for clock generation using ring oscillator VCO and LC oscillator in a digital CMOS process" Miyazaki, T.; Hashimoto, M.; Onodera, H.; Design Automation Conference, 2004. Proceedings of the ASP-DAC 2004. Asia and South Pacific, 27-30 Jan. 2004

9. "Low-phase-noise LC quadrature VCO using coupled tank resonators in a ring structure" ElSayed, A.M.; Elmary, M.I.; IEEE Journal of Solid-State Circuits, Volume: 36, Issue: 4, April 2001

10. "Square-extensional mode single-crystal silicon micromechanical resonator for low-phase-noise oscillator applications" Kaajakari, V.; Mattila, T.; Oja, A.; Kiihamaki, J.; Seppa, H.; IEEE Electron Device Letters, Volume: 25, Issue: 4, April 2004

11. "Phase noise characterization of SAW oscillators based on a Newton minimization procedure" Klemer, D.P.; Shih, K.-M.; Clark, E.E., III; IEEE Transactions on Microwave Theory and Techniques, Volume: 39, Issue: 5, May 1991

12. "Precision surface-acoustic-wave (SAW) oscillators" Parker, T.E.; Montress, G.K.; IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control, Volume: 35, Issue: 3, May 1988

13. "Analysis of phase noise due to bang-bang phase detector in PLL-based clock and data recovery circuits" Vichienchom, K.; Wentai Liu; Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on Circuits and Systems, Volume: 1, 25-28 May 2003

14. "Analysis of a half-rate bang-bang phase-locked-loop" Ramezani, M.; Andre, C.; Salama, T.; Circuits and Systems II: IEEE Transactions on Analog and Digital Signal Processing [see also IEEE Transactions on Circuits and Systems II: Express Briefs], Volume: 49, Issue: 7, July 2002

14. "CVPD-034 Differential Voltage Controlled Oscillator Datasheet" Crystek Crystals Corportation

15. "AD5320 Digital to Analog Converter Datasheet" Analog Devices, Inc.

16. "Charge-Pump Phase-Lock Loops" Gardner, F.; IEEE Transactions on Communications, Volume: 28, Issue: 11, Nov 1980

17. "Frequency limitations of a conventional phase-frequency detector" Soyuer, M.; Meyer, R.G.; IEEE Journal of Solid-State Circuits, Volume: 25, Issue: 4, Aug. 1990

18. "Clock and data recovery circuit for 2.5Gbps Gigabit Ethernet transceiver" Shuguiang Li; Junyan Ren; Lianxing Yang; Fan Ye; Zhang, Y.M.M.; ASIC, 23-25 Oct. 2001