uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Michal Debski
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Self-Calibrating Floating-Point Analog-to-Digital Converter

TITRE DE LA THÈSE / TITLE OF THESIS

V. Groza
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

T.A. Kwasniewski

T. Yeap

Gary W. Slater
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

Self-Calibrating Floating-Point Analog-to-Digital Converter

Author: Michal Debski

Advisor: Voicu Groza

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of

Master of Applied Science in Electrical Engineering

under the auspices of
School of Information Technology and Engineering (SITE)

University of Ottawa
Ottawa, Ontario, Canada
March 2005

Your file Votre référence
ISBN: 0-494-11251-4
Our file Notre référence
ISBN: 0-494-11251-4

Acknowledgements

I would like to thank my supervisor, Dr. Voicu Groza, for his supervision and guidance with this research. His expertise, support and encouragement are very much appreciated.

I would like to express my thanks to Dr. Voicu Groza and Dr. Dan Ionescu for their initial ideas which started this research. I would also like to express my gratitude to my father, Dr. Stan Debski, who has provided countless ideas during the document review process.

Additionally, I would like to show appreciation to Analog Devices and Texas Instruments Inc for providing sample components which were used in building of the prototype described in this thesis.

Finally, I would like to thank my family, for their patience throughout the years.

Abstract

The Floating-Point Analog-to-Digital Converter (FPADC) is an extended version of the Fixed-Point ADC. It is designed to deal with a broader dynamic range of signals while exhibiting a smaller relative quantization error. The traditional implementation of the FPADC is characterized by a high relative precision, but it requires high-precision high-speed components in order to achieve that. The high precision of the high-speed components comes at a greater cost. This constraint limits the availability of FPADCs to high-priced designs.

The thesis addresses a low-speed and a low-cost calibration approach for the FPADC. It presents the architecture, design and implementation platform of a self-calibrating differential predictive FPADC which is characterized by utilizing low-grade components. The precision is maintained at high values by additional hardware that periodically performs calibration cycles. Starting with a review of the field of FPADC the thesis develops the understanding of the Floating Point ADCs. The implementation is then extended to include a high precision low speed calibrating ADC. A complete implementation of the design is carried out and described. Finally, experimental measurements are performed to test the new FPADC and present the acquired results.

Keywords: *Floating-Point Analog-to-Digital Converter, Field Programmable Gate Array, Self Calibration, BIST*

Table of Contents

Glossary of Terms

ADC	Analog to Digital Converter
BIST	Built-In Self Test
CCD	Charge Coupled Device
DAC	Digital to Analog Converter
DFT	Discrete Fourier Transform
DNL	Differential Non-Linearity
DUT	Device Under Test
DPFPADC	Differential Predictive FPADC
FP	Floating Point
FPADC	Floating Point Analog to Digital Converter
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HW	Hardware
IC	Integrated Circuit
I/O, IO	Input Output
INL	Integral Non-Linearity
LS	Least Squares
LSB	Least Significant Bit
LUT	Lookup Table
MSB	Most Significant Bit
MUX	Multiplexer
Opamp	Operational Amplifier
OS	Operating System
PC	Personal Computer
P-P	Peak-to-Peak

PSDF	Power Spectral Distribution Function
SC-DPFPADC	Self-Calibrating DPFPADC
SC-FPADC	Self-Calibrating FPADC
SFDR	Spurious-Free Dynamic Range
SNR	Signal to Noise Ratio
SW	Software
THD	Total Harmonic Distortion
TSD	Total Spurious Distortion
VGA	Variable Gain Amplifier

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Goals of the Thesis

Linear Analog to Digital Converters are characterized by a constant quantization step. They are very well-understood and have an extensive market penetration. They also enjoy a wide-spread interest in the research community. The Floating Point Analog-to-Digital Converters employ a variable quantization step in order to maximize their Signal-to-Noise Ratio. Unlike the field of Linear ADCs, the area of Floating Point Analog-to-Digital Converters is relatively new and uncharted. This relative lack of research and corresponding market presence presents a wealth of possibilities for new research and is the motivation for this thesis.

The overall goal of this thesis is to research and present a novel design of a high-precision Low-Cost Self-Calibrating Floating Point Analog-to-Digital Converter. To achieve this goal, the concept of Self-Calibration is applied to a standard Floating-Point Analog-to-Digital Converter. The concept is then applied to existing hardware and software design techniques in order to create a complete system design. Subsequently, the design is implemented using real-world hardware and software. Finally, an evaluation is carried out in order to evaluate the effectiveness of the design.

1.2 Thesis Contribution

The thesis makes a contribution to the field of FPADCs through performing a proof-of-concept study. It proposes a novel and improved Self-Calibrating FPADC concept which uses low-cost components for the mixed-signal circuitry section. The proposed calibration method measures the error values of the system, and compensates the quantized signal using the calibration values. The thesis carries out a study of the concept and analyzes a possible implementation of the FPADC. Additionally, it proposes a self-calibration methodology for the novel FPADC. The thesis sketches out a design of a reference FPADC architecture utilizing the analysis of the proposed FPADC as the basis. The proposed architecture is the foundation for a real-world implementation which in turn provides a test-bed for analyzing and studying the characteristics of the FPADC. The evaluation of the FPADC demonstrates the feasibility of such a design in the real world.

It is believed that the research presented here will serve as the basis for research of improved future designs of the FPADC.

The dissertation shows that the novel design of Low-Cost, Self-Calibrating Floating Point ADC is viable and provides an increased overall accuracy when compared to a Linear ADC and an uncalibrated FPADC. A number of possible improvements and shortcomings are discovered and presented in the thesis as the conclusion of the research performed here.

1.3 Organization of the Thesis

The thesis is organized into six chapters plus appendices that describe the concepts related to specific area of the work as described below.

Chapter one presents an introduction of the key concepts of Floating-Point Analog-to-Digital converters and Self-Calibration aspects. The chapter additionally offers an introduction to the structure of the thesis and the papers published during the thesis work.

Chapter two of the thesis presents the proposed Self-Calibrating FPADC. First, the high-level description of the new FPADC is presented. It is followed by a study of the parameters governing the new system in order to establish common terms and variables by which the system is later described.

Chapter three describes the design details of the FPADC. More specifically, the chapter introduces the proposed FPADC algorithms which are then used in the implementation stage.

Chapter four includes the details of the implementation of the FPADC. A hardware platform used as a basis for the implementation is described along with the additional components needed. The chapter describes the internal architectures of the FPGA and the Software Control Application used in the FPADC design. The chapter also presents the testing and break-in procedures derived during the implementation process. It aims to inform the reader about the obstacles that were overcome during the prototyping stage.

Chapter five discusses a measure of the accuracy of the new FPADC. Third-party (external) ADC evaluation software is used to compute the characteristics of the newly developed ADC based on captured sample data. The results are then compared to those obtained from a standard linear ADC. The chapter also provides a conclusion of the results gathered during the thesis work.

Chapter six gives a summary of the thesis, discusses discovered problems, and presents the future work to be performed in order to improve the FPADC.

The appendices contain the information on the following topics:

- Schematic and Bill of Materials for the mixed-signal add-on card components of the FPADC
- Source code of MATLAB programs used in FPADC testing
- Source code of the FPADC FPGA logic core implementation
- Source code of the FPADC software control application

1.4 Published Papers

Parts of this dissertation have already been subject of the following scientific papers:

- "*Design and Implementation of a Self-Calibrating Floating Point Analog-to-Digital Converter*" by M. Debski, V. Groza, D. Ionescu [22]
- "*Self-Calibrating Differential Predictive Floating Point Analog-to-Digital Converter*" by M. Debski, V. Groza [23]

The works listed have been the foundation of the work included in this thesis. Publication [22] describes the design and implementation of the original Self-Calibrating FPADC which has been ceased due to a hardware problem unrelated to the FPADC. It is the foundation of the concept of the Self-Calibrating FPADC, and serves as the basis for chapters one and two. Publication [23] describes the current design and implementation of the improved Self-Calibrating FPADC which was developed and implemented on another hardware platform. It is the basis for chapters one through four.

1.5 Background Information

1.5.1 Floating-Point Analog-to-Digital Converter

The Floating-Point Analog-to-Digital Converter (FPADC) is an extended version of the Fixed-Point ADC designed to deal with a broader dynamic range of signals at an equivalent resolution. The converter attempts to minimize the signal-to-noise ratio through varying the quantization step depending on the magnitude of the input signal.

Typical uses of the FPADC include applications where wide-range analog signals have to be processed with minimum signal quality loss due to the internal noise of the converter:

- Digital radios where a flexible input quantizer is needed
- Hearing aids
- Wide-range CCD applications

The traditional approach when dealing with sampling of such signals is to use a logarithmic amplifier connected to a Fixed-Point ADC with a digital correction stage [7]. This approach however presents a problem, as the higher the compression ratio of the signal the lower the accuracy.

A standard FPADC conversion approach is to use a two-step quantizing process. First, a coarse-grain linear ADC is used to calculate the coarse range of the signal (exponent). The exponent is used in a variable-gain amplifier [2, 4, 6 and 8], quantizer-subtractor [5] or a programmable-range ADC [3]. The resulting sub-range component is then sampled by the second fine-grain linear ADC, which extracts the remainder (mantissa) of the signal. The combination of the mantissa and the exponent form the floating-point representation of the value of the sample. Figure 1 shows a typical Floating-Point ADC architecture.

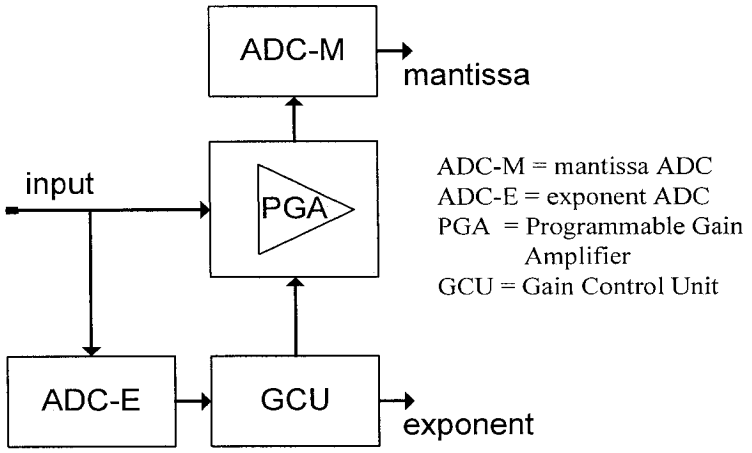


Figure 1 – Typical Subranging Floating Point ADC

In comparison to Fixed-Point ADC, the FPADC provides significant power savings and more relaxed component tolerances. It also allows for a direct floating-point format digital readout without the use of correction table [1]. An added benefit to the large dynamic range is the fact the SNR is less-dependant on the level of the input signal. Figure 2 presents a typical quantization curve of a Floating-Point ADC.

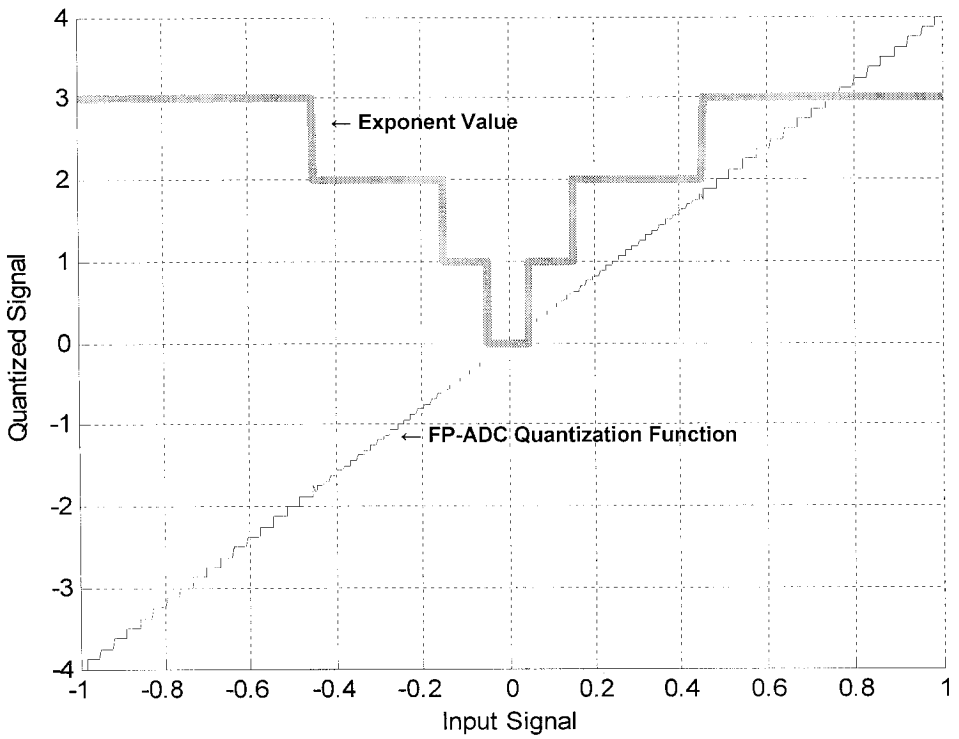


Figure 2 – Quantization properties of an FPADC

An example of an FPADC implementation has been proposed in publications [1], [2] and [8], presented in Figure 3. In the proposed FPADC an amplifier network is implemented which provides a number outputs consisting of the input signal amplified at different values. Additionally, each output is independently sampled through a sample-and-hold circuit. The logic circuit in the control block selects the signal with the proper gain in order to minimize the SNR.

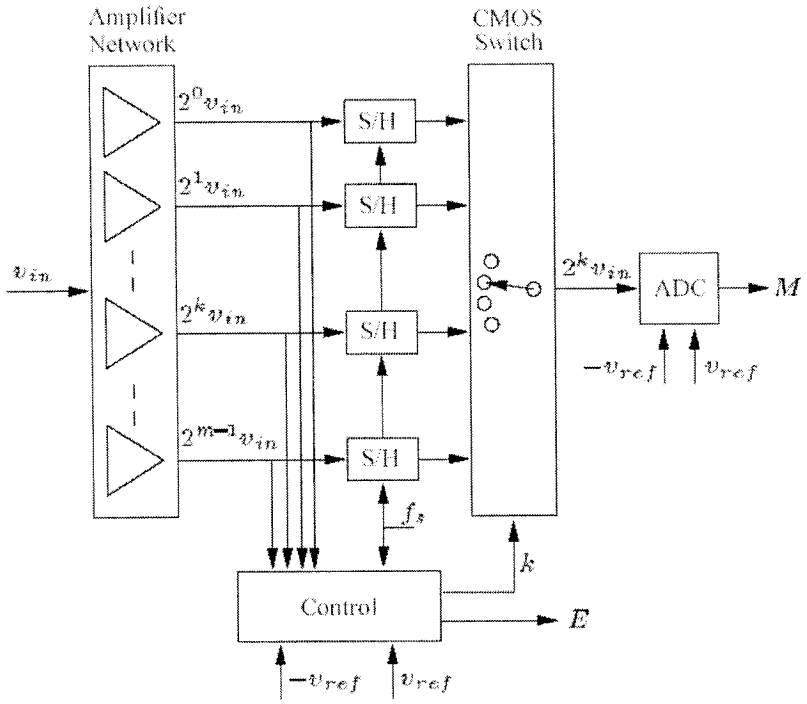


Figure 3 – FPADC Described in Publication [8]

This particular FPADC is identified to suffer from the following four sources of errors, described in Figure 4:

- Gain mismatch due to non-exponential gain factors of amplifiers
- Delay mismatch resulting from different signal delays in amplifiers working at different gains
- Amplifier offset from the different amplifiers
- Saturation due to slow recovery of amplifiers

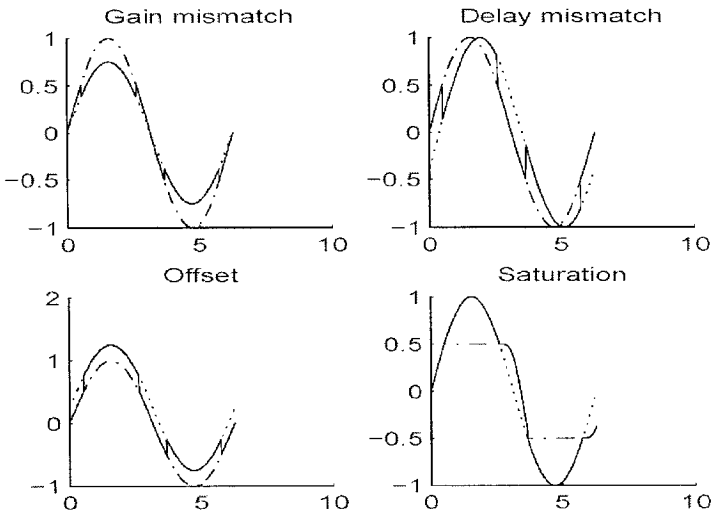


Figure 4 – FPADC Errors Identified in Publication [8]

Another novel high-speed design of the FPADC has been described in publication [19] which uses a parallel sampling architecture. In such a design, the remainder and range ADCs operate in parallel. If the signal falls within the operating range of the remainder ADC, a successful floating-point sample is obtained. If, due to the characteristics of the input signal the difference value on the input of REMAINDER ADC is out of the ADC's sampling range, the gain is readjusted and the sampling process is re-attempted.

1.5.2 Differential Predictive FPADC

The DPFPADC aims to overcome the non-ideal sampling rate restrictions of two-step FPADC. The solution is to add a predictor before the gain amplifier-subtractor. The predictor employs a form of extrapolation [10 and 11] to derive the anticipated sample value. This way a complete sample can be obtained in one measurement cycle if the correct exponent value is predicted. In the case of a signal that is out of the predicted bound the system requires two measurement cycles. According to the standard two-step FPADC described in [2, 4, 5, 6 and 8], a second cycle is then required to adjust the gain amplifier-subtractor to the correct range. A number of various prediction algorithms can be implemented which directly affect the accuracy of the converter. The ideal algorithm should exploit as many as possible of the expected statistical properties of the measured signal. In the case of solution proposed in [11] the polynomial regressive extrapolation was used for its speed and simplicity of implementation. The architecture is presented in Figure 5.

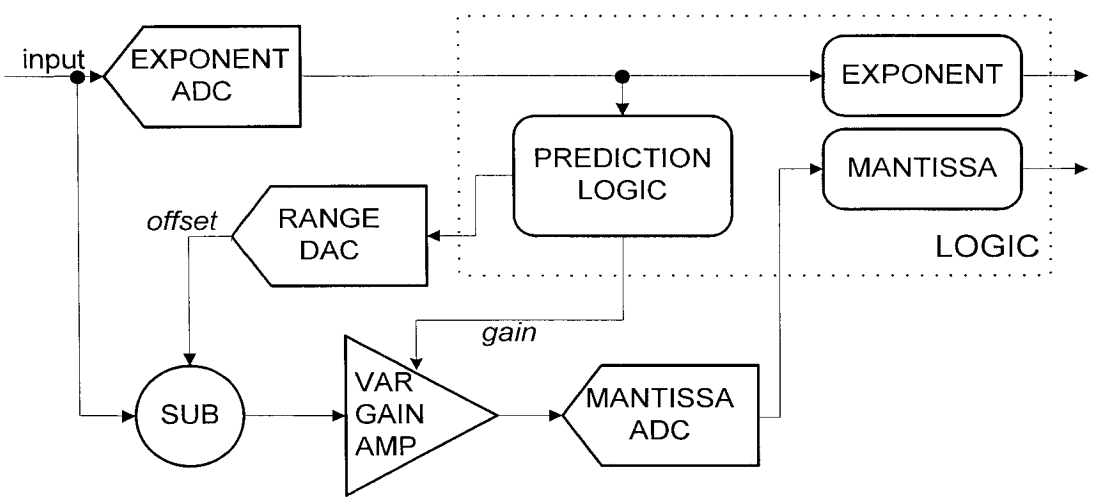


Figure 5 – Differential Predictive FPADC

1.5.3 Calibrating ADC

Due to the inherent complex analog design present in the Floating-Point ADCs there is a need for high-precision BIST in order to correct the errors introduced by the high number

of analog components. The sources of ADC and DAC errors and available correction techniques have been described in detail in [24]. The major sources of static errors are:

- Amplifier offsets
- Gain errors
- Component mismatches due to thermal drift of components
- Non-linearities
- Timing jitter

This thesis mainly addresses the first two sources of error.

A number of approaches for on-chip self-calibration of DACs and classic and subranging ADCs have been devised. One such approach, described in [20, 21], proposes an on-line (non-intrusive) BIST technique which is based on statistical blind testing method in which the input signal serves the purpose of the test signal. Another approach, described in [21, 22 and 23] is to perform the calibration of the system in an off-line fashion. A well-known signal is produced by a high-resolution testing DAC and the discrepancies are measured by a high-resolution ADC. The discovered discrepancies are then used in correcting the output of the system to produce more accurate values. A traditional way of dealing with digital correction is to use a one-dimensional or a two-dimensional lookup table [24]. The table produces the corrected sample data from the uncorrected input sample values.

2 Proposed Approach

2.1 Self-Calibrating Floating Point ADC

The proposed approach is to design an enhanced FPADC based on a self-calibrating differential FPADC [10, 22 and 23] along with a development test platform which allows for evaluation of the design. The enhancements minimize the inaccuracies caused by FPADC's internal components (described in Figure 6) and allow for quick testing and prototyping of different algorithmic approaches.

More specifically, a new functional block is implemented which derives the static transfer characteristics of the amplifiers used in the quantizer section of the FPADC ('Calibration' and 'Correction' logic in Figure 6). It then uses the gathered error values to correct the captured output of the FPADC. The system is easy to prototype due to partial implementation in software (using C/C++ on a Personal Computer), as opposed to an all-hardware implementation.

To address the self-calibration aspect, the new functional block includes a precise integrating ADC ('Calibration ADC' in Figure 6) operating at a much slower speed than the components used during the floating-point quantization. The new ADC periodically executes a calibration cycle which measures the exact static transfer characteristics of the VGA and the associated components. The detected discrepancies are then used for digital correction in the correction logic block. During the normal operation cycle of the proposed FPADC the sampled values are adjusted using the correction data gathered during the calibration phase.

Figure 6 illustrates the major building blocks of the Self-Calibrating DPFPADC (detailed descriptions of the signals and the components are in Section 2.2).

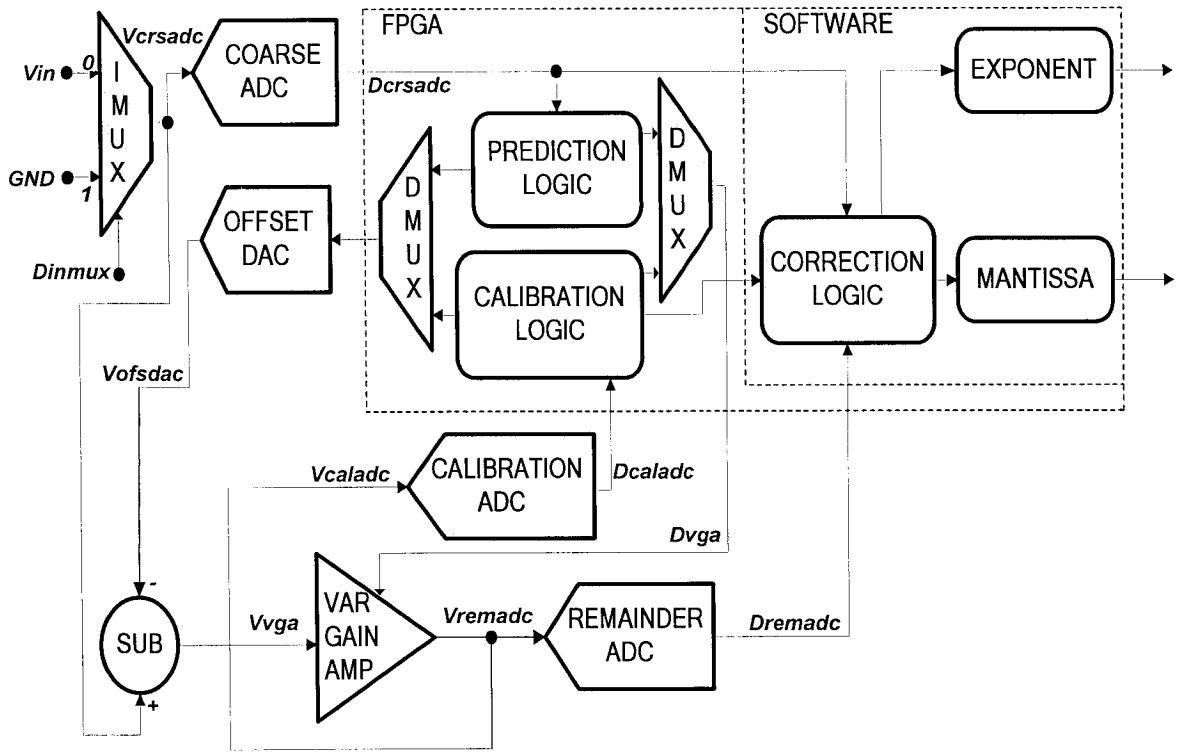


Figure 6 – High-level Block Diagram of the Self-Calibrating Predictive Floating-Point ADC

(Input mux to FPGA/software connections not shown for clarity)

The CALIBRATION ADC has the ability to precisely sample signals at the output of the VGA. Through disabling the input of the system and varying the OFFSET DAC voltage it is possible to discover the static transfer characteristics of the VGA using the CALIBRATION ADC.

The digital logic processing in the system takes place on both an FPGA as well as a Personal Computer running a control software program.

2.1.1 Operation Modes

The FPADC system supports following two operational modes:

- QUANTIZING mode
- CALIBRATION mode

In the QUANTIZING mode the FPADC quantizes a predetermined number of floating point samples by performing a timed acquisition of raw subcomponent values (REMAINDER ADC *Dremadc*, COARSE ADC *Dcrsadc*, OFFSET DAC *Dofsdac* and VGA gain index *Dvga*) and storing them in a memory *cache* on the FPGA. The control program executing on the Personal Computer then downloads the cached raw values, adjusts them using a previously gathered calibration data, and presents the final sample values in a floating-point format. The components that take part in the QUANTIZING cycle are:

- FPGA/Software Software Application
- Coarse ADC
- Offset DAC
- Remainder ADC
- Subtractor
- Variable Gain Amplifier
- Input Mux

The CALIBRATION mode is active when the control logic performs a number of self-tests on the mixed-signal portion of the system in order to determine the static error values governing the FPADC system. The tests are mainly executed by the control program with the FPGA acting as an I/O interface for the mixed-signal components. The components that take part in the CALIBRATION cycle are as follows:

- FPGA/Software Control Application
- Calibration ADC
- Offset DAC
- Subtractor
- Variable Gain Amplifier
- Input Mux

2.1.2 System Constraints

The typical voltage swing of the measured signals is assumed to be limited to 2V peak-to-peak in order to lie within the linear portions of the ADC's and the DAC's

characteristics. It is assumed that the CALIBRATION ADC has the highest sample resolution ranging from 20 to 24 bits. The COARSE and REMAINDER ADCs are assumed to be of fairly low resolution - typically 4 bits. The OFFSET DAC is assumed to have the resolution between that of the CALIBRATION ADC and the COARSE and REMAINDER ADCs, typically ranging from 8 to 14 bits. Another assumption is that the CALIBRATION ADC supports a relatively low sampling frequency in the vicinity of $1*10^2$ samples/sec and the OFFSET DAC and COARSE and REMAINDER ADCs support much higher frequencies such as $8*10^7$ samples/sec. The VGA is assumed to support four different levels of gain.

In addition, the following deviations from a typical DPFPADC are introduced:

- The Prediction Logic block is implemented so that the predicted next sample value is the last obtained sample value. In other words, no sample extrapolation takes place.
- The REMAINDER ADC out-of-range checking and missing sample estimation is not performed.

2.2 System Definitions

This section establishes a common set of variables and relationships governing the proposed system in Figure 6. The defined variable names and the system equations are then referenced throughout the rest of the document.

2.2.1 Analog Voltages

The analog voltage values are represented by the following variables:

- *Vin*: Input voltage
- *Vcrsadc*: Voltage at the input of the COARSE ADC
- *Vofsdac*: Voltage at the output of the OFFSET DAC
- *Vvga*: Voltage at the input of the VARIABLE GAIN AMPLIFIER
- *Vremadc*: Voltage at the input of the REMAINDER ADC
- *Vcaladc*: Voltage at the input of the CALIBRATION ADC

The voltages are assumed to fall within a bipolar range of -1...+1 volt.

2.2.2 Digital Signals

The digital values corresponding to the analog voltages are represented by the following variables:

- *Dcrsadc*: Digitized value of input voltage *Vcrsadc* at the COARSE ADC
- *Dremadc*: Digitized value of input voltage *Vremadc* at the REMAINDER ADC
- *Dofsdac*: Digital value of output voltage *Dofsdac* at the OFFSET DAC
- *Dcaladc*: Digital value of input voltage *Vcaladc* at the CALIBRATION ADC
- *Dvga*: Gain selection index value at the VARIABLE GAIN AMPLIFIER
- *Dinpmux*: Mux selector value at the Input Mux

2.2.3 System Parameters

The system described in Figure 6 is governed by a set of parameters. The most influential parameters relating to the system components are described below.

COARSE ADC:

- *Dmaxcrsadc*: Maximum allowable absolute value of *Dcrsadc* such that
  $-Dmaxcrsadc <= Dcrsadc < Dmaxcrsadc$
- *crsadc_bits*: Bit-width of the digital sample value obtained from the COARSE ADC such that $Dmaxcrsadc=2^{crsadc\_bits-1}$
- *crsadc_step*: Quantization step of the COARSE ADC which defines the relationship between its digital output and analog input so that
  $Dcrsadc = Vcrsadc / crsadc\_step$

REMAINDER ADC:

- *Dmaxremadc*: Maximum allowable absolute value of *Dremadc*
  $-Dmaxremadc <= Dremadc < Dmaxremadc$
- *remadc_bits*: Bit-width of the digital sample value obtained from the REMAINDER ADC such that $Dmaxremadc=2^{remadc\_bits-1}$
- *remadc_step*: quantization step of the REMAINDER ADC which defines the relationship between its digital output and analog input so that
  $Dremadc = Vremadc / remadc\_step$

OFFSET DAC:

- *Dmaxofsdac*: Maximum allowable absolute value of *Dofsdac*
  - *Dmaxofsdac* <= *Dofsdac* < *Dmaxofsdac*
- *ofsdac_bits*: Bit-width of the digital value sent to the OFFSET DAC such that $Dmaxofsdac=2^{ofsdac\_bits-1}$
- *ofsdac_step*: Quantization step of the OFFSET DAC. It defines the relationship between its digital input and analog output so that *Vofsdac* = *Dofsdac* * *ofsdac_step*

CALIBRATION ADC:

- *Dmaxcaladc*: Maximum allowable absolute value of *Dcaladc*
  - *Dmaxcaladc* <= *Dcaladc* < *Dmaxcaladc*
- *caladc_bits*: Bit-width of the digital sample value obtained from the CALIBRATION ADC such that $Dmaxcaladc=2^{caladc\_bits-1}$
- *caladc_step*: Quantization step of the CALIBRATION ADC. It defines the relationship between its digital output and analog input such that
  Dcaladc = *Vcaladc* / *caladc_step*

VARIABLE GAIN AMPLIFIER:

- $vgagain_n$: Voltage gains of the VARIABLE GAIN AMPLIFIER for a given VGA gain index n = 0 … *num_gains-1*
- $\varepsilon Vvgaoffset_n$: Output offset voltage introduced by the VARIABLE GAIN AMPLIFIER for a given gain index n = 0 … *num_gains-1*
- *num_gains*: number of possible gain selections in the VARIABLE GAIN AMPLIFIER, also referred to as the number of CONVERSION DOMAINS

SUBTRACTOR:

- $\varepsilon Vsuboffset$: Offset output voltage introduced by the SUBTRACTOR

INPUT MUX:

- $\varepsilon Vmuxoffset$: Offset output voltage introduced by the analog mux

2.2.4 System Equations

In this section the various relationships between the various voltages and digital values are derived. The relationships are needed to understand the mapping the values of the FPADC components in relation to other values. The relationships are used in calculation of the sample values as well as calibration of the system.

The voltage relationships of the signals present in the system are as follows:

COARSE ADC:

$$Vcrsadc = \begin{cases} Vin + \varepsilon Vmuxoffset & \text{when Dinpmux is 0} \\ \varepsilon Vmuxoffset & \text{when Dinpmux is 1} \end{cases} \quad (1)$$

OFFSET DAC:

$$Vvga = Vcrsadc - Vofsdac + \varepsilon Vsuboffset \quad (2)$$

REMAINDER ADC:

$$Vremadc = Vvga * vgagain_n + \varepsilon Vvgaoffset_n \quad (3)$$

CALIBRATION ADC:

$$Vcaladc = Vremadc \quad (4)$$

In the digital-to-analog conversion the following conversion equations are used:

$$Dofsdac = Vofsdac / ofsdac\_step,\ Vofsdac = Dofsdac * ofsdac\_step,$$
$$Dcaladc = Vcaladc / caladc\_step,\ Vcaladc = Dcaladc * caladc\_step$$
$$Dcrsadc = Vcrsadc / crsadc\_step,\ Vcrsadc = Dcrsadc * crsadc\_step$$
$$Dremadc = Vremadc / remadc\_step,\ Vremadc = Dremadc * remadc\_step$$

The equations (1), (2), (3) and (4) can be written as

$$Dcrsadc = \begin{Bmatrix} Vin + \varepsilon Vmuxoffset & \text{when Dinpmux is 0} \\ \varepsilon Vmuxoffset & \text{when Dinpmux is 1} \end{Bmatrix} / crsadc\_step$$

$$Vvga = Vcrsadc - Dofsdac * ofsdac\_step + \varepsilon Vsuboffset$$

$$Dremadc = (Vvga * vgagain_n + \varepsilon Vvgaoffset_n) / remadc\_step$$

$$Dcaladc * caladc\_step = Dremadc * remadc\_factor + \varepsilon Vmuxoffset$$

The CALIBRATION ADC allows for measuring of the offset and the gain of the VGA at different gain levels. It is also used in establishing the conversion factor between the REMAINDER ADC and the OFFSET DAC, i.e., the ratio between their quantization steps with the precision allowed by the CALIBRATION ADC.

The conversion factors are derived in the following way:

From equation (3) and (4)

$$Vcaladc = Vvga * vgagain_n + \varepsilon Vvgaoffset_n$$

$$Vcaladc = Vremadc$$

when

$$Vvga = (Vremadc - \varepsilon Vvgaoffset_n) / vgagain_n$$

and

$$Vvga = \varepsilon Vmuxoffset - Vofsdac + \varepsilon Vsuboffset$$

so

$$(Vremadc - \varepsilon Vvgaoffset_n) / vgagain_n = \varepsilon Vmuxoffset - Vofsdac + \varepsilon Vsuboffset$$

$$Vcaladc = Vremadc = (\varepsilon Vmuxoffset - Vofsdac + \varepsilon Vsuboffset) * vgagain_n + \varepsilon Vvgaoffset_n$$

And since

$$Vofsdac = Dofsdac * ofsdac\_step$$

$$Vcaldac = Dcaldac * caldac\_step$$

and $Vremdac = Dremdac * remdac\_step$

the above equations can be combined to obtain:

$$Dcaladc = ((\varepsilon Vmuxoffset - Vofsdac + \varepsilon Vsuboffset) * vgagain_n + \varepsilon Vvgaoffset_n) / caladc\_step$$

$$Dremadc = ((\varepsilon Vmuxoffset - Vofsdac + \varepsilon Vsuboffset) * vgagain_n + \varepsilon Vvgaoffset_n) / remadc\_step$$

Assuming $\varepsilon Vmuxoffset$ and $\varepsilon Vsuboffset$ are zero, the following relationship between *Dcaladc* and *Dofsdac*, *Dcrsadc* and *Dofsdac* and *Dcrsadc* and *Dcaladc* ($\varepsilon Vmuxoffset$ and $\varepsilon Vsuboffset = 0$) can be developed:

$$Dcaladc = ((- Dofsdac * ofsdac\_step) * vgagain_n + \varepsilon Vvgaoffset_n) / caladc\_step$$
$$Dremadc = ((- Dofsdac * ofsdac\_step) * vgagain_n + \varepsilon Vvgaoffset_n) / remadc\_step$$

$$Dremadc * remadc\_step = Dcaladc * caladc\_step$$
$$Dremadc = Dcaladc * (caladc\_step / remadc\_step)$$
$$Dcaladc = Dremadc * (remadc\_step / caladc\_step)$$

The above equations allow us to map values of the OFFSET DAC in terms of units of the REMAINDER and CALIBRATION ADCs. This property is useful for factoring the VGA offsets measured by the CALIBRATION ADC into the final sample calculation and during building of the lookup tables used during quantization.

The above equations also allow us to map values of the REMAINDER ADC in terms of units of the CALIBRATION ADC. This property is useful for factoring the input offset measured by the CALIBRATION ADC into the final sample calculation.

2.3 System Operation

The FPADC system performs three main tasks:

- Quantization of the input signal
- Digital correction of the quantized signal
- Self-Calibration of the internal FPADC components

The tasks are described in detail in this section.

2.3.1 Quantization

A QUANTIZING cycle starts with the acquisition of a coarse value (*Dcrsadc*) that is obtained by directly sampling the analog signal (*Vin*) through the COARSE ADC. This ADC is an *m*-bit uniform midtread quantizer that employs the following rounding quantization for an input range of $Vin \in [-D, D)$:

$$Dcrsadc = \begin{cases} -(2^{m-1}) \cdot \dfrac{\Delta}{2} & \text{for } Vin < -(2^{m-1}) \cdot \dfrac{\Delta}{2} \\ \left\lfloor \dfrac{Vin + \frac{\Delta}{2}}{\Delta} \right\rfloor \cdot \Delta & \text{for } Vin \in \left[-(2^{m-1}) \cdot \dfrac{\Delta}{2}, (2^{m-1} - 1) \cdot \dfrac{\Delta}{2} \right] \\ (2^{m-1} - 1) \cdot \dfrac{\Delta}{2} & \text{for } Vin > (2^{m-1} - 1) \cdot \dfrac{\Delta}{2} \end{cases} \quad (1)$$

where the quantization step is $\Delta = \dfrac{D}{2^{m-1}}$

and $\lfloor a \rfloor$ represents the greatest integer less than or equal to a.

In the case of a two-step FPADC, *Dcrsadc* is used to calculate the exponent *e*. To express the exponent as an *E*-bit number, the COARSE ADC has to have a resolution $m \geq 2^E$, with *m* and *E* natural numbers ($m \in \mathbf{N}$, $E \in \mathbf{N}$). Powers of 2 are chosen for *m* (i.e., $m = 2^E$) to take full advantage of the *m*-bit resolution in expressing the exponent values. The exponent *e* is employed to do a partition of the input range $[-D, D)$ in conversion domains $\{D_e, e \in [0, 2^E\text{-}1) \cap \mathbf{N}\}$:

$$e=\begin{cases} 0 & \text{if } V_{\text{in}} \in D_0=\left[-\frac{\Delta}{2},\frac{\Delta}{2}\right) \\ \left\lfloor \log_2\left|\frac{\text{Dcrsadc}}{\Delta}\right|\right\rfloor+1 & \text{if } V_{\text{in}} \in D_e=\left[-(2^{e+1}-1)\frac{\Delta}{2},-(2^e-1)\frac{\Delta}{2}\right)\cup\left[(2^e-1)\frac{\Delta}{2},(2^{e+1}-1)\frac{\Delta}{2}\right) \\ 2^E-1 & \text{if } V_{\text{in}} \in D_{2^E-1}=\left(-\infty,-(2^{2^E-1}-1)\frac{\Delta}{2}\right)\cup\left[(2^{2^E-1}-1)\frac{\Delta}{2},\infty\right) \end{cases} \quad (2)$$

In the case of a parallel one-step FPADC, this coarse value (*Dcrsadc*) is used to determine a prediction of the next-cycle sample (*Dcrsadc*). Equation (2) is employed as well to find the exponent e, but with *Dcrsadc* substituted by the predicted value *Dcrsadc*.

If implementing a standard FPADC [19], the REMAINDER ADC digitizes the input sample *Vin* after being amplified in the variable gain amplifier VGA, such that |*Vremadc*| = |*Vin*| * *gain* ∈ (*D*/2, *D*). In this case, DAC OFFSET is employed along with the analog-subtractor SUB only to compensate for the errors introduced by the components for the current conversion domain.

When implementing a differential FPADC [11], the predicted value is converted into an appropriate OFFSET DAC value through the use of an *offset_lut*[] look-up table. The following two approaches can be developed based on the way the OFFSET DAC value (*Vofsdac*) is calculated:

$$Vofsdac=\begin{cases} Vofsdac_1 = D_e/2 = \text{midpoint of } D_e \\ Vofsdac_2 = Dcrsadc \end{cases} \quad (3)$$

The first approach selects a fixed OFFSET DAC voltage for each conversion domain. This ensures that the resulting REMAINDER ADC voltage falls within the allowable ADC ranges.

The second approach selects a variable OFFSET DAC voltage so that the REMAINDER ADC voltage is as close to zero as possible.

In both cases, on the analogue side of things, the output from the OFFSET DAC is subtracted from the value of the input sample (*Vin*), difference which represents the 'fine' information in the analogue signal that would normally be lost in a non-floating point ADC with a resolution like COARSE ADC's. The resulting analogue signal (*Vvga* = *Vin* – *Vofsdac*) is then passed onto the Variable-Gain Amplifier (VGA), whose gain is set accordingly to the value of the predictor, as dictated by the values stored in the *gain_lut*[] lookup table:

$$gain = \begin{cases} gain_1 = 2^{2^E - e - 1} & \text{since in the first case } Vremadc \in D_e \\ gain_2 = 2^m & \text{since in the 2nd case } Vremadc < \text{ADC COARSE} \end{cases}$$

The resulting amplified analog signal *Vremadc* = *gain* * *Vvga* = *gain* * (*Vin* – *Vofsdac*) is then quantized by the REMAINDER ADC.

2.3.1.1 Conversion Domains

The COARSE ADC value acquired by the FPADC performs a partition of the range of the input signal (*Vin*) in different conversion domains. Each such *domain* is characterized by a *DAC offset* (to be subtracted from the input signal; described in the *domain_offset*[] table) and a *gain_value* that changes from one *conversion domain* to another at the threshold points defined by the *domain_threshold[]*.

The typical relationship between the input voltage *Vin* and the conversion domains D_e along with their corresponding gains is described in Figure 7.

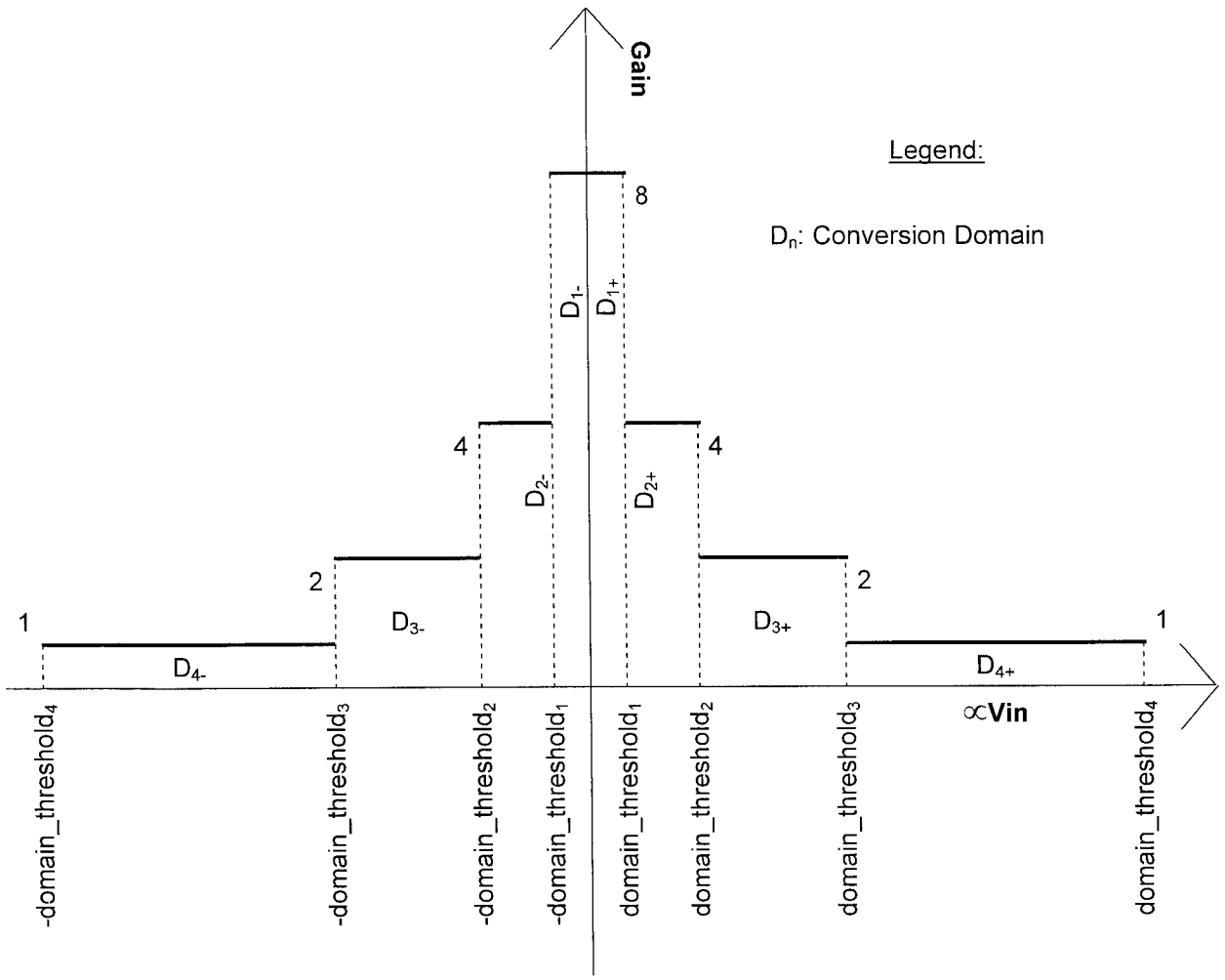


Figure 7 – Conversion Domain Switching Points

2.3.1.2 Conversion Domain Offsets

Each conversion domain (based on input voltage value *Vin*) requires the input voltage to be DC shifted in order to fall within the REMAINDER ADC sampling voltage range. The relationships between the input voltage *Vin* (=*Vcrsadc*) and the required DC base levels are described in Figure 8.

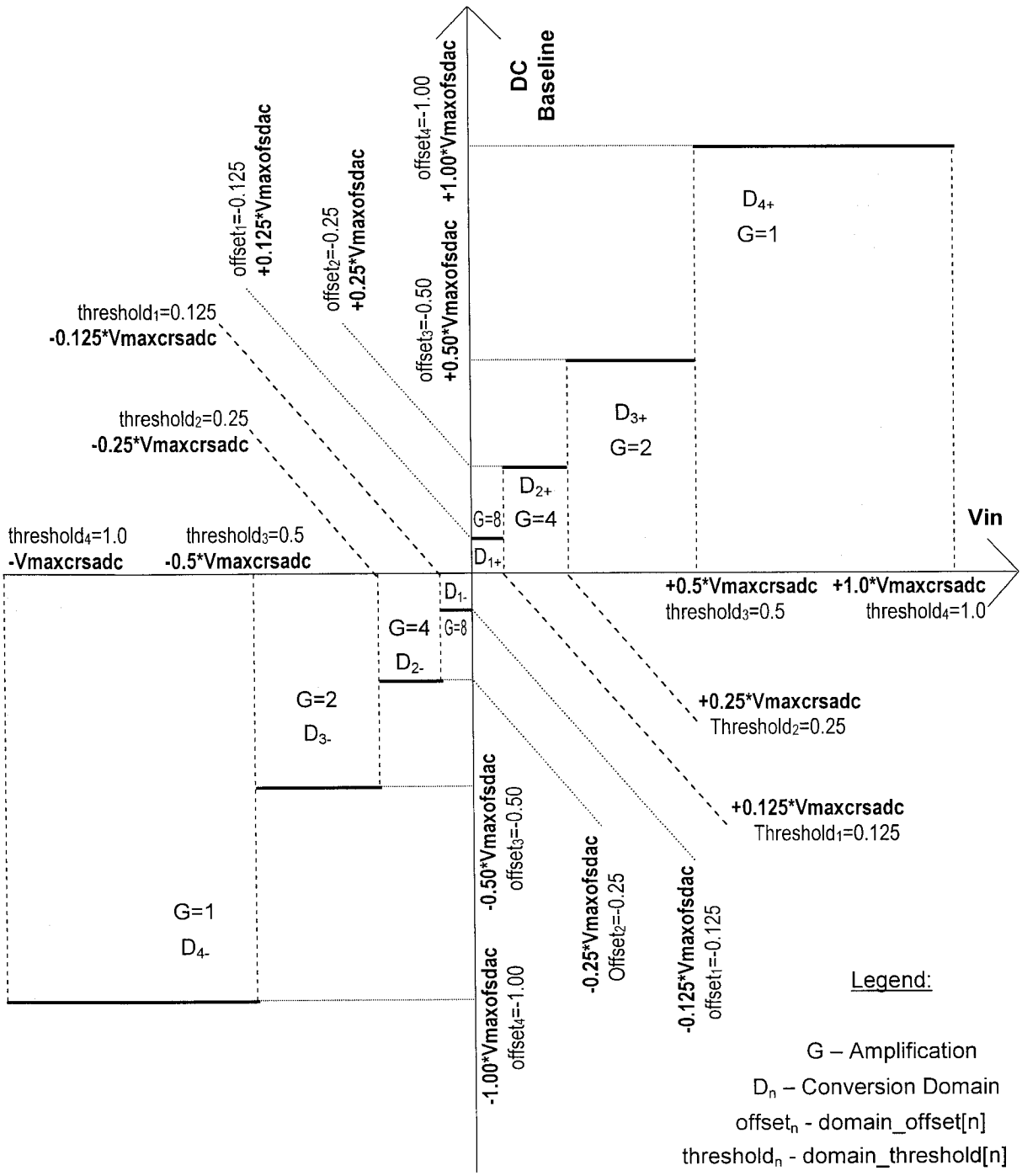


Figure 8 – Relationship of Input Voltage and the DC Offset Shift

2.3.1.3 Adjusted Conversion Domain Thresholds and Offsets

Conversion domain switching points based on the value of *Vin* are characterized by the values of the conversion domain threshold table $domain\_threshold_n$. The values of OFFSET DAC corresponding to the different conversion domains are characterized by the values of the conversion domain offset table of $offset_n$.

In reality, the COARSE ADC has a finite number of quantized steps. The ideal conversion domain switching points and offsets described above are based on the non-quantized value of *Vin*. The quantizing characteristics of the ADC have to be taken into account. Figure 9 illustrates the relationship between the COARSE ADC input voltage *Vin* and its quantized value *Dcrsadc* (two highest-gain conversion domains shown) and the corresponding conversion domains.

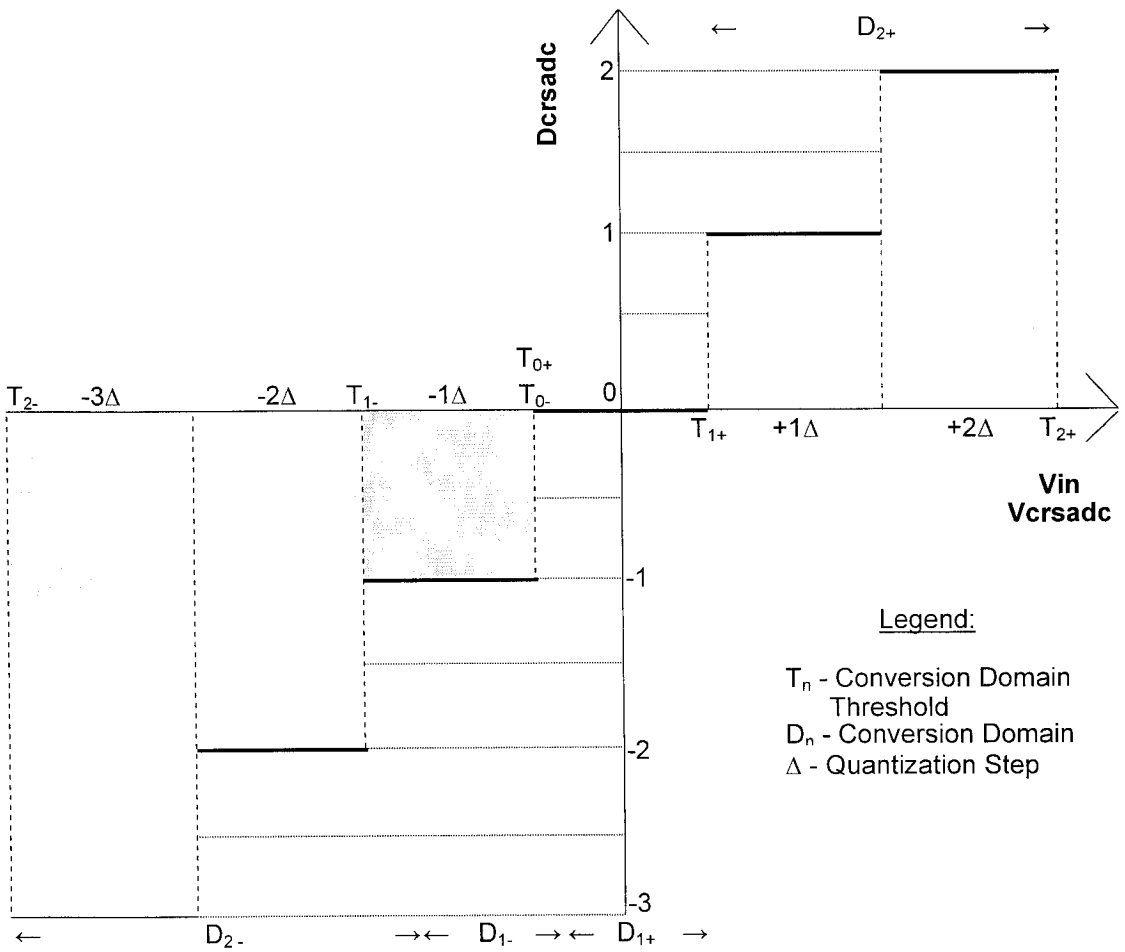


Figure 9 – Detailed Relationship of Conversion Domain, Input Voltage and COARSE ADC Value

After taking into account the switching characteristics of the ADC we observe the following *Vin* ranges for each of the conversion domains:

Positive conversion domain threshold:

$$T_{n+} = \left(-\frac{1}{2} + 2^n - 1 \right) * \Delta_{crsadc}$$

$$\text{where } D_{n+} = \left[T_{(n-1)+}, T_{n+} \right)$$

Negative conversion domain threshold:

$$T_{n-} = \left(-\frac{1}{2} - 2^n + 1 \right) * \Delta_{crsadc}$$

$$\text{where } D_{n-} = \left[T_{n-}, T_{(n-1)-} \right)$$

From the above equations we can see that positive and negative conversion domain *thresholds* are equivalent (mirrors of each other) except for being shifted towards the negative values by $\frac{1}{2} * \Delta_{crsadc}$ to accommodate for the switching properties of the ADCs.
In addition to the conversion domain switching thresholds, each of the conversion domains needs an offset voltage *Vofsdac* in order to shift the selected subrange of the input signal into the (+*Vmaxremadc*, -*Vmaxremadc*) range acceptable for the input of the REMAINDER ADC. The desired OFFSET DAC voltages for each of the conversion domains are shown (two conversion domains shown) in Figure 10.

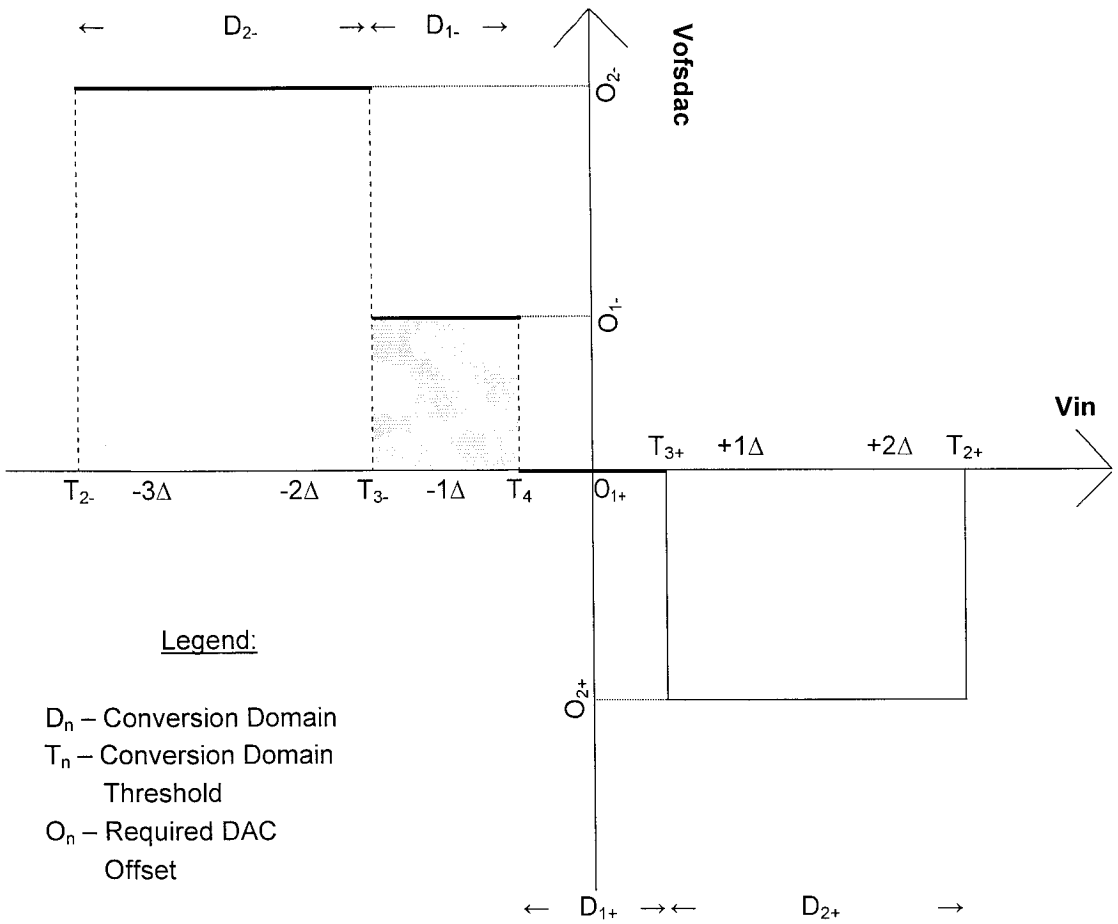


Figure 10 – Detailed Relationship of Conversion Domain, Input Voltage and Offset DAC Voltage

The conversion domain offset voltages can be quantified in the following way:

Positive conversion domain offsets:

$$O_{n+} = -\frac{2^{n-1} + 2^n - 3}{2} * \frac{\Delta_{crsadc}}{\Delta_{ofsdac}}$$

Negative conversion domain offsets:

$$O_{n-} = \frac{2^{n-1} + 2^n - 1}{2} * \frac{\Delta_{crsadc}}{\Delta_{ofsdac}}$$

2.3.2 Calibration

As mentioned before, the CALIBRATION cycle identifies the FPADC error values imposed by the analog portion of the FPADC system through the process of self-testing.

In this particular system the VGA is assumed to introduce the largest error due to the discrete nature of the components used to implement it. The static VGA error components are depicted by Figure 11.

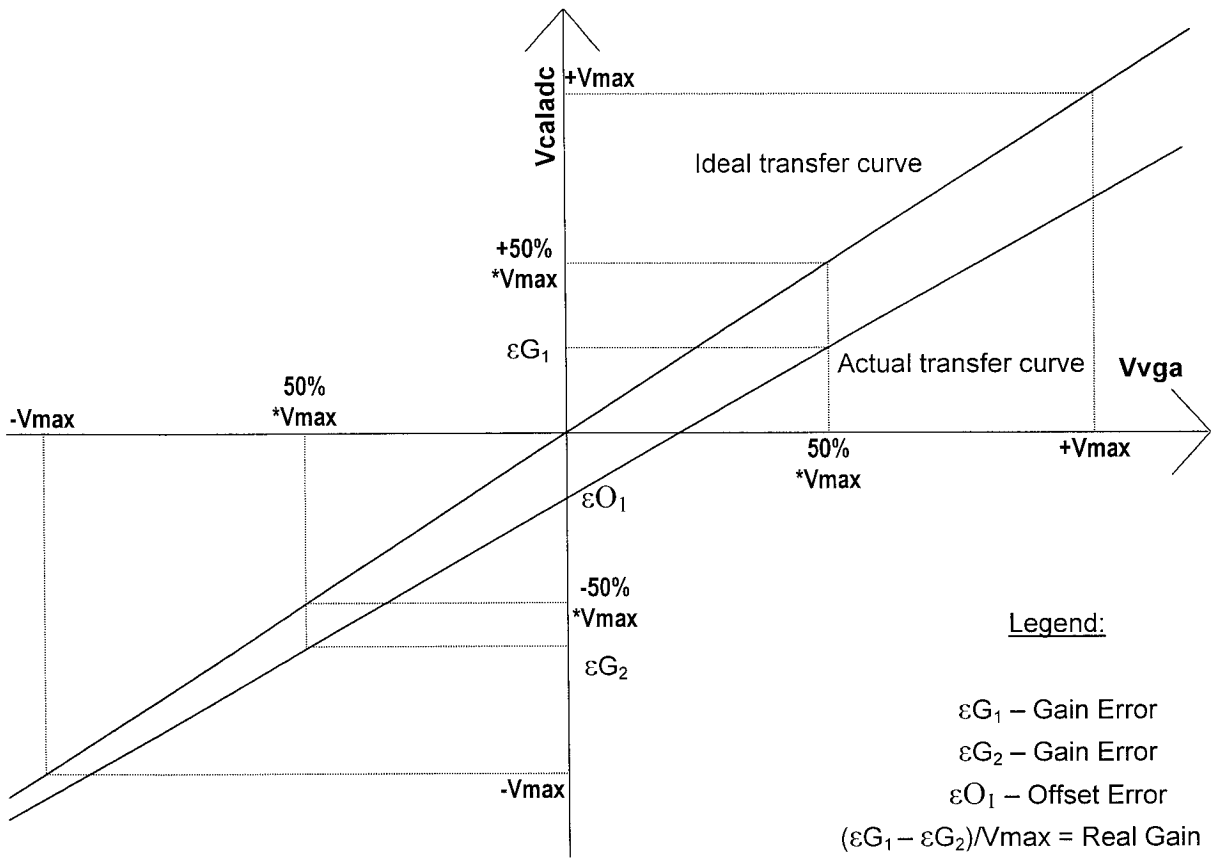


Figure 11 – Error Components in the VGA

The true gain for each of the conversion domains can be calculated by measuring the slope of the gain curve of the CALIBRATION ADC for each of the conversion domains. This is done by performing the following two measurements:

- Measure *Dcaladc* (εG_1) when *Dofsdac* = *+50% * (Dmaxofsdac/*$2^{num\_gains-Dvga}$*)*

- Measure *Dcaladc* (εG_2) when *Dofsdac* = *-50% * (Dmaxofsdac/*$2^{num\_gains-Dvga}$*)*

The true gain is calculated as:

$$\text{vgagain}_{Dvga} = (\varepsilon G_1 - \varepsilon G_2)\frac{2^{num\_gains-Dvga}}{Dmaxofsdac}$$

Offset voltage of the VGA can be calculated by using the CALIBRATION ADC to measure the offset voltage εO1 produced when the output of the OFFSET DAC is set at zero.

2.3.3 Correction

Once the QUANTIZING takes place on the FPGA and saves the subcomponent values and into a memory buffer on the FPGA, the software component retrieves the saved COARSE ADC, REMAINDER ADC, OFFSET DAC and VGA Gain values from the cache memory. The software component then composes a final sample value by correcting it using previously obtained calibration values. The final sample value is calculated using the following equations:

$$mantissa = \left(Dremadc * \frac{caladc\_step}{remadc\_step} + Dofsdac * \frac{caladc\_step}{ofsadc\_step} * vgagain_{Dvga} - \frac{\varepsilon Offset_{Dvga}}{caladc\_step} \right) * \frac{2^{num\_gains-Dvga}}{vgagain_{Dvga}}$$

$$exponent = Dvga$$

$$final\_sample\_value = mantissa * 2^{exponent}$$

The mantissa returns a value in the range of [-Dmaxcaladc, + Dmaxcaladc].
The exponent is in the range of [0, num_gains).

3 Design

The FPADC described in Figure 6 employs a number of novel algorithms. This chapter describes the proposed algorithmic approaches for the FPADC.

The employed algorithms can be divided into two main groups. The groups are based on what FPADC mode they are used in. They are the following:

- Algorithms used while the system is operating in QUANTIZING mode. This mode is used to digitize the input voltage into a corresponding floating point sample value.
- Algorithms used during the CALIBRATION mode. This mode is used to establish the error values governing the system. The error values are obtained through a number of self-tests.

The above groups and their corresponding algorithms are described in detail in the following sections.

3.1 Quantizing Cycle

The QUANTIZING cycle is executed through two consecutive processes. One, executed on the programmable logic, gathers a predetermined number of components values from the VGA, OFFSET DAC and COARSE and MANTISSA ADCs in a real-time manner, and stores them to the cache memory. Once the memory buffer is filled, the software component running on a personal computer retrieves the saved values. It then applies a correcting algorithm to calculate the final floating point sample values.

More specifically, a coarse value (Dcrsadc) is first obtained by directly quantizing the analog signal (Vin) through the COARSE ADC. This coarse value (Dcrsadc) is then used to lookup an appropriate OFFSET DAC and VGA gain value through the use of the internal FPGA lookup tables *offset_lut*[] and *gain_lut[*. The tables use the value of *Dcrsadc* as the input (index) and output the value of the OFFSET DAC *Dofsdac* and VGA gain *Dvga*.

The system diagram while the QUANTIZING cycle is active is shown in Figure 12.

The subtraction of *Vofsdac* DC offset and amplification of the difference by the VGA signal creates the remainder voltage at the input of the REMAINDER ADC. The resulting voltage represents the 'fine' information in the analogue signal that would normally be lost in a non-floating point ADC with a resolution like COARSE ADC's. Once the REMAINDER ADC value is collected, all the subcomponent values (COARSE ADC (current Vin), OFFSET DAC (predicted Vin), REMAINDER ADC (*Vin - OFFSET DAC = Vin - predicted Vin*), and VGA Conversion Domain) are stored in internal FPGA memory. The stored values are retrieved from the memory and corrected by the second QUANTIZING cycle process, executed in software.

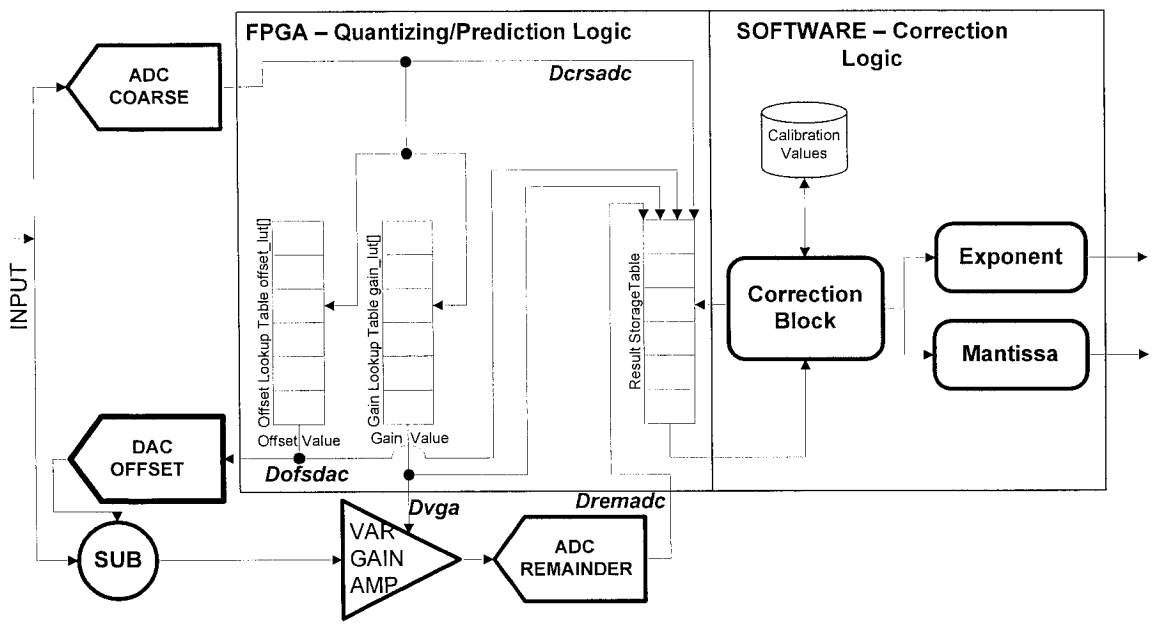


Figure 12 – Quantizing Block Diagram

3.1.1 Lookup Tables

The FPGA lookup tables *offset_lut*[] and *gain_lut[]* are created by the software component from the previously described threshold tables *domain_offset[]* and *domain_threshold[]*.

The lowest index value of the table *domain_threshold[0]* refers to the lowest conversion domain (highest gain) and the highest index value of the table *domain_threshold[numgain-1]* refers to the highest conversion domain (lowest gain). The value of *domain_threshold[]* range from 0 to 1. The value represents the absolute percentage value of the COARSE ADC for which a given conversion domain is being selected.

The *domain_offset[]* table specifies the value *Dofsdac* for a specific gain. The values range from 0 to 1 where 1 implies +100% or -100% of full value of *Dmaxofsdac*, depending on the sign of the COARSE ADC value *Dcrsadc*.

For instance, the FPADC selects the lowest gain and the following OFFSET DAC values when these COARSE ADC values are detected:

- *+100% * Dmaxcrsadc > Dcrsadc > +45% * Dmaxcrsadc*
  Resulting *Dofsdac* = +72.5% * *Dmaxofsdac*
- *-100% * Dmaxcrsadc < Dcrsadc < -45% * Dmaxcrsadc*
  Resulting *Dofsdac* = -72.5% * *Dmaxofsdac*

A sample content of the lookup tables is shown in Table 1 (assuming table size is 2^{12}).

The sample table has 4096 entries (2048 positive and 2048 negative), and supports 4 domain gains. The OFFSET DAC is assumed to have 16 bit output (32768 possible positive values and 32768 possible negative values). The *domain_offset[]* and *domain_threshold[]* tables are assumed to have the following values:

- *domain_threshold[]* table are *{ 0.050, 0.150, 0.450, 1.000 }*.
- *domain_offset[]* table are *{ 0.025, 0.100, 0.300, 0.725 }*.

The algorithm used to fill the values of the tables is described further in the document in Section 3.1.4.

Index (Dcrsadc)	*domain_offset[]*	*domain_gain[]*	*conversion domain*
+2047	-42000	3	↑
...	-42000	3	D_{4+}
+0850	-42000	3	↓
+0849	-18000	2	↑
...	-18000	2	D_{3+}
+0250	-18000	2	↓
+0249	-6000	1	↑
...	-6000	1	D_{2+}
+0050	-6000	1	↓
+0049	0	0	↑
...	0	0	D_{1+}
-0050	0	0	↓
-0051	4000	0	↑
...	4000	0	D_{1-}
-0150	4000	0	↓
-0151	10000	1	↑
...	10000	1	D_{2-}
-0350	10000	1	↓
-0351	22000	2	↑
...	22000	2	D_{3-}
-0950	22000	2	↓
-0951	46000	3	↑
...	46000	3	D_{4-}
-2048	46000	3	↓

Table 1 – Typical *domain_offset[]* and *domain_gain[]* Lookup Table

3.1.2 Quantizing Process

The main quantizing process is executed on the FPGA at the beginning of the QUANTIZING cycle. It calculates the intermediate sample values. Figure 13 presents the state flow diagram of the main quantizing process.

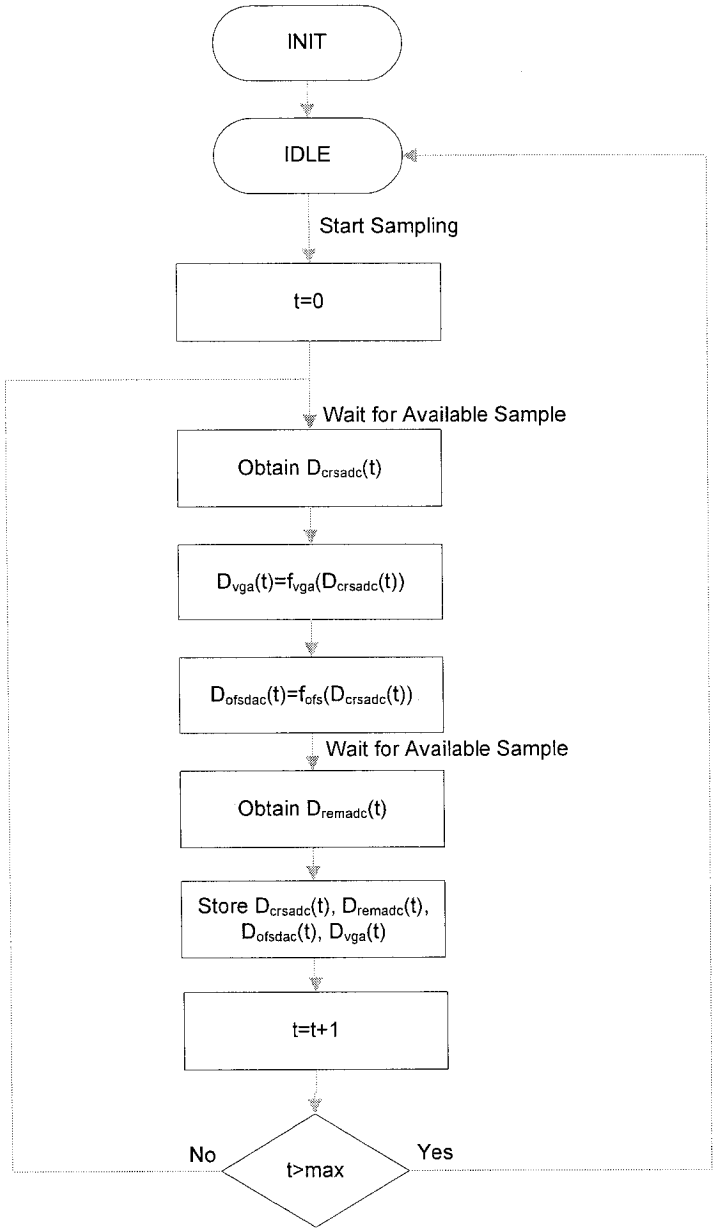


Figure 13 – State Flow Diagram of Quantizing Algorithm

The functions $f_{vga}()$ and $f_{ofs}()$ in the above figure are realized using the hardware lookup tables *gain_lut[]* and *offset_lut[]*, described in Section 3.1.1.

3.1.3 Correction Process

The correction process is executed by the software after the quantization of the intermediate values during the QUANTIZING cycle process. It retrieves the intermediate

sample values from the FPGA cache memory and calculates the final floating-point samples from them.

Figure 14 presents the main correction process state flow diagram.

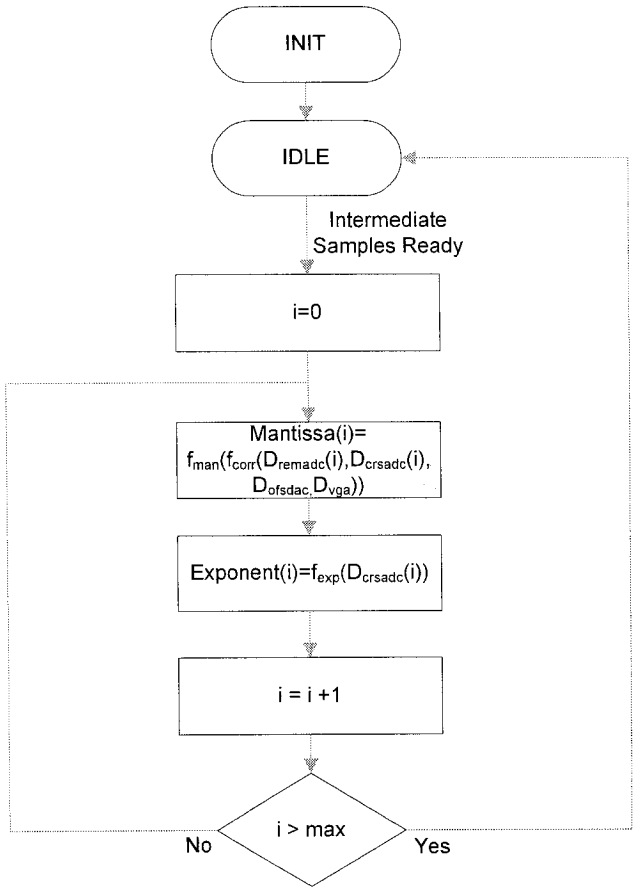


Figure 14 – State Flow Diagram of Correction Algorithm

Functions $f_{man}()$ and $f_{exp}()$ which calculate the mantissa and exponent of the floating-point signal are described in Section 2.3.3.

3.1.4 Lookup Table Creation Process

The hardware lookup tables *offset_lut[]* and *gain_lut[]* are built from threshold tables *domain_threshold[]* and *domain_offset[]* which contain conversion domain threshold and offset values described in Section 2.3.1.

The lookup table creation takes place in the software. The threshold tables required by the algorithm are described in Table 2.

Parameter Name	Parameter Type	Function
domain_offset[0...num_gain]	*float*	User set DAC OFFSET for given conversion domain
domain_threshold[0...num_gain]	*float*	User set gain switching point for given conversion domain

Table 2 – Lookup Table Generation Parameters

The table creating mechanism exploits the fact that the conversion domain ranges are mostly symmetric across negative and positive values of *Vin* (except for a shift due to switching boundaries of the COARSE ADC, described in Section 2.3.1). For example, the system implemented in Chapter 4 has four different gain settings. This results in eight total conversion domains (four negative, and four positive), as required. The needed threshold tables only need to have four entries each.

Figure 15 describes the state diagram of the algorithm used to convert the *threshold tables* into the hardware *lookup tables* (which is performed prior to the execution of the QUANTIZING process).

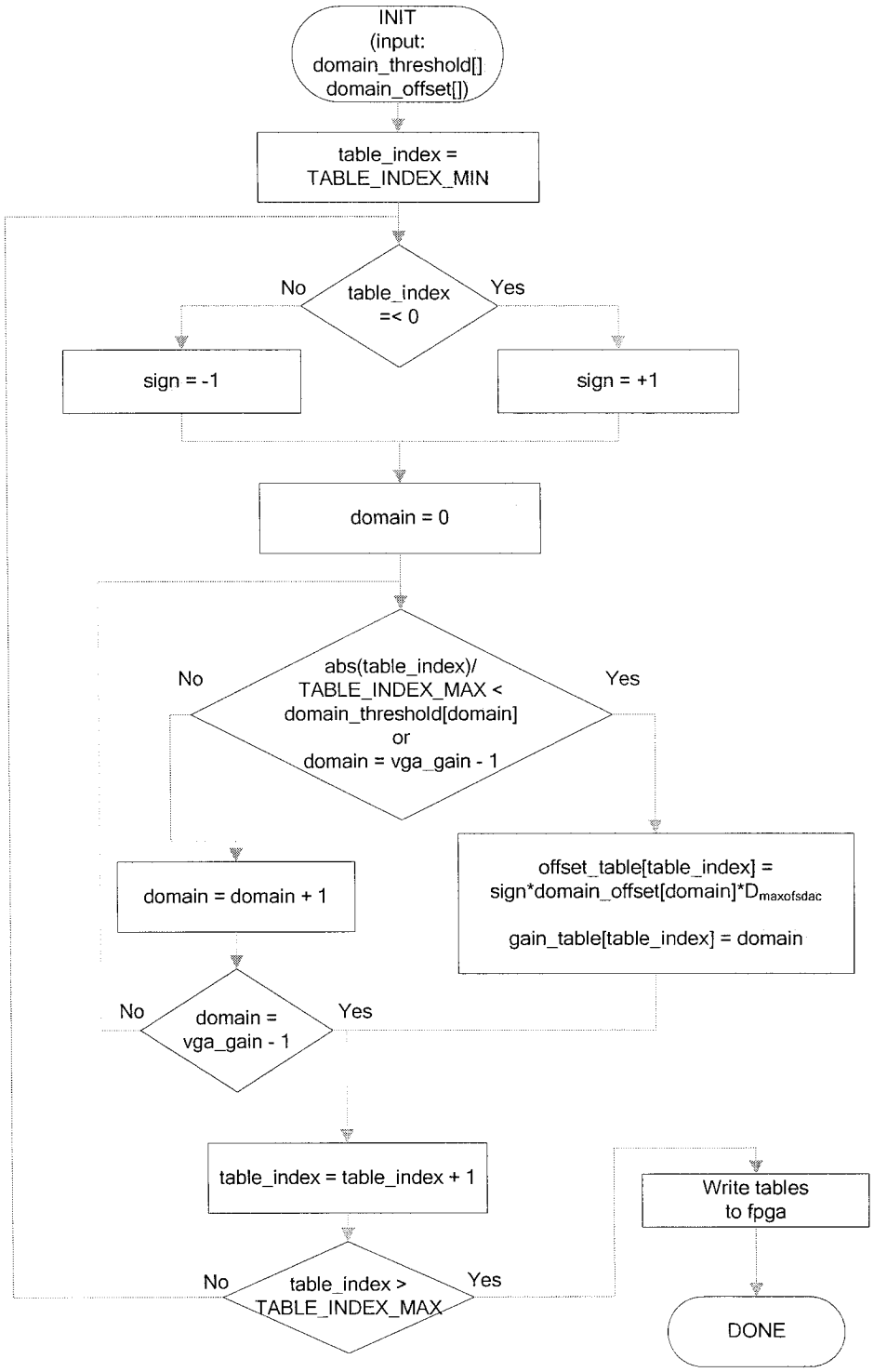


Figure 15 – State Flow Diagram of Algorithm for Building FPGA Lookup Tables

3.2 Calibration Cycle

The calibration logic measures the static errors in the output of the VGA and saves the correction values in a set of correction registers. The calibration mechanism is independent of the quantizing mechanism, although the quantizing mechanism is dependant on the calibration results. The QUANTIZATION cycle cannot be executed during the CALIBRATION cycle, and vice-versa. The calibration logic is mainly contained within the software portion of the FPADC.

The calibration mechanism measures the following system inaccuracies (all are measured while the input signal is set to zero:

- VGA offset error $\varepsilon Vvgaoffset_n$: Obtained by measuring the CALIBRATION ADC value for each conversion domain
- VGA gain error: Calculated by measuring the CALIBRATION ADC values for constant OFFSET DAC values for each VGA gain (positive and negative DAC value, typically +0.5 and -0.5 of max value *Dmaxofsdac*), and then obtaining the ratio of the gains

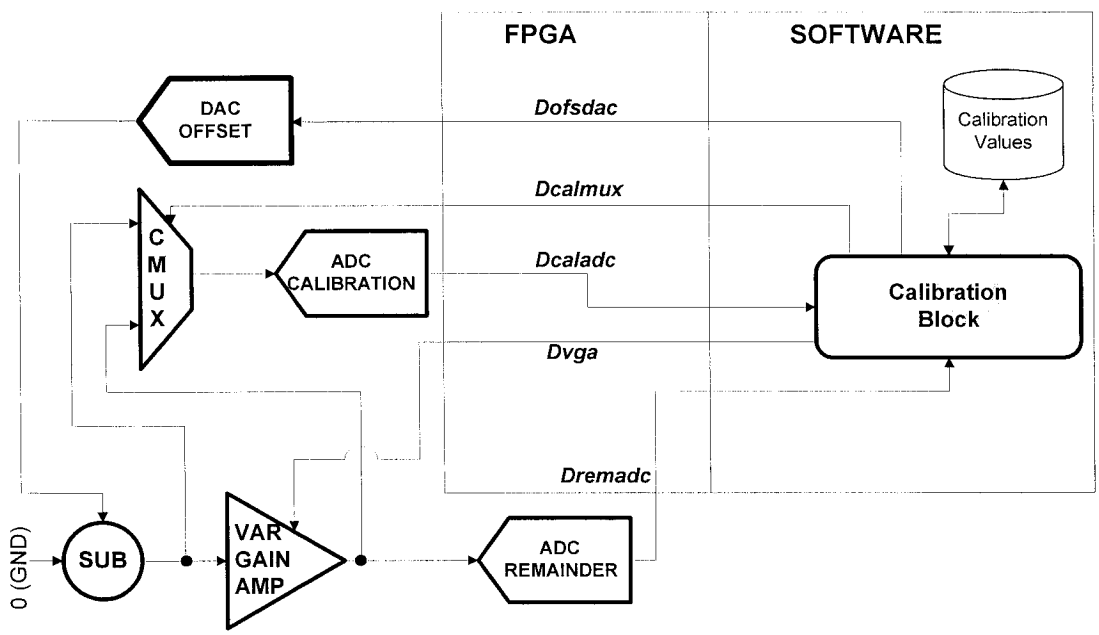


Figure 16 – Calibration algorithm diagram

Figure 16 describes the interconnections between the FPADC components while the CALIBRATION cycle is active.

The calibration values obtained during the CALIBRATION cycle are subsequently used in building the FPGA lookup tables *gain_lut[]* and *offset_lut[]* (described in Section 3.1.4) and in calculation of the final FPADC sample values (described in Section 3.1.3).

The parameters obtained during the calibration are listed in Table 3.

Parameter Name:	*gain_value[0...num_gain]*	**C Type:** *float*
Function:	True VGA gain as measured by the CALIBRATION ADC for each conversion domain.	
Parameter Name:	*gain_remadc_to_caladc_step*	**C Type:** *float*
Function:	Multiplicant used to obtain the CALIBRATION ADC value, based on the REMAINDER ADC value for.	
Parameter Name:	*gain_ofsdac_to_caladc_step*	**C Type:** *float*
Function:	Multiplicant used to obtain CALIBRATION ADC value based on OFFSET DAC value.	
Parameter Name:	*gain_bias_caladc[0...num_gain]*	**C Type:** *float*
Function:	Measured residual bias seen by CALIBRATION ADC when input is 0.	

Table 3 – Calibration Parameters

3.2.1 VGA Gain Calculation

The VGA Gain is used in final sample calculation and in the FPGA lookup table creation. The gain is calculated for each of the VGA gain settings by finding the slope of the two

obtained points, described in Section 2.3.2. The two points are the CALIBRATION ADC values obtained while the OFFSET DAC is set to a fixed negative and positive value ($+0.5*D_{maxofsdac}/(2^{num\_gains-Dvga})$ and $-0.5*D_{maxofsdac}/(2^{num\_gains-Dvga})$).

The process is described by the flowchart in Figure 17.

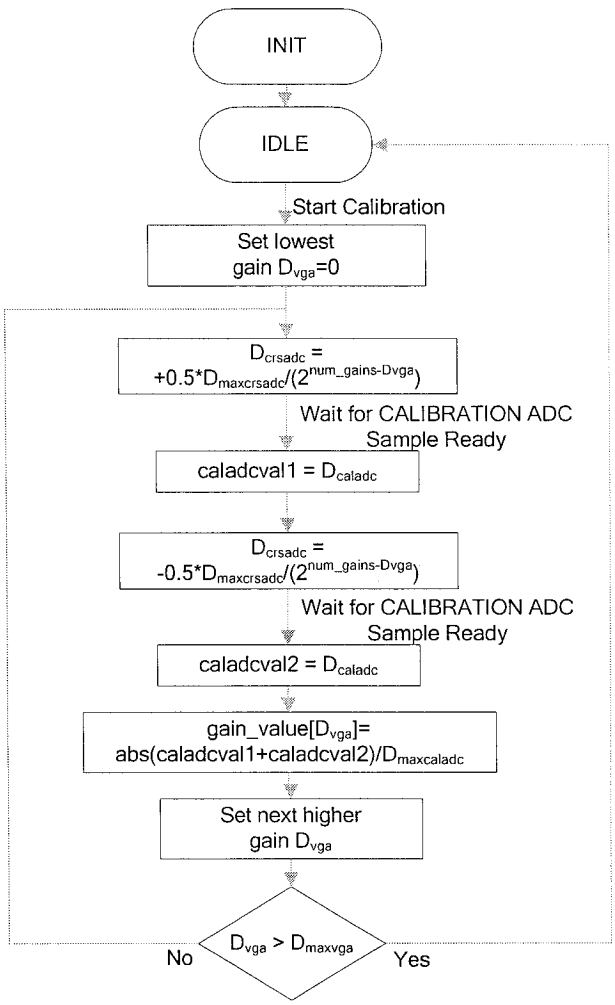


Figure 17 – State Flow Diagram of Gain Calibration Algorithm

3.2.2 VGA Offset Calculation

The VGA Offset is used in final sample calculation and in lookup table creation. The offset is calculated for each of the VGA gains by finding the value of the CALIBRATION ADC when the value of the OFFSET DAC is zero.

The process is described by the flowchart in Figure 18.

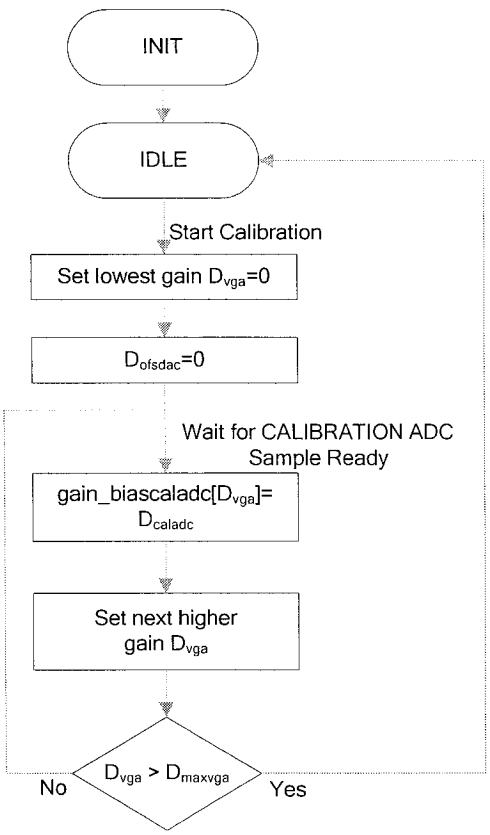


Figure 18 – State Flow Diagram of Gain Offset Calibration Algorithm

3.2.3 OFFSET-DAC to REMAINDER-ADC Multiplicant Calculation

This multiplicant is used in converting OFFSET DAC value to an equivalent REMAINDER ADC value. Two OFFSET DAC values are set and two sets of CALIBRATION ADC and REMAINDER ADC measurements taken. The multiplicant is the ratio of the slopes created by the respective CALIBRATION ADC and two

REMAINDER ADC points. The two OFFSET DAC values are +0.5*Dmaxofsdac and -0.5*Dmaxofsdac. The calculation is repeated for each of the possible VGA gain settings.

The process is described by the flowchart in Figure 19.

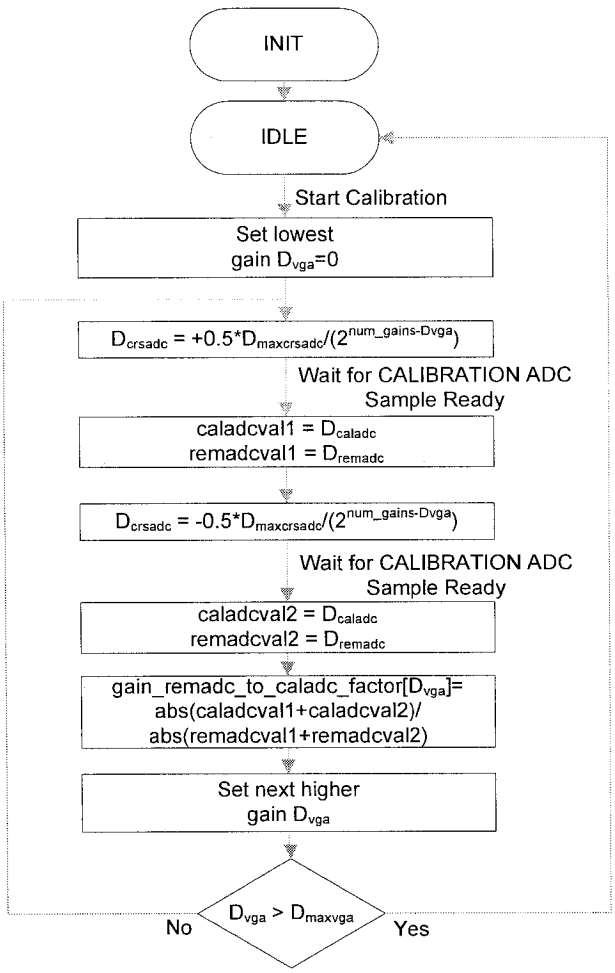


Figure 19 – State Flow Diagram of REMAINDER-ADC to CALIBRATION-ADC Multiplicant Calculation Algorithm

This calculation is performed when the REMAINDER ADC is operating at the full resolution of 14 bits in order to obtain the most accurate results. If the ADC is to be operated at 4 bits, a different algorithm has to be used. The algorithm has to look for the REMAINDER ADC transition changes while changing the OFFSET DAC values.

3.2.4 OFFSET-DAC to CALIBRATION-ADC Multiplicant Calculation

This multiplicant is used in converting CALIBRATION ADC value to an equivalent OFFSET DAC value. The multiplicant is calculated by setting two OFFSET DAC values and corresponding CALIBRATION ADC measurements taken. The multiplicant is the ratio of the slopes of the two obtained CALIBRATION ADC points and the two OFFSET DAC values. The two OFFSET DAC values are +0.5*Dmaxofsdac and -0.5*Dmaxofsdac.

The process is described by the flowchart in Figure 20.

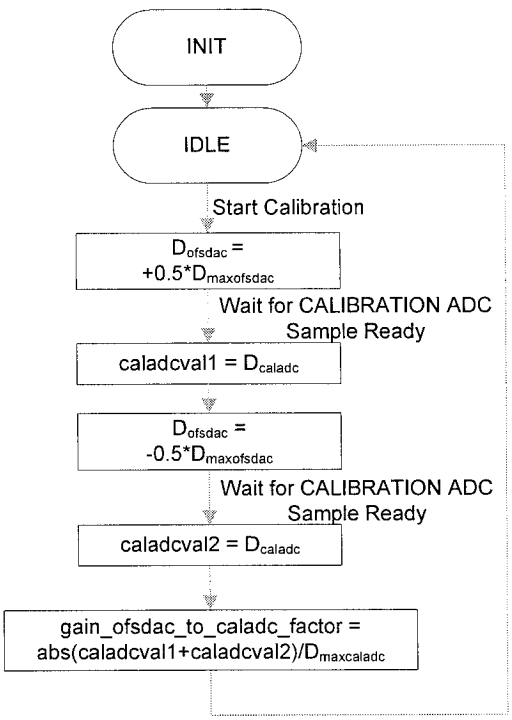


Figure 20 – State Flow Diagram of OFFSET-DAC to CALIBRATION-ADC Multiplicant Calculation Algorithm

4 Implementation

4.1 Background Information

The system implements a Self-Calibrating Floating Point Analog to Digital Converter, as outlined earlier. The implementation consists of a software component (control application running under Microsoft Windows XP OS), which lets the user interact with the FPADC. More specifically, the software allows for interfacing with hardware FPGA component (state machines responsible for communication with the control app and hardware components), and the mixed-signal hardware components (DAC, ADCs, AMPs). An off-shelf hardware development kit is used as a main board for the prototype (Altera board w/ Altera EP1S80 FPGA, DAC and ADCs). The FPADC logic is implemented in an FPGA residing on the main board.

4.2 Hardware

The FPADC circuit is implemented using an Altera Stratix EP1S80 FPGA development board [12]. The board has onboard ADCs, DACs, RS232 communication port, SRAM and FLASH memories, clock, and LEDs, as outlined in Figure 21.

The Altera board is well equipped for the development of the FPADC. All of the high-connection count components are present on-board: OFFSET DAC, REMAINDER ADC and COARSE ADC. The remaining components making up the complete FPADC system are positioned off-board on a daughtercard which is connected to the Altera Development system through one of its expansion connectors.

The FPADC was originally developed using a HOT2 PCI development system from Virtual Computer Corporation (Figure 22). The board consisted of a PCI add-on card and a prototyping daughtercard hosting all of analog and mixed-signal FPADC components (COARSE and REMAINDER ADCs, OFFSET DAC, CALIBRATION ADC, VGA and MUX). Due to a hardware failure of the non-volatile configuration memory on the HOT2 board, the prototype was redesigned and recoded for the Altera prototyping system.

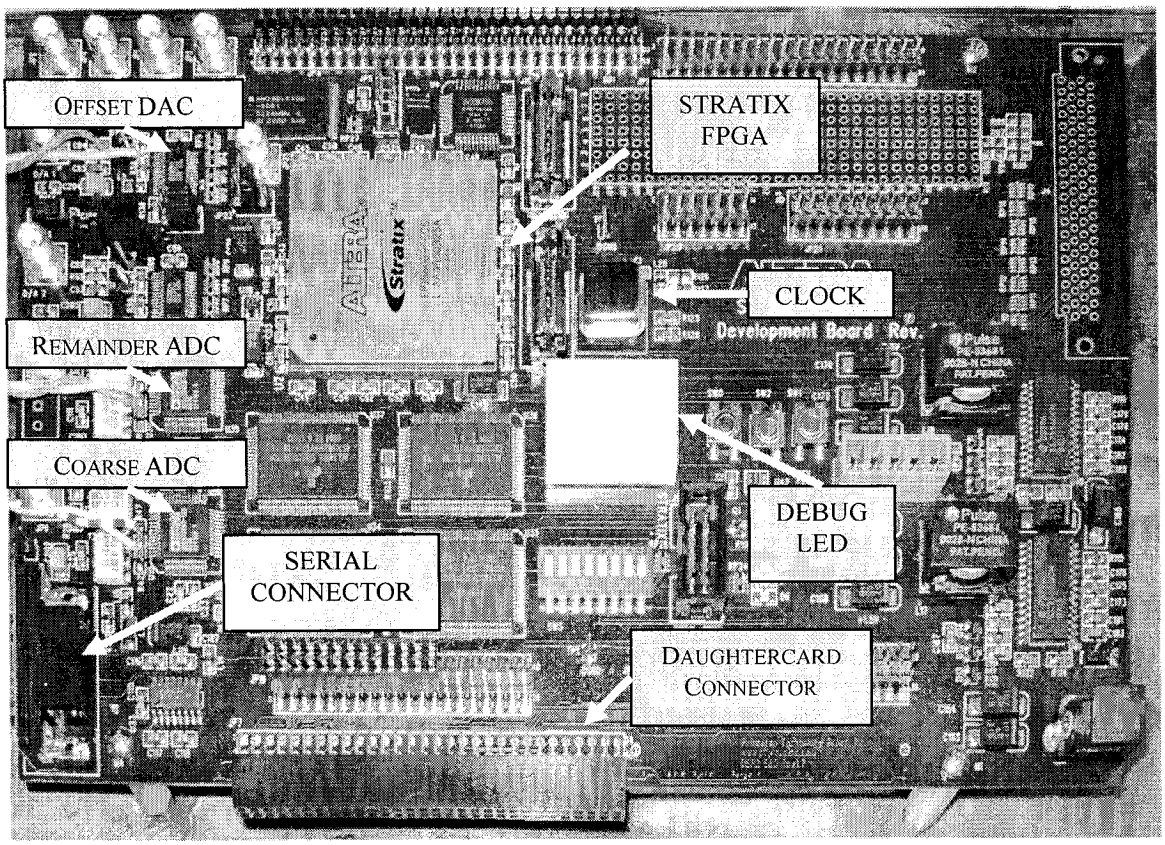


Figure 21 - Stratix EP1S80 Board – Connectors and Components

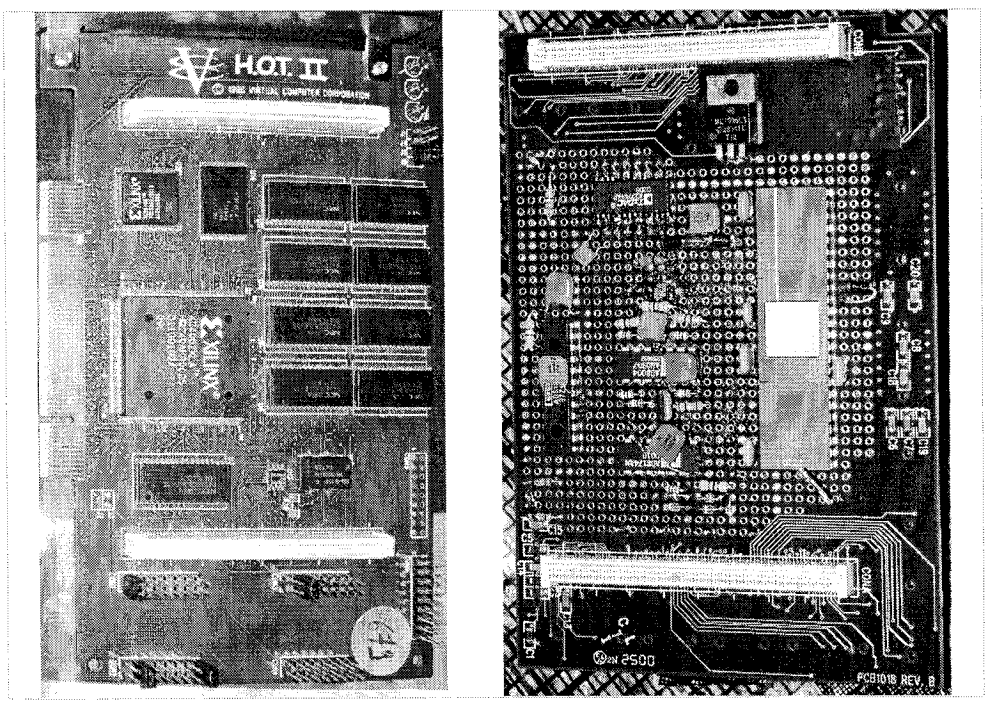


Figure 22 – Self-Calibrating FPADC Design using HOT2 Xilinx PCI Board

4.2.1 Daughtercard

The Altera Stratix board does not have a complete set of components needed for the implementation of the proposed FPADC. Therefore, a daughtercard is used which houses the required components. The daughtercard connects with the EP1S80 board through an on-board connector. The connector chosen for this task is JP19 due to its closeness to the DACs and ADCs, as outlined in Figure 21.

Figure 23 illustrates the FPADC daughtercard connection to the EP1S80 board.

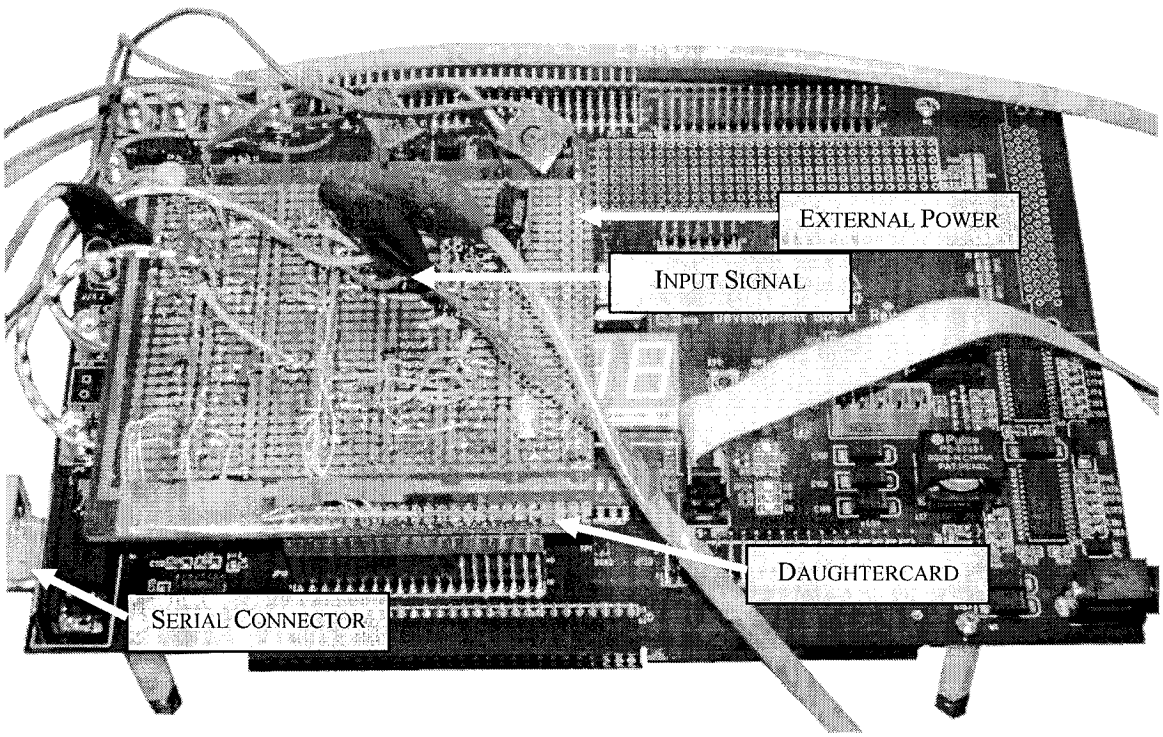


Figure 23 - Stratix EP1S80 evaluation board with FPADC daughtercard

The diagram in Figure 24 illustrates the interconnections of the daughtercard components. Figure 26 shows the placement of the components as implemented on the FPADC daughtercard.

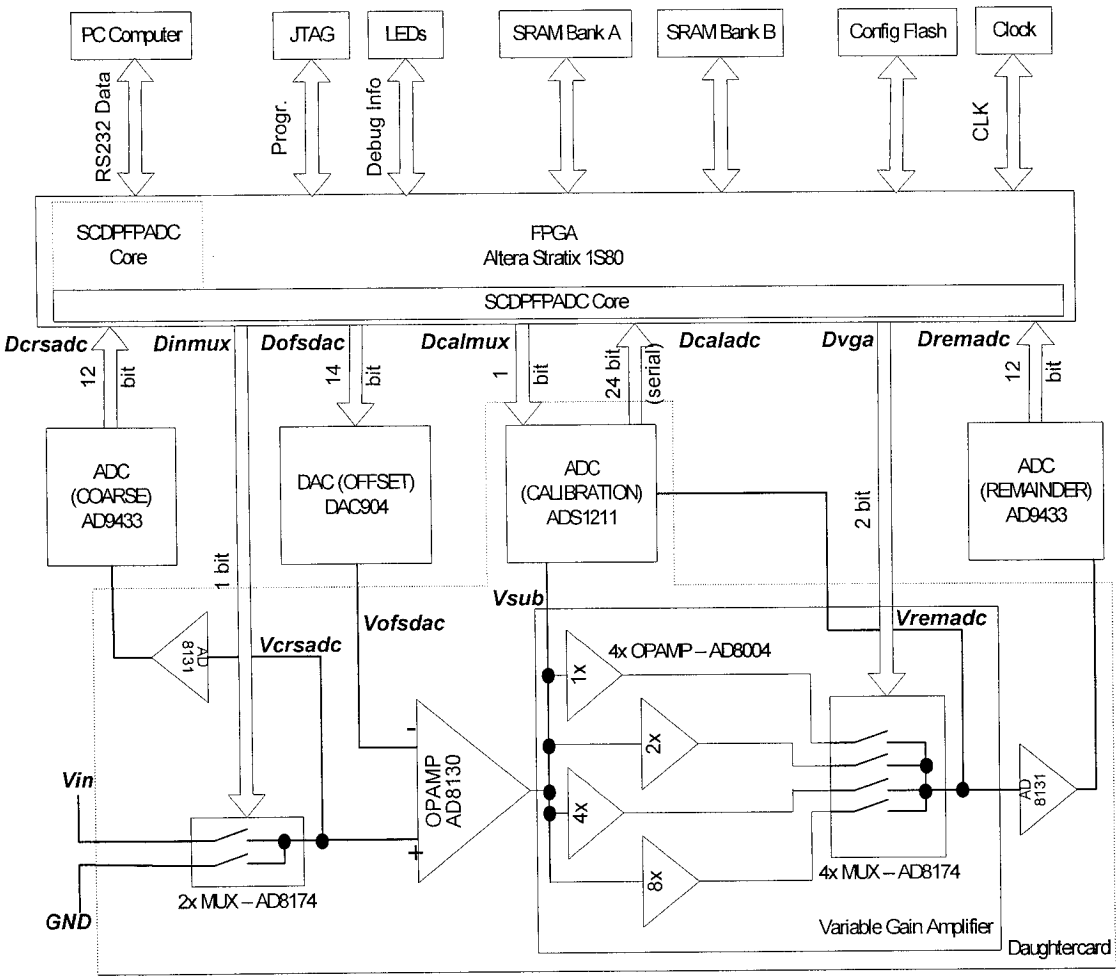


Figure 24 - Implementation of FPADC on Altera Stratix EP1S80 board

The schematic of the daughterboard is subdivided into five functional blocks, shown in Figure 25. The function blocks, along with major ICs used, are:

- Variable Gain Amp: AD8004 Quad OpAmp, AD8174 Quad Mux
- Calibration ADC: ADS1211 Serial ADC
- Input MUX: AD8174 Quad Mux
- Subtractor: AD8130 Differential Receiver
- Coarse ADC Driver: AD8131 Differential Transmitter
- Remainder ADC Driver: AD8131 Differential Transmitter

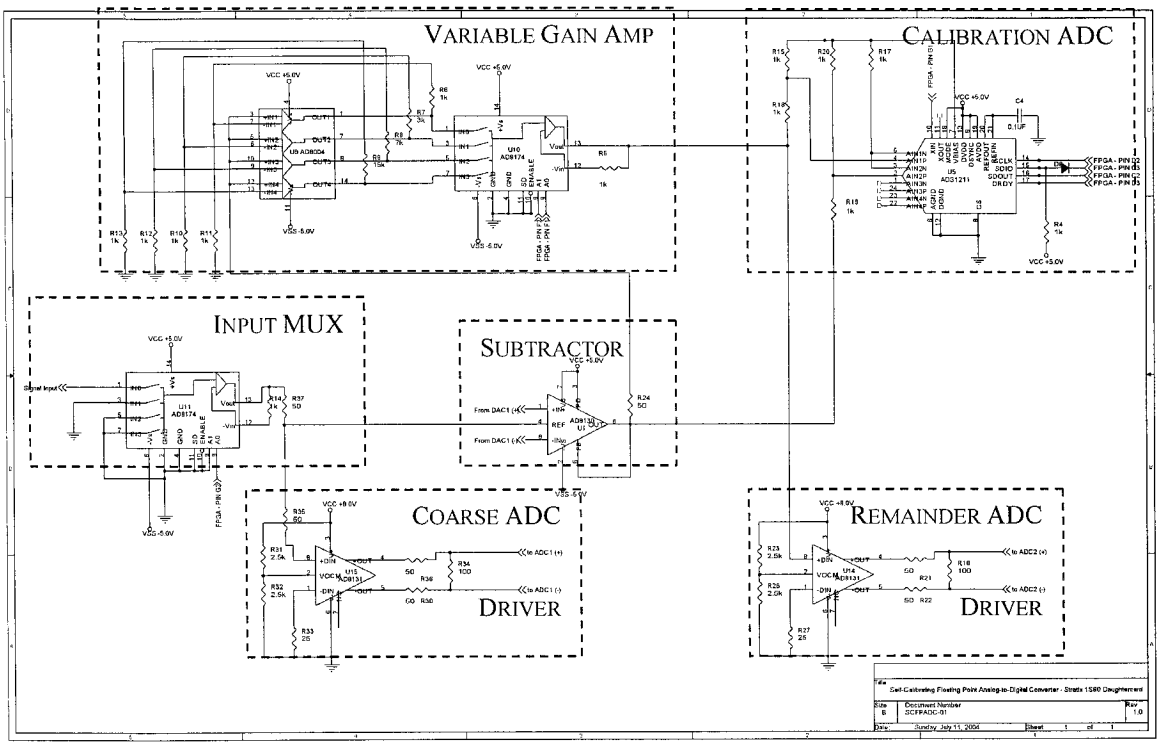


Figure 25 - Schematic of FPADC Daughtercard

The functional blocks serve the following roles:

- Input MUX: Selects either a ground level or input signal level
- Variable gain amp: Four operational amplifiers with different gains, and 4-to-1 mux to select the amplified signal
- Coarse ADC driver: Converts and biases a single-ended signal to a 4V bias differential signal required by the coarse ADC
- Remainder ADC driver: Converts and biases a single-ended signal to a 4V bias differential signal required by the remainder ADC
- Subtractor: Subtracts the single ended signal from the input mux to a negative of the differential voltage from the DAC

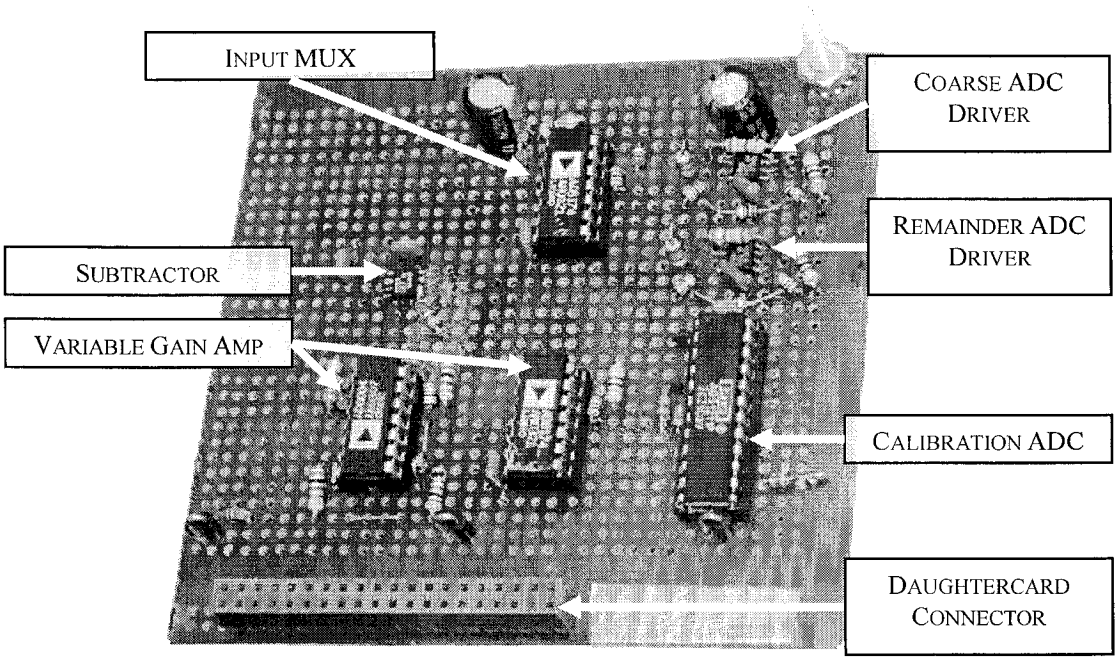


Figure 26 - Picture of FPADC Daughter-card Components

4.3 Programmable Logic

4.3.1 Background Information

The FPGA component is responsible for processing the commands issued by the Software Control Application, and relaying the appropriate commands to the underlying FPADC hardware. The commands that will result in hardware interaction are DAC, ADC, VGA, CLOCK and LED commands. The component ensures the proper clocking and timing in such way that the Software Control Application can run almost asynchronously (no timing information will be generated by the software, except for certain debugging functions). It also maintains internal state information such as the Calibration Tables, Run-Time and LED states.

The FPGA component consists of the following state machines:

- RS232 command read/result send process
- Parallel (Offset/Coarse/Remainder) ADC/DAC read/write latching process

- Quantizing process: Writes partial FPADC sample values to the internal SRAM memory using correction tables
- Calibration process: Responsible for generating the calibration values and gathering correction values – offloaded by the software
- Serial (Calibration) ADC command read/write process

The FPGA code is implemented in VHDL and compiled under Altera Quartus II 3.0. The FPGA code manages the following functions:

- Generating clocks for the external devices (ADCs, DACs, Calibration ADC)
- Gathering sample values from the ADCs
- Setting values of the DACs
- Providing Serial IO communication for Calibration ADC
- Providing RS232 communication for interconnection with a PC
- Reading and writing of the sample memory cache
- Reading and writing of the lookup tables
- Performing 'Fast Quantizing' operation where the FPADC fills up the sample cache memory with the sample values
- Providing debugging functionality

The FPADC FPGA code is split into the following functional blocks:

- FPADC_TOP: Top FPGA entity which combines all other FPADC FPGA components
- FPADC_MAIN: Core of the FPADC which processes software control application communication packets
- FPADC_RS232: Byte-level communication UART for RS232 communication
- FPADC_PKTPROCESSOR: Serializer/deserializer module responsible for assembling command packets from the incoming byte stream (FPADC_RS232 sending data to FPADC_MAIN) and transmitting packets into the outgoing byte stream (FPADC_MAIN sending data to FPADC_RS232)
- FPADC_SIO: Command-level communication UART for communication using SIO protocol as implemented in the AD1211 ADC.

- FPADC_LED: Debugging module driving the onboard LED display. This module takes a binary-encoded number and converts it to HEX form, suitable for display on 7-segment LED array.

Figure 27 presents the internal connections between the FPADC FPGA modules.

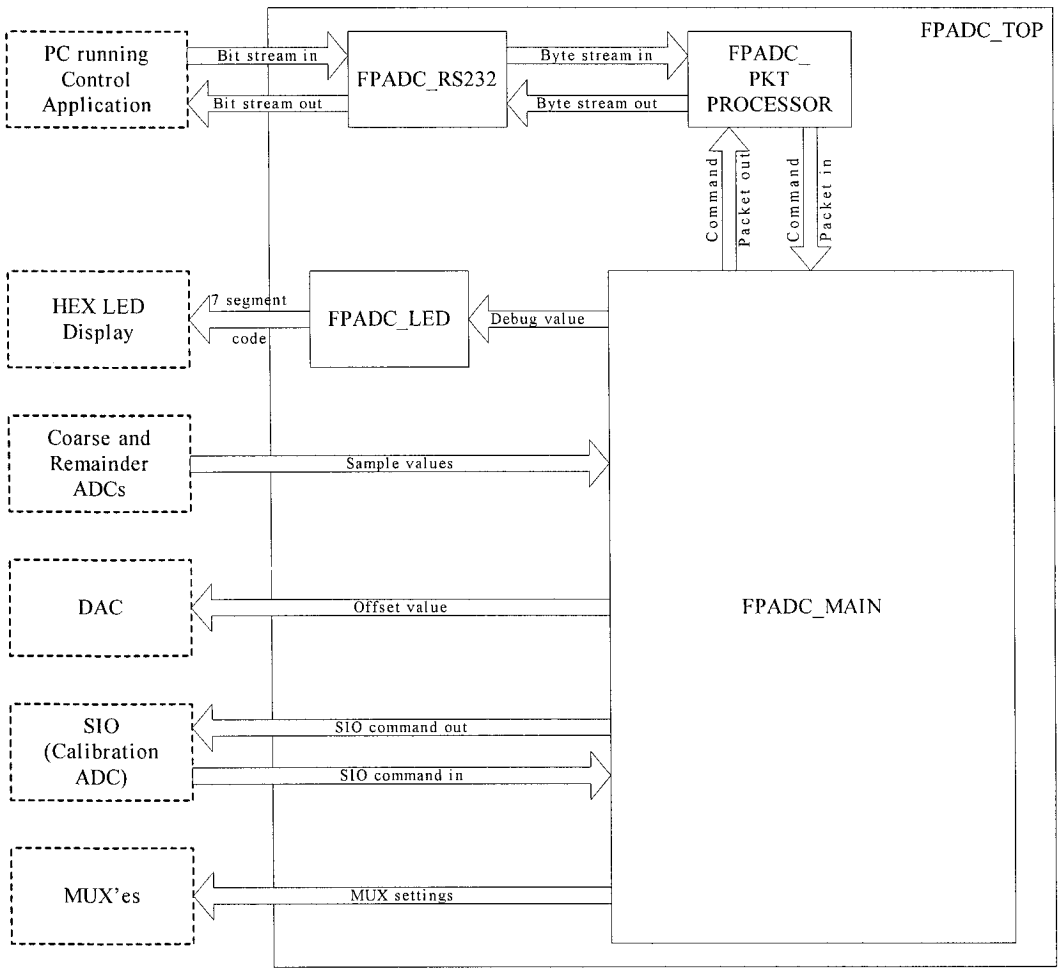


Figure 27 - FPGA Component Internal Datapaths

The FPGA_MAIN is the core module responsible for the user-visible FPGA functions. It processes and executes user commands in the decoded user packets. The FPGA_MAIN module has direct access to the mixed-signal component control signals. It also hosts the

gain_lut[] and *offset_lut*[] lookup tables. The module comprises of the following processes:

- Fast Quantizing: performs full FPADC sampling, lookup and cache access
- Input/ Output: latches in COARSE ADC and REMAINDER ADC values, latches out DAC/MUX values
- Command Processing: decodes user command packets and forms reply packets

4.3.2 Description of Quantizing and IO process in FPADC_MAIN

The process consists of a sixteen level state machine triggered by a falling edge of the DAC/ADC clock at the rate of $4*10^7$ triggers/sec (25 ns/trigger). It executes the algorithm described in Figure 28 while performing the QUANTIZING cycle.

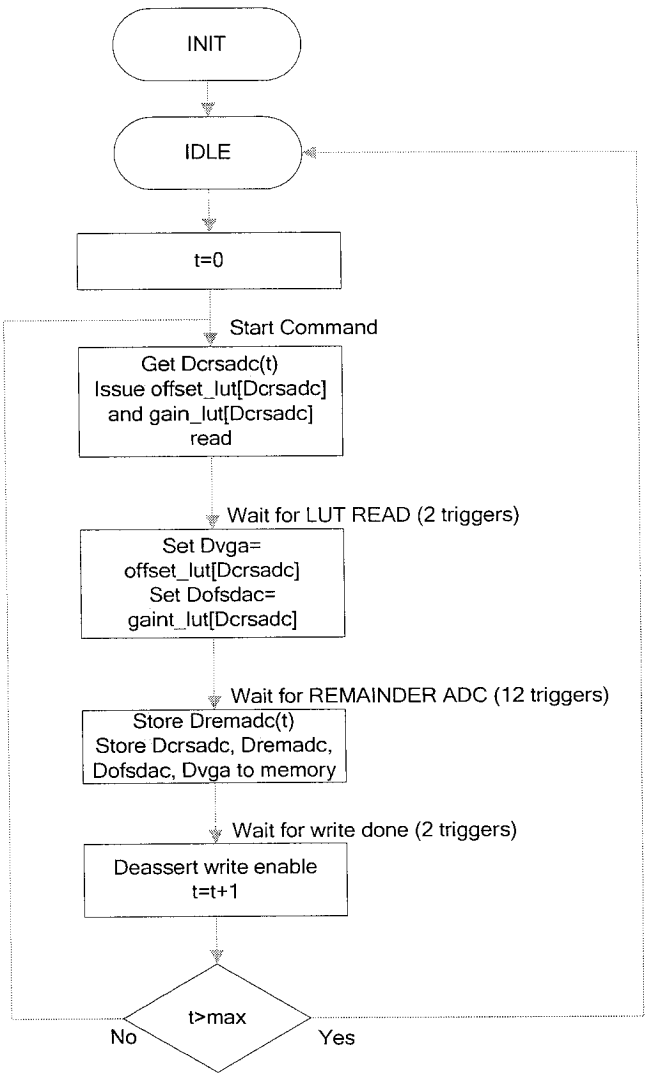


Figure 28 – State Flow Diagram of Quantizing Algorithm – FPGA portion

4.4 Control Software

4.4.1 Background Information

The control application enables the user to interact with the FPADC system in a user-friendly manner. It also provides correction and calibration functions of the FPADC. In particular, the application has the following functionality:

- Digital Correction of FPADC Signal
- Calibration Functionality of FPADC System
- Communication with FPADC FPGA component

- Resetting the FPADC FPGA component to a default state
- Loading and initialization of the FPADC FPGA lookup tables
- Debug display of instantaneous internal values of the FPADC FPGA
- Plotting final and subcomponent FPADC values
- Saving plotted values for further analysis

4.4.2 Software Application Architecture

The software application consists of the following modules:

- RS232 Communication: Transmits and receives the FPADC commands
- FPADC Library: Performs high-level FPADC functions, translating them to appropriate commands, and returns command results (for example, performs quantizing and download all memory values)
- User Command: Breaks user requests from the GUI interface into appropriate FPADC library calls. Displays and saves gathered results

The architecture of the software application is shown in

Figure 29.

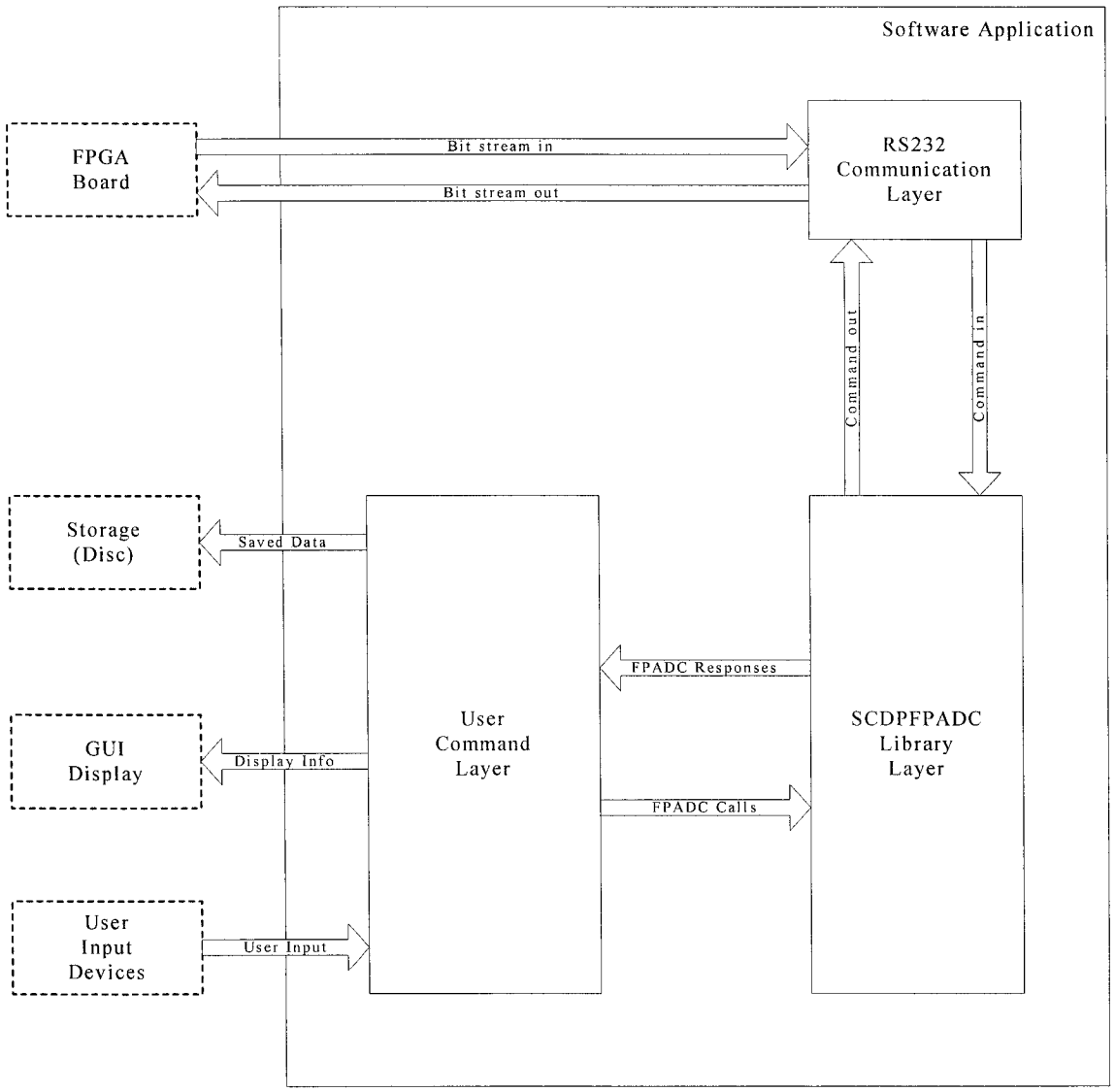


Figure 29 – Internal Architecture of the Software Application

4.4.3 Detailed Description

The control application is implemented in Microsoft Visual C for the Windows OS. Upon loading the application the main window, shown in Figure 30, is displayed.

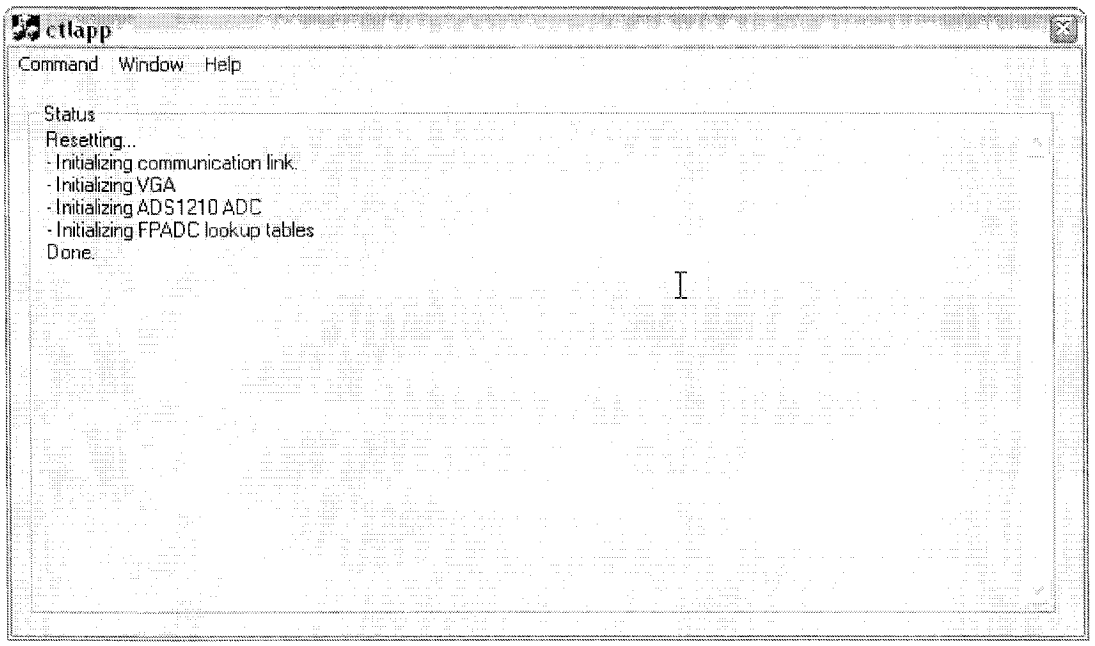


Figure 30 - Main Window IDD_CTLAPP_DIALOG

4.4.3.1 Main Control Dialog

The main application window IDD_CTLAPP_DIALOG displays a drop-down listing of the FPADC functions, shown in Figure 31. It also provides a text scroll area for debugging information. The menu provides the following options:

- **Command** – Executes a command on the FPADC FPGA component
- **Window** – Displays other windows
- **Help** – Displays help menu

The **Command** option has the following sub-options:

- **Reset** – Resets the FPADC to default values
- **Calibrate** – Issues FPADC CALIBRATION cycle
- **Acquire** – Issues FPADC QUANTIZING cycle (fill internal FPGA memory)
- **Memory Dump** – Provides printout of contents of FPADC FPGA internal memory
- **User LED State** – Change state of FPADC debug LED
- **Sample dump** – Presents a parsed printout of FPADC FPGA internal memory

- **Calibration dump** – Prints out contents of calibration tables
- **Exit** – Quits the program

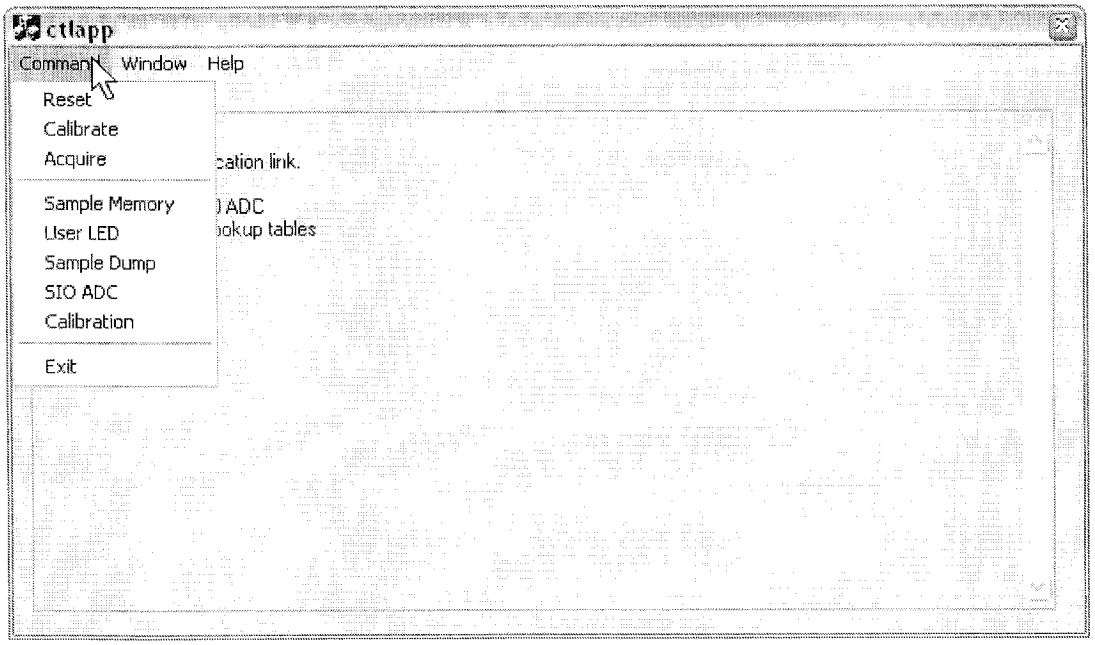


Figure 31 - Example of available menus

The **Window** option opens the Graphing Window (IDD_WAVEFORM_DIALOG).

The **Help** option has the following sub-option:

- **About** – Information window (IDD_ABOUTBOX) which displays program version and copyright information

4.4.3.2 Graphing Waveform Window

The Graphical Waveform Display window (IDD_WAVEFORM_DIALOG) provides a means for the user to interact with the FPADC system while displaying the sampled data in a graphical manner. This provides a graphical waveform display area, "CLOSE" button to close the window, Data Source buttons (Software-Driven Calibration, Remainder and Coarse ADCs, Hardware-Driven Remainder and Coarse ADCs, Hardware-Driven corrected FPADC quantizing) and "Trigger" and "Auto" buttons which control manual and auto-refreshing of the waveform area.

The graphing window IDD_WAVEFORM_DIALOG consists of user controls and a graphing area depicted in Figure 32.

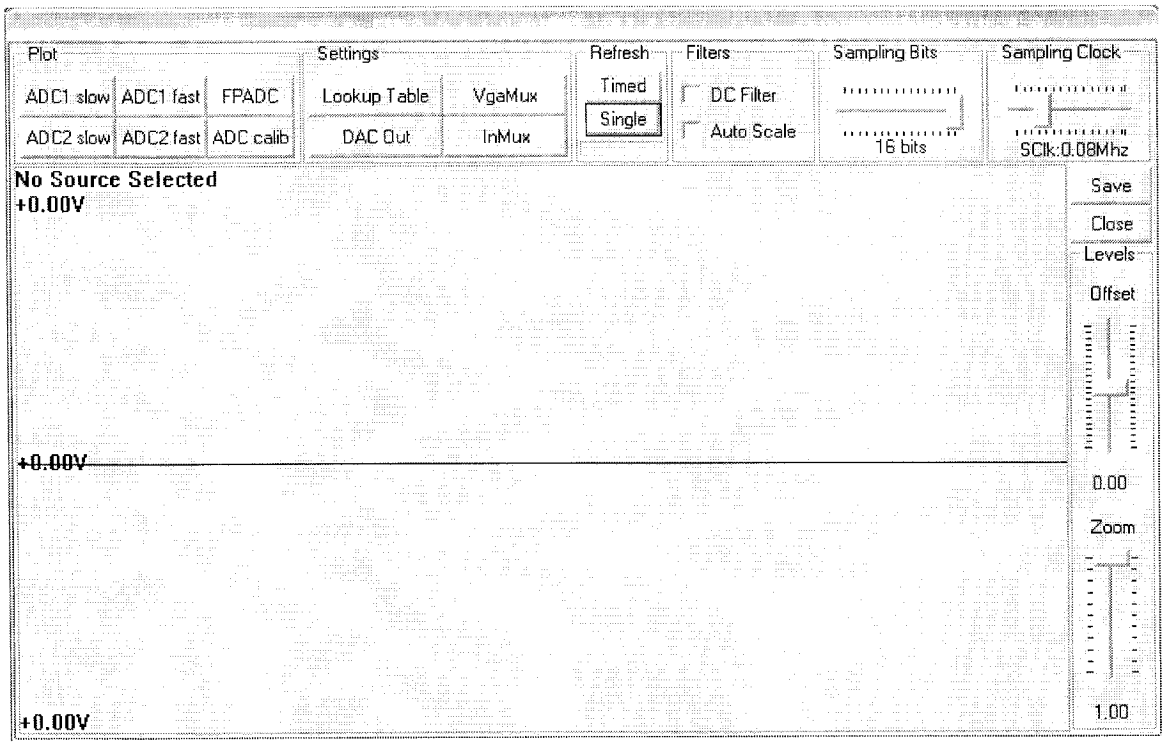


Figure 32 - Plotting window

The options available to the user are as follows:

- **ADC1 Slow Plot:** Plots values of the Remainder ADC using the software for the time base
- **ADC2 Slow Plot:** Plots values of the Coarse ADC using the software for the time base
- **ADC1 Fast Plot:** Plots values of Remainder ADC using the FPGA for the time base
- **ADC2 Fast Plot**: Plots values of Coarse ADC using the FPGA for the time base
- **FPADC Plot:** Plots values of FPADC using the FPGA for the time base
- **ADC Calib Plot:** Plots values of the calibration ADC using the software for the time base

- **Lookup Table Settings:** Generates the lookup tables
- **DAC Out Settings:** Cycles through all possible values of the DAC
- **VGA Mux Settings:** Cycles through all possible values of the VGA
- **IN Mux Settings**: Toggles the Input MUX
- **Timed Refresh:** Enables auto-refresh of the currently selected plot
- **Single Refresh**: Disables auto-refresh and performs manual refresh of the currently selected Plot
- **DC Filter**: Centers the currently visible plot in the y-domain
- **Auto Scale Filter:** Maximizes the current plot in the y-domain
- **Quantizing Clock**: Sets the value of the FPADC quantizing clock
- **Quantizing Bits**: Sets the number of mask bits for Remainder and Coarse ADCs
- **Offset Level** : Varies the vertical offset of the current plot
- **Zoom Level:** Varies the magnification of the current plot

Figure 33 shows an example reconstructed FPADC signal which the user sees after pressing the 'FPADC' button after the lookup tables are initialized. The input mux is set to external input, and a sine signal is present at the system input.

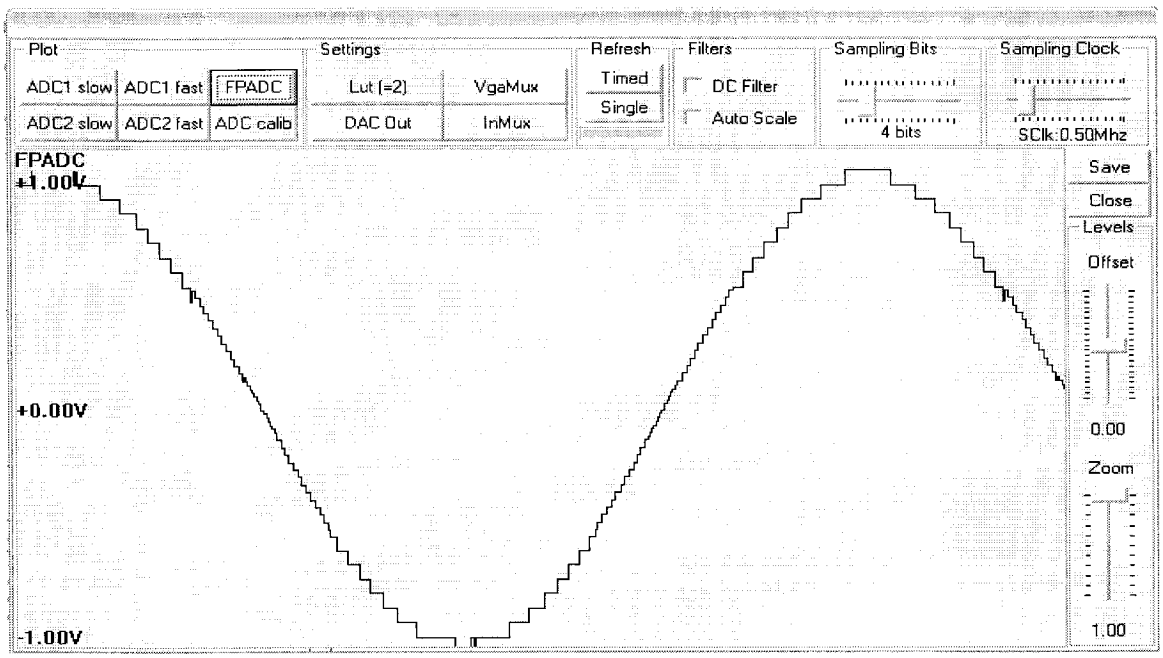


Figure 33 – Typical FPADC signal plot

4.4.4 Software to FPGA Communication Protocol

The PC communicates with the FPADC FPGA through a packet protocol. The PC acts as a communication master and the FPADC acts as a slave. The PC sends commands to the FPADC and reads the FPADC response (if available for given command).

The commands used for communication with the FPADC FPGA component can be broken into the following:

- Setting value of range DAC
- Obtaining values from Coarse and Remainder ADCs
- Reading commands from and writing commands to Calibration ADCs
- Setting gain value of Variable Gain Amplifier (VGA)
- Setting MUX values
- Initiating QUANTIZING cycle
- Reading internal cache memory
- Writing internal lookup tables

The sequence of events during a typical MASTER-SLAVE communication exchange is presented Figure 34.

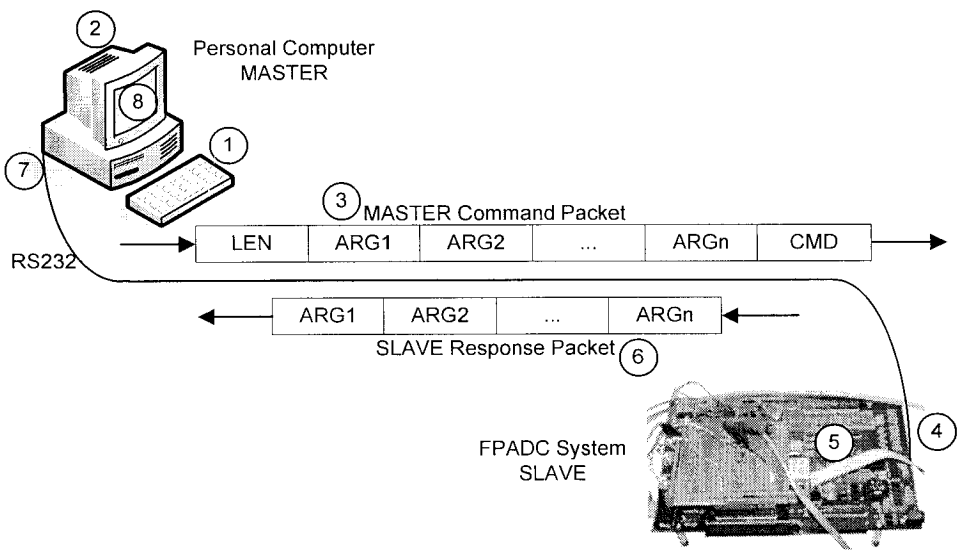


Figure 34 – Control Application to FPADC FPGA Communication Sequence

The sequence of events in the above figure is as follows:

1. User or Program issues a call to the FPADC library to perform an FPADC operation
2. The FPADC library assembles a MASTER communication packet
3. The MASTER communication packet is sent to the FPADC system
4. The FPADC system receives the command
5. The FPADC executes the command, and assembles a SLAVE packet which contains the results of the command
6. The SLAVE communication packet is transmitted to the Control Application
7. The control application receives the SLAVE response command, and interprets it accordingly
8. The results are passed to the originator of the library call

4.4.4.1 Master-to-Slave Packet Structure

The MASTER to SLAVE packet consists of the length indicator, command number, and a variable list of arguments. The command packet has the format depicted in Table 4 where a command with a single 32 bit argument is shown.

Bit 00 ... 07	Bit 08 ... 15	Bit 16 ... 23	Bit 24 ... 31	Bit 32 ... 39	Bit 40 ... 47
PKT LEN	ARG1	ARG2	ARG3	ARG4	CMD

Table 4 – Software-to-FPGA Command Packet Structure

The command has the following fields:

- PKTLEN – Length of the packet in bytes minus 2 (PKTLEN and CMD fields are always sent)
- ARG1 – Least significant byte of the argument
- ARG2 – Second least significant byte of the argument
- ARG3 – Second most significant byte of the argument
- ARG4 – Most significant byte of the argument

- CMD – Command to be executed. Different commands have different argument lengths, as specified by the table of commands

4.4.4.2 Slave-to-Master Packet Structure

The SLAVE to MASTER response packet consists of a number of 32 bit words determined by the type of the command sent. Table 5 presents the format of a single 32 bit response word.

Bit 00 … 07	Bit 08 … 15	Bit 16 … 23	Bit 24 … 31
ARG4	ARG3	ARG2	ARG1

Table 5 – FPGA-to-Software Response Packet Structure

The response packet has the following fields:

- ARG4 – Most significant byte of the argument
- ARG3 – Second most significant byte of the argument
- ARG2 – Second least significant byte of the argument
- ARG1 – Least significant byte of the argument

4.4.4.3 Master Command List

The FPADC system responds to the list of commands presented in Table 6.
The input of 'n/a' implies that the MASTER command has no input arguments. The output of 'n/a' means that the SLAVE response carries has no relevant data.

Command Name:	*READ_ADC1*		
Input:	n/a	**Response:**	*adc_val*
Description:	Return instantaneous quantized value *adc_val* from the Coarse ADC		
Command Name:	*READ_ADC2*		

Input:	n/a	**Response:**	*adc_val*
Description:	Return instantaneous quantized value *adc_val* from the Remainder ADC		
Command Name:	*WRITE_DAC1*		
Input:	*dac_val*	**Response:**	n/a
Description:	Set value of the output of Offset DAC to *dac_val*. The value holds until next WRITE_DAC1 command except for the duration of the SAMPLE_CYCLE command.		
Command Name:	*SIO_ADS1210_CMD*		
Input:	*cmd_out*	**Response:**	*cmd_in*
Description:	Issue command *cmd_out* on the serial IO bus connected to the ADS1210/ADS1211 ADC, and read the response *cmd_in* from the device.		
Command Name:	*WRITE_OFFSET_TABLE*		
Input:	*index, value*	**Response:**	n/a
Description:	Write *value* to the lookup table offset_lut[] at location *index*.		
Command Name:	*WRITE_GAIN_TABLE*		
Input:	*index, value*	**Response:**	n/a
Description:	Write *value* to the lookup table gain_lut[] at location *index*.		
Command Name:	*SAMPLE_CYCLE*		
Input:	*num_samples*	**Response:**	n/a
Description:	Issue QUANTIZING cycle which will fill up the FPGA cache		

	memory with *num_samples* intermediate values.
Command Name:	*READ_SAMPLE_HI_TABLE*
Input:	*index* **Response:** *value*
Description:	Read high 32 bits of the cache memory at location *index* and returns it in *value*.
Command Name:	*READ_SAMPLE_LO_TABLE*
Input:	*index* **Response:** *value*
Description:	Read low 32 bits of the cache memory at location *index* and returns it in *value*.
Command Name:	*SET_VGA_GAIN*
Input:	*vga_val* **Response:** n/a
Description:	Set value of gain of the variable gain amplifier to *vga_val*. The value holds until next SET_VGA_GAIN command except for the duration of the SAMPLE_CYCLE command.
Command Name:	*SET_CLOCK_SPEED*
Input:	*divider* **Response:** n/a
Description:	Set the clock divider used in the *SAMPLE_CYCLE* command. Quantizing speed is (20MHz/(*divider*+1)) samples/sec.
Command Name:	*SET_IN_MUX*
Input:	*mux_val* **Response:** n/a
Description:	Set the value of the input mux to *mux_val*. The value holds until next SET_IN_MUX command.

Command Name:	*SET_ADC_MASK*		
Input:	*mask*	**Response:**	n/a
Description:	Set the ADC bitmask which is logically AND-ed with the internal value of the Remainder and Coarse ADC to simulate different bit-width ADCs		

Table 6 – List of Communication Commands

4.5 Break-in Procedures

During the implementation stage the FPADC is brought up and tested in functional steps. Each of the steps verifies a standalone functionality of the system. The following list represents the functional blocks that are verified and tested in chronological order. Each module is tested for functional accuracy as well as implementation problems.

- Implement FPGA I/O core to blink an LED.
- Implement Host-to-FPADC communication protocol using RS232. Implement a FPADC command to blink used LED on the FPADC prototype board. Verify operation of the command state machine by issuing different LED states repetitively.
- Implement FPGA cache memory.
- Implement an FPADC command to read the cache memory. Verify correct operation by writing multiple patterns to memory and reading them back.
- Implement FPGA ADC, DAC and MUX I/O.
- Implement FPADC FPGA command to read single values from parallel ADCs and write single values to the parallel DAC. Connect 1st ADC to a signal generator generating a slow (1 kHz) waveform. Connect 2nd ADC to output of DAC. Implement a graphing window in control software to plot (a) the output of 1st ADC (b) output of 2nd ADC while a sine wave is generated by the DAC. Verify that the waveforms obtained represent the generated signals. Watch out for latching timing errors, which show up as incomplete sample values.

- Implement an FPADC command, which stores values of 1st or 2nd ADC to memory at high speeds. Enhance control software to plot the obtained data. Test by generating a fast (1 MHz) waveform through signal generator and DAC. Verify operation by obtaining the graphs. Watch out for timing errors, which show up as 'jagged' edges in the graphs.
- Implement the Serial I/O state machine to communicate with the serial ADC. Implement an FPADC command to communicate with the ADC. Implement control software commands to calibrate the ADC and read sampled values. Verify operation by connecting the ADC to the parallel DAC and generate a predictable pattern and sample the voltages. Check that the test results are repetitive.
- Implement the self-calibration PFPADC software, corresponding FPGA state machine and control software. Verify operation of the system by comparing the calibration tables generated by multiple calibration runs (should be very similar within show periods from each other). Perform multiple runs of the system with calibration off as well as with calibrated values.
- Implement FPADC command to control the Variable Gain Amp. Implement the control state machine in the FPGA. Test by connecting the parallel ADCs, DAC and VGA in the final configuration. Verify operation by connecting a signal generator driving a signal into the input of the FPADC. Plot the COARSE and REMAINDER ADC values versus the input signal (this will require slight changes of the memory-writing FPGA state machine). Check for DAC overdrive conditions.
- Implement FPADC commands to maintain correction tables within the FPGA. Implement control software to populate tables with values. Verify operation by plotting internal signals of the SC-DPFPADC while a signal from a signal generator is input into the system. Enable correction by writing appropriate correction tables to the FPGA. Check operation of the correcting mechanism by comparing plots of corrected versus uncorrected signals.

The only test equipment piece needed for the development is a typical signal generator.

5 Testing

5.1 Background Information

A standard method for comparing ADC performance is to calculate the difference of the obtained sampled values of a well-known signal such as a sinusoid to an ideal signal recovered through a fitting mechanism [14, 15, 16 and 17]. There are a number of different ways of analyzing the difference of the ideal fitted signal and the obtained signal described in IEEE-STD-1241 [14]. This thesis will concentrate on graphical methods such as ones used in third-party ADC evaluation software [13]. The graphical comparison methods of interest are:

- Residuals Plot [15]
- Modulo-Time Plot [16]
- Power Spectral Distribution Function [17]

The above methods are described in more detail in Sections 5.1.1, 5.1.2 and 5.1.3.

The fitting algorithms used in the program are described in [14], and are:

- 3-parameter, known-frequency
- 4-parameter, least-squares fit

There are a number of other non-graphical methods used to evaluate the performance of ADCs. They are:

- Discrete Fourier Transform methods, used to calculate Total Harmonic Distortion, Total Spurious Distortion, Spurious-Free Dynamic Range and Signal-to-Non-Harmonic Ratio.
- Nonlinearity Transfer Curve methods, which are used to locate the code transitions. This method can also be used to calculate Integral Non-Linearity and Differential Non-Linearity.
- Step Response methods, which can be used to calculate the Frequency Response of the DUT through DFT methods

The testing of the FPADC is performed using a version of the ADC Test Data Evaluation Program for MATLAB [13]. The program analyzes ADC samples of a sine signal using the algorithms suggested by IEEE-STD-1241 [14] for ADC testing. Even though the program is mainly designed for testing of linear ADCs, it is adequate for FPADC testing due to the graphical nature of the tests.

5.1.1 Residuals Plot

The Residuals Plot is the first of the graphical methods used.

The Residuals Plot is a display of the differences between the original signal and the fitted version versus sampling time. For an ideal converter this is the quantization noise, although in the case of a real ADC the Residuals are the sum of all noises of the system and the quantization noise [15].

The Residuals Plot is used for a rough estimation of the input range coverage of the signal.

5.1.2 Modulo-Time Plot

The Mod-T Plot is another of the graphical methods used.

The Mod-T plot is related to the Residuals Plot, but instead of plotting the differences against time they are plotted against the phase of the fitted signal [16].

The Mod-T Plot offers a useful way of determining the correlation between the quantization error of the ADC and the input signal. Harmonic distortions can be identified through patterns correlated with the base harmonic. Jitter can be identified by noise in the Mod-T plot which is proportional to the 1^{st} derivative of the signal. Missing codes can be identified by non-uniform distribution of the residual signal.

5.1.3 Power Spectral Distribution Function

The PSDF Plot is the last of the graphical methods used.

The PSDF displays the power spectral distribution function of the residual information. It is the integral of the power density function. For an ideal linear ADC, where the quantization noise is constant and the power spectral density of the noise is constant, the PSDF returns a linear plot. The PSDF Plot is mainly used to identify harmonics introduced during the sampling process. The harmonic and spurious distortions can be identified as discontinuities in the PSDF plot [17].

5.2 Tested Parameters

The test of the FPADC looks at a number of system properties discovered through the testing in order to evaluate the robustness of a certain configuration of the system.

The parameters that are to be tested are as follows:

- Linearity of the system
- Gain mismatch between the different conversion domains
- Amplifier offset between for the different conversion domains
- Conversion ratios between the steps of the DAC and the ADCs

The tests performed during the testing stage are qualitative in nature due to the graphical properties of the output of the test. However, we can establish a set of quantitative qualities which will give a firm indication of the performance of the system.

The measured parameter is the Mod-T graph range for each of the conversion domains. More specifically, the maximum and minimum value is measured for each conversion domain. The difference of the values represents the converter error, which ideally should be ±½ LSB. Assuming that the gains for each of the conversion domains are consecutive powers of two, the error values will also follow the same exponential relationship.

5.3 Test Setup

The test setup consists of the following equipment:

- Signal Generator capable of generating 2V p-p SINE signal at 1-100kHz
- FPADC Hardware (Stratix baseboard and FPADC daughtercard)

- Personal Computer running FPADC software and ADCTEST software

The test setup is presented in Figure 35.

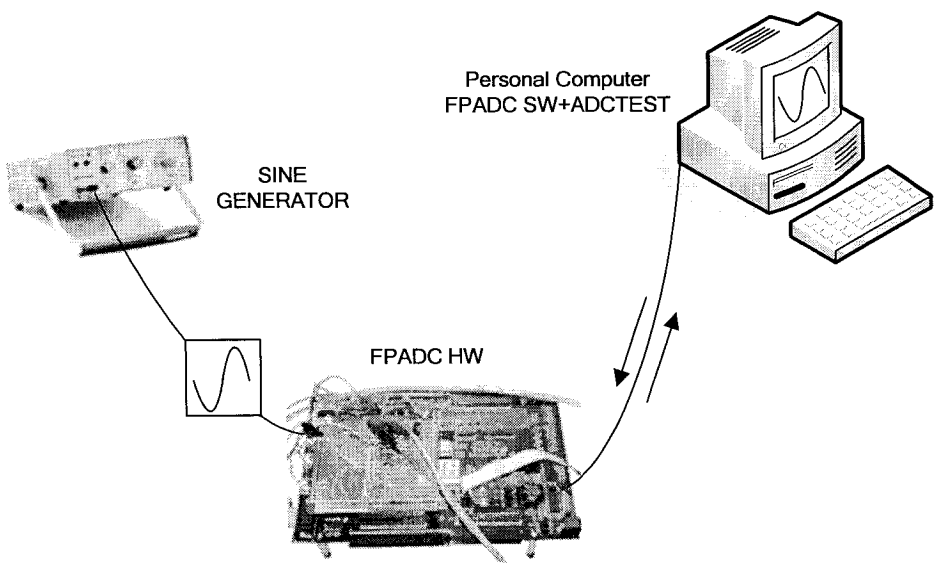


Figure 35 – Control Application to FPADC FPGA Communication Sequence

The Sine Generator generates a sinusoidal test signal which is then sampled by the FPADC. The FPADC software saves the samples in a file readable by the ADCTEST program. Next, the ADCTEST program analyzes the signal and displays the analysis results.

5.4 Test Methodology

The tests of the SC-FPADC are performed to verify that the SC-FPADC provides improvements over a Linear ADC as well as an uncalibrated FPADC. More specifically, the following results are collected:

- Ideal simulated Linear ADC signal synthesized using Matlab: This test establishes the ideal characteristics of a linear ADC, and demonstrates the three plots for an ideal device.

- Values obtained from a Linear ADC using the FPADC system: This test will establish the base noise and linearity of the system, as the test signal and the circuitry excluding the VGA and REMAINDER ADC is not expected to be ideal.
- Ideal simulated Floating Point ADC signal synthesized using Matlab: This test establishes the ideal characteristics for a FPADC, and demonstrates the three plots for an ideal device for a FPADC.
- Values obtained from the FPADC after calibration is performed: A full calibration cycle is performed in this test prior to quantization. This test shows the benefits of using the calibration algorithm over the non-calibrated FPADC and the linear ADC tested earlier. Ideally, a distribution of possible values of components used in the FPADC implementation has to be collected for a complete comparison.
- Values obtained from an uncalibrated FPADC: Ideal expected values of the correction parameters are used in this test. This test shows the inaccuracies which are exhibited by non-calibrating FPADCs. The expected values are purely theoretical.

In each of the above cases the residual plots described in Section 5.1 are used to distinguish the negative and positive characteristics. The test signal is a 1V p-p 500Hz sinusoid sampled at 500,000 samples/sec.

The following speculations are made about the outcome of the test:

- A Linear ADC exhibits lesser transfer characteristics than the FPADC
- An uncalibrated FPADC has worse transfer characteristics than a calibrated FPADC

5.5 ADCTEST Matlab Software

The testing of the FPADC is performed using a version of the ADC Test Data Evaluation Program for MATLAB [13]. The program allows for testing ADCs using the algorithms suggested by IEEE-STD-1241 [14] for ADC testing. More specifically, the program implements three and four parameter sine-wave fit algorithms [15, 16 and 17].

The program interfaces with the user via a GUI interface. The program's input is a text file containing quantized values which is obtained by saving the sampled information from the FPADC Control Program, described in Section 4.4.

The interface available to the user in the ADC Test program is presented in Figure 36.

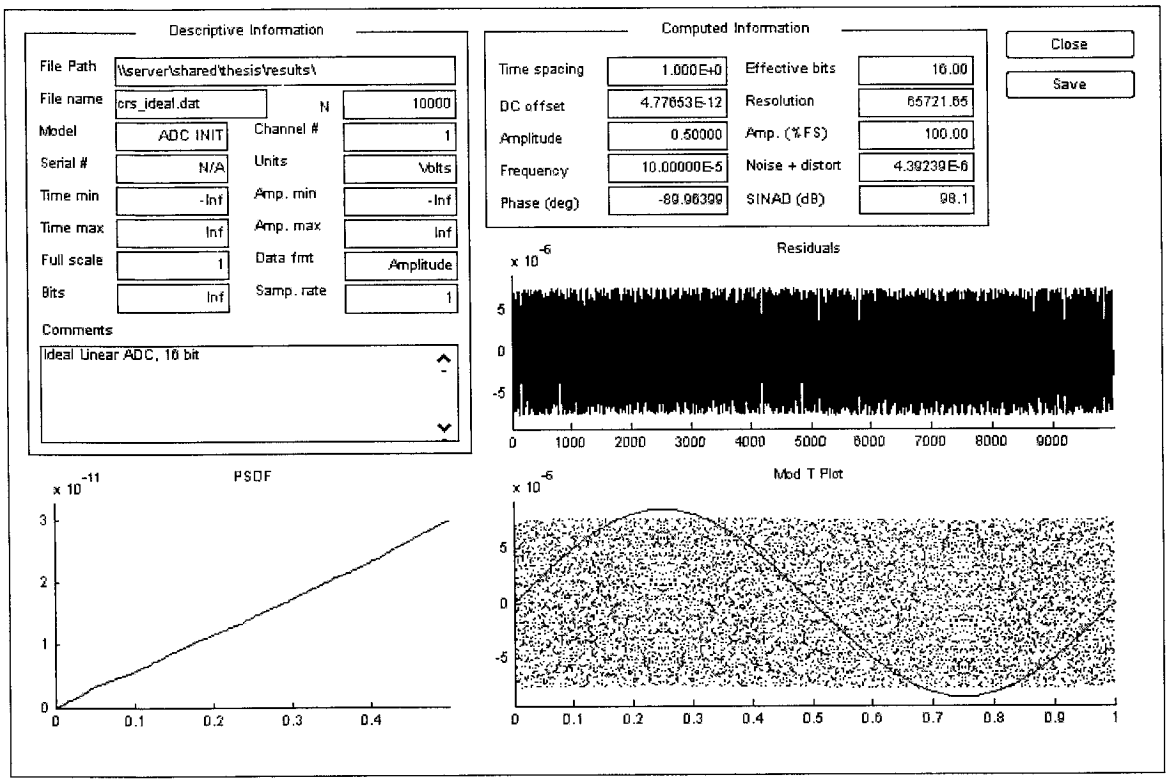


Figure 36 – Example of ADC Analysis Using ADCTEST Program

The three plots are described in Section 5.1

5.6 Ideal Linear ADC Test

This baseline test is performed using an ideal ADC signal generated by the Matlab program *gen_ideal_crsadc.m*. The program generates a signal equivalent to an ideal 6 bit Linear ADC. The test results are presented in Figure 37.

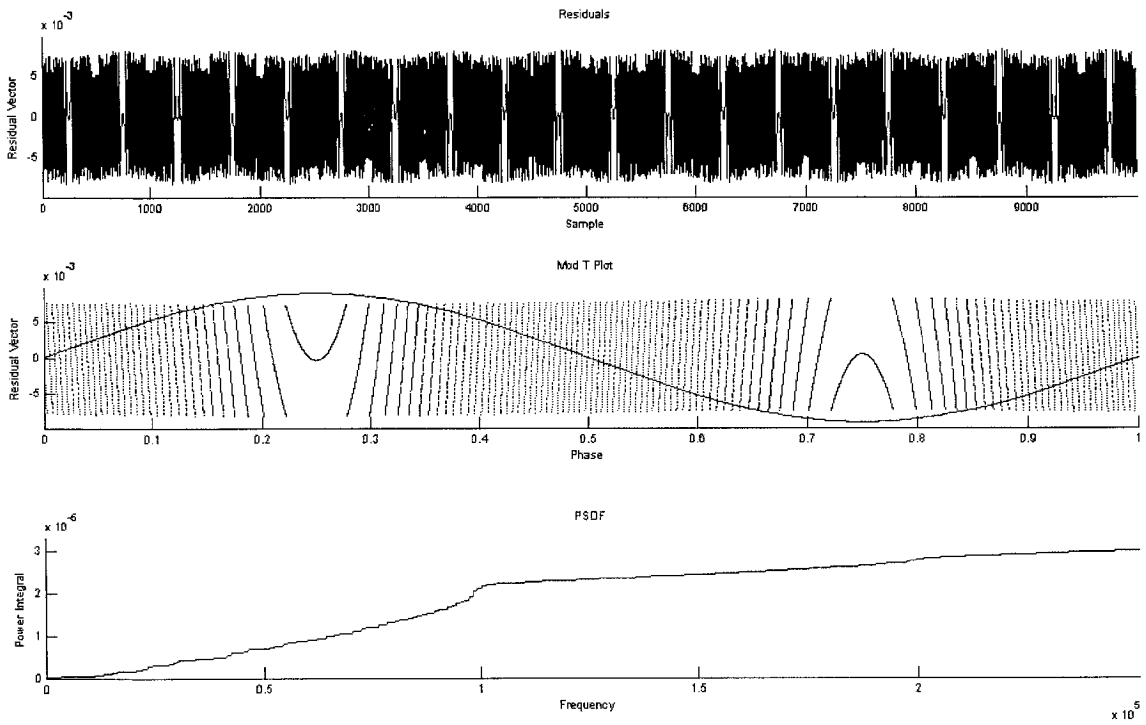


Figure 37 – Test Results for an Ideal Linear ADC

The following conclusions can be drawn from the test results:

- The signal exhibits good linearity: The Residual Vector plot has constant residual vector range for different phases
- The residual vector range is well within reason: The plot spans from $-7.5*10^{-3}$ to $+7.5*10^{-3}$ for the total range of $1.5*10^{-2}$ which is the size of the quantization step ($\frac{1}{2^6} = 1.5625*10^{-2}$)
- The PSDF plot exhibits little harmonics except for the 2nd harmonics (1000Hz): An edge is seen at 1000Hz

5.7 Linear ADC Test

This test is performed using the linear ADC used in the Coarse ADC in the FPADC. The ADC is set to 6 bit resolution. The test results are presented in Figure 38.

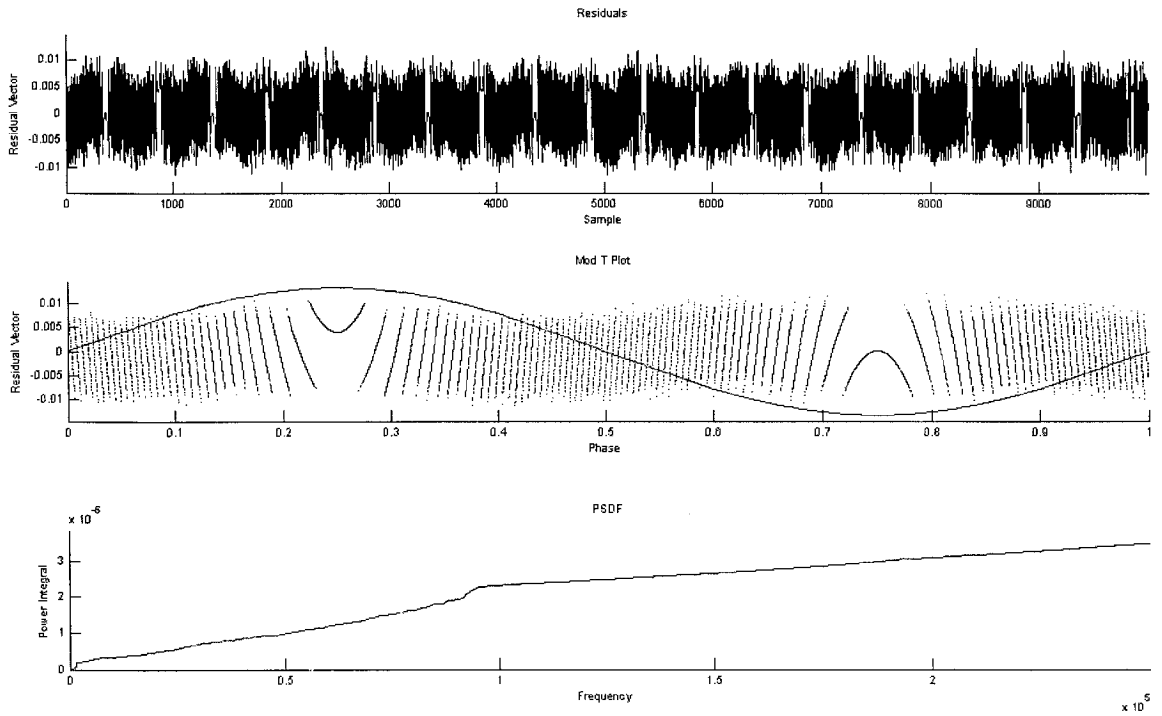


Figure 38 – Test Results for a Linear ADC

The following conclusions can be drawn from the test results:

- The ADC has a slight step non-linearity: This can be observed by looking at the Residuals and Mod-T Plot. The plots exhibit a slight 'waviness' on the Residuals Plot for samples 1000 to 2000, 4000 to 7000 and 8000 to 10000 on the Residuals plot. One the Mod-T plot, it is visible for phases of 0.1 to 0.2, 0.3 to 0.7 and 0.8 to 1.0
- The residual vector range is within reason: The plot spans from $-1.0*10^{-2}$ to $+1.0*10^{-2}$ for the total range of $2.0*10^{-2}$ which is close to the size of the quantization step ($\frac{1}{2^6} = 1.5625*10^{-2}$). The larger than ideal variation can be contributed to the non-linearity and slight internal noise of the system.
- The PSDF plot exhibits little harmonics except for the 2^{nd} harmonics (1000Hz): An edge is seen at 1000Hz, identical to the ideal signal in Section 5.6

5.8 Ideal FPADC Test

This baseline test is performed using an ideal FPADC signal generated by the Matlab program *gen_ideal_fpadc.m*. It is used to establish the plots for an ideal FPADC, and it is meant to be used as a relative comparison with the actual sampled ADC signals. The program generates a signal equivalent to a 6 bit FPADC with 4 conversion domains. The test results are featured in Figure 39.

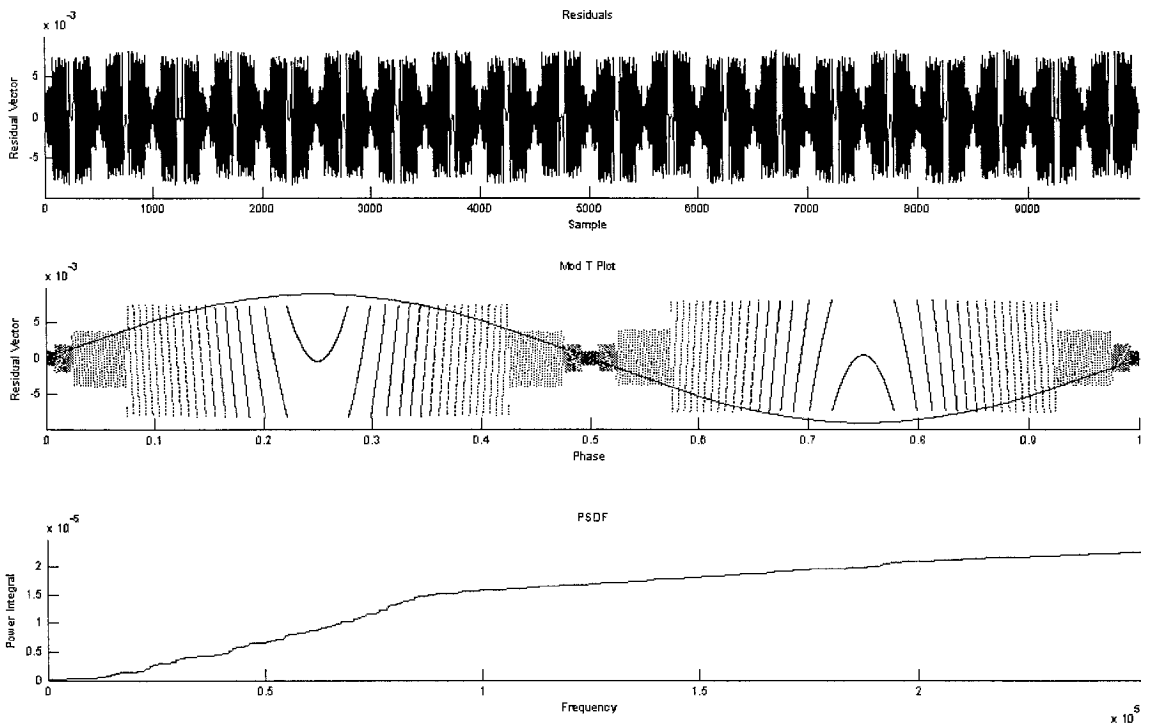


Figure 39 – Test Results for an Ideal Floating-Point ADC

The following conclusions can be drawn from the test results:

- The signal exhibits good linearity: The Mod-T plot has constant residual vector ranges for each of the conversion domains
- The residual vector range is well within reason: The respective residual vector ranges for each of the conversion domains are $1.6*10^{-3}$, $3.2*10^{-3}$, $7.5*10^{-3}$ and $1.5*10^{-2}$ which are the values of the quantization steps for the respective

conversion domains ($\frac{1}{2^9}$ = 1.953125*10^{-3}, $\frac{1}{2^8}$ = 3.90625*10^{-3}, $\frac{1}{2^7}$ = 7.8125*10^{-3}, $\frac{1}{2^6}$ = 1.5625*10^{-2})

- The PSDF plot exhibits little harmonics except for the 2nd harmonics (1000Hz): An edge is seen at 1000Hz on the PSDF plot

5.9 Calibrated FPADC Test

This test is performed using the FPADC after a calibration cycle has been issued. The FPADC is set to 6 bit resolution and employs 4 conversion domains. The test results are presented in Figure 40.

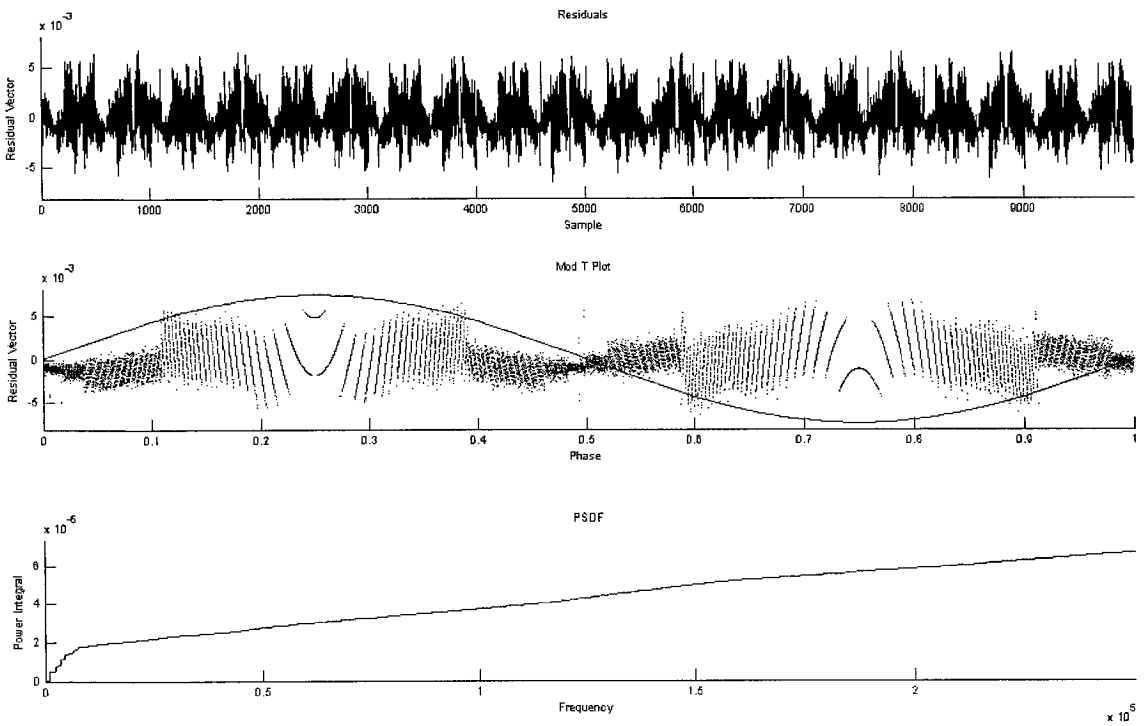


Figure 40 – Test Results for a Calibrated FPADC

The following conclusions can be drawn from the test results:

- A slight offset mismatch can be seen in the Mod-T plot for the different conversion domains: It is exhibited by the residuals for different conversion domains not being symmetric around the zero-point residual

- The signal exhibits fairly good linearity: The Mod-T plot has slightly variable residual vector ranges for each of the conversion domains
- The residual vector range is smaller than in the Ideal FPADC: The respective residual vector ranges for each of the conversion domains are $1.0*10^{-3}$, $2.0*10^{-3}$, $4.0*10^{-3}$ and $1.0*10^{-2}$ which are the respective sizes of the quantization steps (compared to respective ideal values of $\frac{1}{2^9} = 1.953125*10^{-3}$, $\frac{1}{2^8} = 3.90625*10^{-3}$, $\frac{1}{2^7} = 7.8125*10^{-3}$, $\frac{1}{2^6} = 1.5625*10^{-2}$ from Section 5.8)
- The PSDF plot exhibits some harmonics: An edge is seen at 100Hz on the PSDF plot

5.10 Uncalibrated FPADC Test

This test is performed using the FPADC. The FPADC is set to 6 bit resolution and employs 4 conversion domains. The system has not been calibrated and all calibration values are set to the expected ideal values.

The following assumptions are made:

- The respective VGA gains are 8,4,2 and 1
- The respective VGA offsets for all conversion domains are 0

The test results are presented in Figure 41.

The following conclusions can be drawn from the test results:

- A large gain and offset mismatch effect can be clearly seen in the Mod-T plot and the Residuals Plot for the different conversion domains. It is exhibited by the 'sloping' of the residuals on the phase axis of the Mod-T plot, and 'sloping' and large variations between residuals on the Residuals Plot
- The signal exhibits low linearity: The Mod-T plot has highly variable residual vector ranges for each of the conversion domains
- The residual vector range is much larger than that of the Calibrated FPADC: The respective residual vector ranges for each of the conversion domains are $1.5*10^{-3}$,

$3.0*10^{-3}$, $1.0*10^{-2}$ and $1.5*10^{-2}$ which are the respective sizes of the quantization steps (compared to respective values of $1.0*10^{-3}$, $2.0*10^{-3}$, $4.0*10^{-3}$ and $1.0*10^{-2}$ from Section 5.9)

- The PSDF plot exhibits high harmonics: A large edge is seen at 100Hz on the PSDF plot, and variation of the PSDF the for frequencies above 100Hz is low

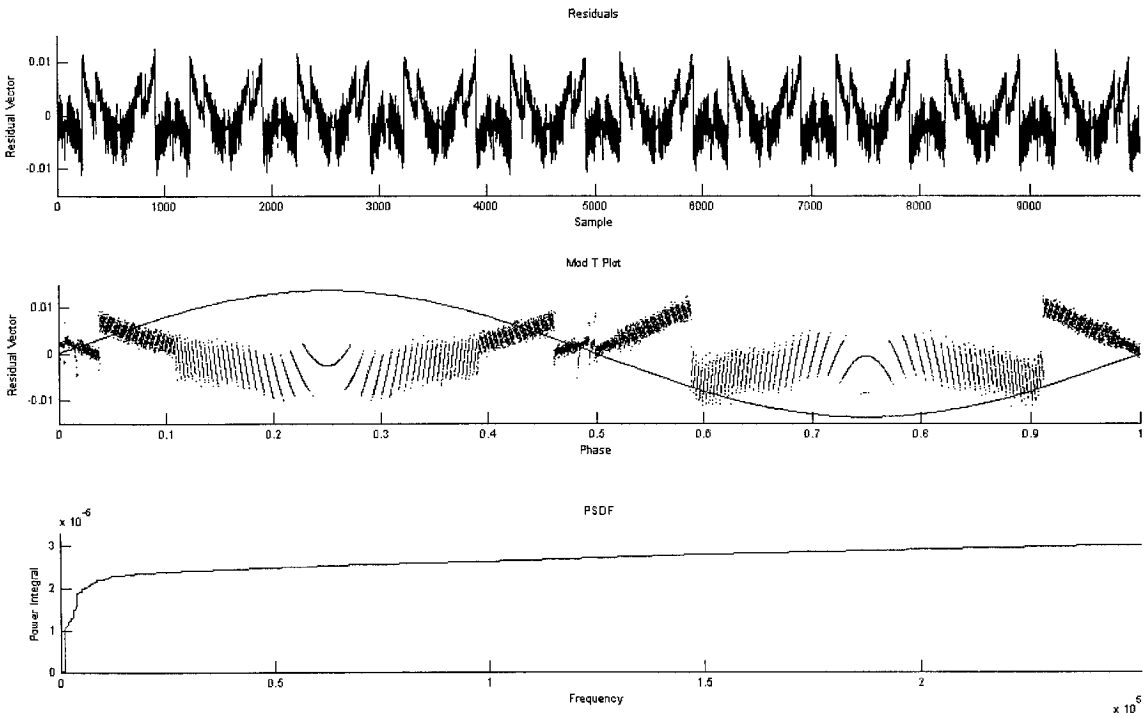


Figure 41 – Test Results for an Uncalibrated FPADC

5.11 Comparison of the Results

The following properties have been observed during the testing:

- The FPADC system has a slight non-linearity, as measured by the Linear DAC in Section 5.7.
- The measured non-linearity contributes to the final non-linearity of the FPADC system.
- The measured characteristics of the calibrated FPADC indicate that the system is not ideal.

- However, when compared to an uncalibrated FPADC, the calibrated FPADC shows large improvements is signal quality.

The non-ideal results are speculated to be due to the following reasons:

- Internal system noise and non-linearities
- Inaccurate calculation of the calibration parameters. It is speculated that the source of the inaccuracies is the CALIBRATION ADC which claims to be a 24 bit device. However, it was found that the ADC can exhibit different DC offsets in its measurements after different powerup cycles. This behavior could be contributed to the internal system noise. It is speculated that the CALIBRATION ADC's internal powerup calibrations are not working ideally in the FPADC circuit

5.12 Conclusion

A concept FPADC is presented, designed, implemented and tested in the thesis along with a FPADC prototyping system. Instead of a simple simulation the described system is implemented using real-world hardware. A considerable effort and time has been put into the research associated with design, implementation and testing, as one stumbles onto numerous problems when dealing with mixed-signal component implementations. Countless hours have been spent tweaking and optimizing the design.

It is believed that the proposed high-precision Low-Cost Self-Calibrating Floating Point Analog-to-Digital Converter proof-of-concept has been attested in this thesis. The gathered test results indicate that the system exhibits the improved characteristics of a FPADC. The resulting quantizing resolution is higher for smaller input signals, which in turn maximizes the SNR of the system. The findings also reveal the weaknesses of the FPADC and an avenue of future improvements.

In the current implementation there are problems with system noise and linearity. This is however expected from a mixed-signal circuit constructed on a prototyping breadboard without performing extensive signal path simulations. It has been shown nevertheless that

the FPADC and the associated calibration algorithm do improve the signal quality when compared to a Linear FPADC.

6 Summary and Future Improvements

6.1 Summary

The goal of this thesis is to develop, implement and test the Self-Calibrating FPADC. To achieve that goal, a number of various sub-goals are met. First, the concept and the design of a novel Self-Calibrating FPADC are outlined. Next, a real-world hardware and software implementation is performed. Then, a comparison of the newly designed SCFPADC against a linear ADC and an uncalibrated SCFPADC are performed and analyzed.

Chapter one presents the key concepts of Floating-Point Analog-to-Digital converters. The chapter also provides an introduction to the structure of the thesis.

Chapter two of the thesis presents the proposed Self-Calibrating FPADC. The parameters and the equations governing the new system are described in order to establish common terms and variables by which the system is later described.

Chapter three reveals the design of the FPADC. The chapter introduces the proposed FPADC algorithms.

Chapter four presents the implementation of the FPADC. A hardware development platform is used along with the additional board in FPADC hardware implementation. The chapter describes the internal architectures of the FPGA and software components of the FPADC. The chapter also presents testing and break-in procedures derived, aiming to inform the reader about the obstacles that were overcome during the prototyping stage.

Chapter five discusses the testing of the new FPADC. ADC evaluation software is used to derive the characteristics of the FPADC. The results are then presented in comparison to a standard linear ADC. The chapter also provides a conclusion of the results and the shortcomings discovered through the design, development and testing of the FPADC.

Chapter six is the summary of the thesis and the proposed future work.

6.2 Contribution of the thesis

The thesis makes the following contributions:

- Proposing of a novel Self-Calibrating FPADC concept
- Design of the FPADC
- Implementation of the FPADC
- Evaluation of the FPADC

This dissertation has shown that the novel design of Low-Cost, Self-Calibrating Floating Point ADC is viable and provides an increased overall accuracy when compared to a Linear ADC and an uncalibrated FPADC. A number of possible improvements and shortcomings are discovered and presented in the thesis.

6.3 Future work

Several topics can be pursued in the future in order to expand and improve the research on the FPADC described in this thesis. The ultimate goal is to create a stand-alone device which exhibits lower noise and better signal characteristics.

The improvements are listed in the following sections.

6.3.1 Full Predictor Implementation

The presented FPADC does not estimate the anticipated future values of sample values. This feature is needed in order to implement the parallel-sampling FPADC which will allow of doubling of the sampling rates. The implementation of the advanced prediction algorithms in the predictor section in the FPGA will allow for a parallel-sampling FPADC implementation.

6.3.2 Out-of-Range Checking

The FPADC currently does not check whether the REMAINDER ADC is operating with an input outside of its allowable range. This feature is also needed for a true parallel sampling FPADC. The implementation of out-of-range Remainder ADC signal detection

algorithm in the FPGA will allow for a more robust handling of fast-changing signals when the full predictor described in Section 6.3.1 is implemented.

6.3.3 FPGA-Only Implementation

The current FPADC implementation is in both software and the FPADC. This is a great solution for prototyping. However, it is somewhat impractical for embedded uses when there is no spare computing power available. Implementation of the correction and calibration algorithms entirely in the FPGA will allow for the FPADC to be a more practical stand-alone solution.

6.3.4 Conversion Domain Hysteresis

The FPADC exhibits a problem where the conversion domain range fluctuates rapidly between two values when the input signal crosses the conversion domain boundary back and forth due to the signal noise. In order to counteract this, an implementation of a hysteresis algorithm is needed to avoid the rapid switching because of input noise.

6.3.5 Fully Differential Implementation

The signal noise is a problem in the FPADC. This is due to the single-ended design of the mixed-component circuitry on the FPADC daughtercard. An implementation of full differential analog circuitry will be beneficial for the FPADC as the noise imposed by the system will be minimized due to the noise-canceling characteristics of differential designs.

6.3.6 Multiple Calibration Passes

It was found through performing multiple calibration cycles that the obtained calibration values differ slightly for each run. It is speculated that the variances are due to the noise issues described in Sections 6.3.4 and 6.3.5. Implementing multiple calibration passes and averaging of the calibration results in the correction stage will minimize the effects of the speculated prevailing high-frequency system noise.

6.3.7 More Accurate Multiplicant Calculation

This calculation is performed when the REMAINDER ADC is operating at the full resolution in order to obtain the most accurate results. If the ADC is to be operated at 4 bits, a different algorithm has to be used. The addition of this algorithm will make it possible for the FPADC to be built using low resolution ADCs.

7 References

[1] J. Yuan, J. Piper, "Floating-Point Analog-To-Digital Converter", Proceedings of ICECS '99, Vol. 3, p. 1385-8, September 1999

[2] J. Yuan, J. Piper, "A Delay-Balanced Binary-Weighted CMOS Amplifier Tree for a Floating-Point A/D Converter", Proceedings of the 16th NORCHIP Conference, p. 131-138, November 1998

[3] K. Kondoh, K. Watanabe, "An Autoranging Switched-Capacitor Analog-to-Digital Converter", IEEE Transactions on Instrumentation and Measurement, Vol. 1M-36, No. 4, p. 879, December 1987

[4] D. Thompson, B. Wooley, "A 15-bit Pipelined Floating-Point A/D Converter", 1999 European Solid-State Circuits Conference Proceedings, Duisburg, Germany, p. 170-173, September 1999

[5] G. Chiorboli, A. Boni, G. Franco, "Test of Subranging Converters with Digital Correction", IEEE Transactions on Instrumentation and Measurement, Vol. 46, No. 4, p. 975, August 1997

[6] F. Chen, C. Chen, "A 20-b Dynamic-Range Floating-Point Data Acquisition System", IEEE Transactions on Industrial Electronics, Vol. 38, No. 1, p. 10, February 1991

[7] K. Kalliojarvi, J. Kontro, Y. Neuvo, "Novel Floating-Point A/D- and D/A-Conversion Methods", Proceedings of IEEE International Symposium on Circuits and Systems, Vol. 2, p. 14, 1994

[8] J. Yuan, J. Piper, "Realization of a Floating-Point A/D Converter", Proceedings of IEEE International Symposium on Circuits and Systems, Vol. 1, p. 404-407, May 2001

[9] V. Groza, "Floating-Point Data Acquisition System with Improved Noise immunity", IEEE Instrumentation and Measurement Technology Conference, Vail, CO, USA, May 2002

[10] V. Groza, "Floating-Point ADC Optimized for Acquisition of Deterministic Signals", IEEE Instrumentation and Measurement Technology Conference, Anchorage, AK, USA, May 2002

[11] V. Groza, B. Dzerdz, "Differential Predictive Floating-Point Analog-to-Digital Converter", SITE, University of Ottawa, 2002

[12] Altera Corporation, "Stratix EP1S80 DSP Development Board Data Sheet", http://www.altera.com/literature/ds/ds_stratix_dsp_bd_pro.pdf

[13] János Márkus, "ADC Test Data Evaluation Program for MATLAB (http://www.mit.bme.hu/projects/adctest/)", Department of Measurement and Information Systems, Budapest University of Technology and Economics

[14] IEEE Std 1241-2000, "Standard for terminology and test methods for analog-to-digital converters", IEEE, Piscataway, NJ, USA, 2001

[15] I. Koll´ar, "Improved determination of the best fitting sine wave in ADC testing", Proc. of the 21st IEEE Instr. and Meas. Technology Conference, IMTC/2004, Como, Italy, May 18–20 2004, vol. 1, p. N/A, Accepted for publication

[16] F. H. Irons and D. M. Hummels, "The modulo time plot – a useful data acquisition diagnostic tool", IEEE Trans. on Instrumentation and Measurement, vol. 45, no. 3, pp. 734–738, June 1996

[17] J. J. Blair, "A method for characterizing waveform recorder errors using the power spectral distribution", IEEE Trans. on Instrumentation and Measurement, vol. 41, no. 5, pp. 604–610, October 1992

[18] IEEE Std 754-1985, "IEEE Standard for Binary Floating-Point Arithmetic", 1985

[19] V. Groza, "High Resolution Floating-Point ADC", IEEE Trans. on Instrumentation and Measurement, vol. 50, no. 6, pp. 1822–1829, December 2001

[20] Le Jin, Kumar L Parthasarathy, Degang Chen and Randall Geiger, "A Blind Identification Approach to Digital Calibration of Analog-to-Digital Converters for Built-In Self-Test", ACM Transactions on Design Automation of Electronic Systems, Volume 8, Issue 4, pp. 522 – 545, 2003

[21] Jan Gert Cauwenberghs, Member, Gabor C. Temes, "Adaptive Digital Correction of Analog Errors in MASH ADC's—Part I: Off-Line and Blind On-Line Calibration", IEEE Transactions on Circuit and Systems: Analog and Digital Signal Processing", vol. 47, no. 7, July 2000

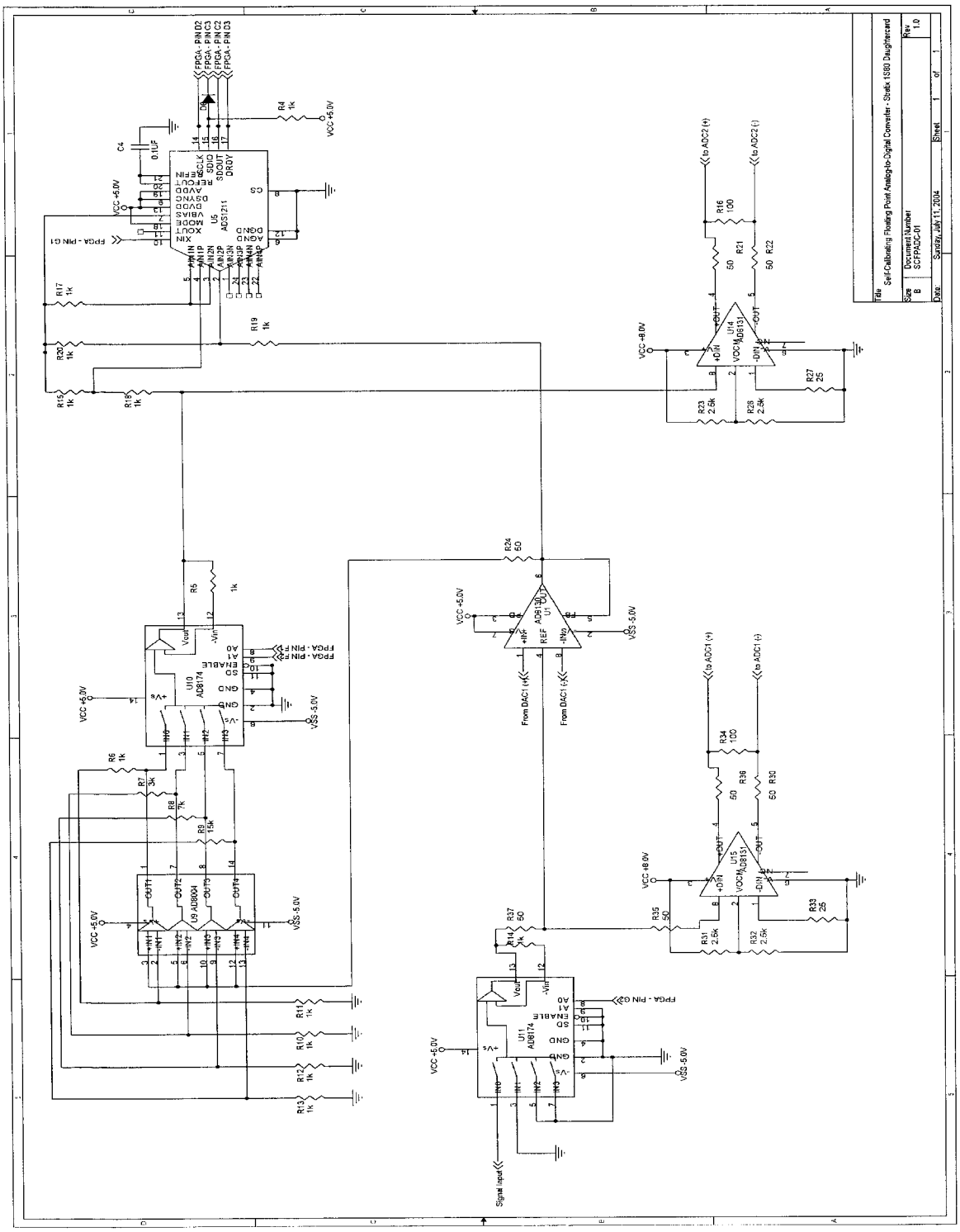
[22] V. Groza, M. Debski, D. Ionescu, "Design and Implementation of a Self-Calibrating Floating Point Analog-to-Digital Converter", IMTC 2004 IEEE Instrumentation and Measurement Technology Conference, May 2004

[23] V. Groza, M. Debski, "Self-Calibrating Differential Predictive Floating Point Analog-to-Digital Converter", IEEE Instrumentation and Measurement Technology Conference IMTC 2003, pp. 1604-1606, May 2003

[24] T. Rahkonen, "Error Correction Techniques in High-Speed A/D and D/A Converters", Dept. of Electrical Engineering and Infotech Oulu, University of Oulu, 2001

Appendix A - Schematic and Bill of Materials

Schematic



Bill of Materials

Self-Calibrating Floating Point Analog-to-Digital Converter - Stratix 1S80 Daughtercard

Item	Quantity	Reference	Part
1	1	C4	0.1UF
2	1	D8	MV209
3	13	R4,R5,R6,R10,R11,R12,R13, R14,R15,R17,R18,R19,R20	1k
4	1	R7	3k
5	1	R8	7k
6	1	R9	15k
7	2	R16,R34	100
8	7	R21,R22,R24,R30,R35,R36, R37	50
9	4	R23,R26,R31,R32	2.5k
10	2	R27,R33	25
11	1	U1	AD8130
12	1	U5	ADS1211
13	1	U9	AD8004
14	2	U10,U11	AD8174
15	2	U14,U15	AD8131

Appendix B - Source Code

Matlab Simulation

Ideal Linear ADC Generator – gen_ideal_crsadc.m

```
x = 1:1:10000;
y = sin((x/1000.77777)*2*pi)/2;
o = y;

step=1/(2^6);
for k=1:1:length(x)
   y(k)=floor(0.5+(y(k)/step))*step;
end;

figure;
plot(x,y,x,o);

disp('Ideal ADC Generator');
fid=fopen('\\server\shared\thesis\results\crs_ideal.dat','w');
fprintf(fid,'%10.10e\n',y);
fclose(fid);
```

Ideal FPADC Generator – gen_ideal_fpadc.m

```
x = 1:1:10000;
y = sin((x/1000.7777)*2*pi)/2;

for k=1:1:10000
   if abs(y(k))>0.225
      step=1/(2^6);
   elseif abs(y(k))>0.075
      step=1/(2^7);
   elseif abs(y(k))>0.025
      step=1/(2^8);
   else
      step=1/(2^9);
   end;


   y(k)=floor(0.5+(y(k)/step))*step;
end;

figure;
plot(x,y);

disp('Ideal FPADC Generator');
fid=fopen('\\server\shared\thesis\results\fp_ideal.dat','w');
fprintf(fid,'%10.10e\n',y);
fclose(fid);
```

Hardware – FPGA Sourcecode

Organization

The FPGA source code is organized as follows:

```
|-- MFC4D.tmp
|-- cmp_state.ini
|-- debug.fsf
|-- fpadc_adc.vhd
|-- fpadc_altera.asm.rpt
|-- fpadc_altera.cdf
|-- fpadc_altera.csf
|-- fpadc_altera.done
|-- fpadc_altera.eco
|-- fpadc_altera.fit.eqn
|-- fpadc_altera.fit.rpt
|-- fpadc_altera.hexout
|-- fpadc_altera.map.eqn
|-- fpadc_altera.map.rpt
|-- fpadc_altera.pin
|-- fpadc_altera.pof
|-- fpadc_altera.psf
|-- fpadc_altera.quartus
|-- fpadc_altera.qws
|-- fpadc_altera.sim.rpt
|-- fpadc_altera.sof
|-- fpadc_altera.ssf
|-- fpadc_altera.tan.rpt
|-- fpadc_dac.vhd
|-- fpadc_func.sim.rpt
|-- fpadc_func.ssf
|-- fpadc_led.acf
|-- fpadc_led.hif
|-- fpadc_led.vhd
|-- fpadc_main.acf
|-- fpadc_main.hif
|-- fpadc_main.vhd
|-- fpadc_pktprocessor.acf
|-- fpadc_pktprocessor.hif
|-- fpadc_pktprocessor.vhd
|-- fpadc_rs232.sim.rpt
|-- fpadc_rs232.ssf
|-- fpadc_rs232.vhd
|-- fpadc_rs232.vwf
|-- fpadc_sio.vhd
|-- fpadc_top.acf
|-- fpadc_top.bsf
|-- fpadc_top.hif
|-- fpadc_top.vhd
|-- release.fsf
|-- serv_req_info.txt
`-- sopc_builder_debug_log.txt
```

Main FPADC FPGA Project - fpadc_main.vhd

```
library IEEE ;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;

entity fpadc_main is
   port (
   -- standard signals
   CLK_I : in std_logic;
   RESET_I : in std_logic; -- global reset

   -- connection to fpadc module
   TXCMD_DAT_O : out std_logic_vector(255 downto 0);
   TXCMD_LEN_O : out std_logic_vector(7 downto 0);
   TXCMD_RDY_O : out std_logic; -- '1' - issue tx, '0' - idle
   TXCMD_ACK_I : in std_logic; -- '1' - tx can be issued, '0' - busy

   RXCMD_DAT_I : in std_logic_vector(255 downto 0) ;
   RXCMD_LEN_I : in std_logic_vector(7 downto 0);
   RXCMD_RDY_I : in std_logic; -- '1' - data present, '0' - idle
   RXCMD_ACK_O : out std_logic; -- '1' - rx can occur, '0' - busy
   SIO_CLK_EN_O : out std_logic; -- sio clock enable

   ADC1_DAT_I : in std_logic_vector(15 downto 0);
   ADC2_DAT_I : in std_logic_vector(15 downto 0);
   DAC1_DAT_O : out std_logic_vector(15 downto 0);
   VGA_GAIN_O : out std_logic_vector(3 downto 0);
   IN_MUX_O : out std_logic;

   ADC3_TX_DAT_O : out std_logic_vector(15 downto 0);
   ADC3_TX_RDY_O : out std_logic; -- '1' - issue tx, '0' - idle
   ADC3_TX_ACK_I : in std_logic; -- '1' - tx can be issued, '0' - busy
   ADC3_RX_DAT_I : in std_logic_vector(31 downto 0);
   ADC3_RX_RDY_I : in std_logic; -- '1' - data present, '0' - idle
   ADC3_RX_ACK_O : out std_logic; -- '1' - rx can occur, '0' - busy

   DEBUG_O : out std_logic_vector(9 downto 0);

   SCLK_I : in std_logic
   );
end fpadc_main;

architecture rtl of fpadc_main is

   component dpram_64kx4 is
   port
   (
      data        : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
      wren        : IN STD_LOGIC  := '1';
```

```
      wraddress    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
      rdaddress    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
      clock        : IN STD_LOGIC ;
      q            : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
      );
   end component;

   component dpram_64kx16 is
   port
   (
      data         : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
      wren         : IN STD_LOGIC  := '1';
      wraddress    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
      rdaddress    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
      clock        : IN STD_LOGIC ;
      q            : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
      );
   end component;

   component dpram_64kx48 is
   port
   (
      data         : IN STD_LOGIC_VECTOR (47 DOWNTO 0);
      wren         : IN STD_LOGIC  := '1';
      wraddress    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
      rdaddress    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
      clock        : IN STD_LOGIC ;
      q            : OUT STD_LOGIC_VECTOR (47 DOWNTO 0)
      );
   end component;

   component dpram_64kx64 is
   port
   (
      data         : IN STD_LOGIC_VECTOR (63 DOWNTO 0);
      wren         : IN STD_LOGIC  := '1';
      wraddress    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
      rdaddress    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
      clock        : IN STD_LOGIC ;
      q            : OUT STD_LOGIC_VECTOR (63 DOWNTO 0)
      );
   end component;

   signal FAST_SAMPLE_MODE_INT : std_logic; -- fast sampling/store mode
   signal CLOCK_DIV_INT : std_logic_vector(15 downto 0); --clock
divider
   signal MASK_ADC_INT : std_logic_vector(15 downto 0);
   signal MASK_DAC_INT : std_logic_vector(15 downto 0);
   signal LUT_SHIFT_INT : std_logic_vector(15 downto 0);
   signal VGA_GAIN_INT : std_logic_vector(3 downto 0);
   signal MEM_RDADR_INT : std_logic_vector(15 downto 0);
   signal MEM_RDLEN_INT : std_logic_vector(15 downto 0);
```

```
   signal MEM_WRADR_INT : std_logic_vector(15 downto 0);
   signal MEM_WRLEN_INT : std_logic_vector(15 downto 0);
   signal DAC1_DAT_INT : std_logic_vector(15 downto 0);
   signal ADC1_DAT_INT : std_logic_vector(15 downto 0);
   signal ADC2_DAT_INT : std_logic_vector(15 downto 0);
   signal MEM_ADDR_COPY_BUSY_INT : bit_vector(2 downto 0);
   signal DEBUG_INT : std_logic_vector(9 downto 0);
   -- lut signals
   signal GAIN_RAM_DIN_INT : std_logic_vector(3 downto 0);
   signal GAIN_RAM_DOUT_INT : std_logic_vector(3 downto 0);
   signal GAIN_RAM_WREN_INT : std_logic;
   signal GAIN_RAM_WRADDR_INT : std_logic_vector(15 downto 0);
   signal GAIN_RAM_RDADDR_INT : std_logic_vector(15 downto 0);
   signal OFFSET_RAM_DIN_INT : std_logic_vector(15 downto 0);
   signal OFFSET_RAM_DOUT_INT : std_logic_vector(15 downto 0);
   signal OFFSET_RAM_WREN_INT : std_logic;
   signal OFFSET_RAM_WRADDR_INT : std_logic_vector(15 downto 0);
   signal OFFSET_RAM_RDADDR_INT : std_logic_vector(15 downto 0);
   signal SAMPLE_RAM_DIN_INT : std_logic_vector(63 downto 0);
   signal SAMPLE_RAM_DOUT_INT : std_logic_vector(63 downto 0);
   signal SAMPLE_RAM_WREN_INT : std_logic;
   signal SAMPLE_RAM_WRADDR_INT : std_logic_vector(15 downto 0);
   signal SAMPLE_RAM_RDADDR_INT : std_logic_vector(15 downto 0);
   signal SIO_CLK_EN1_INT : std_logic;
   signal SIO_CLK_EN2_INT : std_logic;

begin
   ------------------------------------------------------------------
   -- Predefined components
   ------------------------------------------------------------------
-

   -- lookup tables

   OFFSET_RAM : dpram_64kx16 PORT MAP (
      data => OFFSET_RAM_DIN_INT,
      wren => OFFSET_RAM_WREN_INT,
      wraddress => OFFSET_RAM_WRADDR_INT,
      rdaddress => OFFSET_RAM_RDADDR_INT,
      clock => CLK_I,
      q => OFFSET_RAM_DOUT_INT
   );

   GAIN_RAM : dpram_64kx4 PORT MAP (
      data => GAIN_RAM_DIN_INT,
      wren => GAIN_RAM_WREN_INT,
      wraddress => GAIN_RAM_WRADDR_INT,
      rdaddress => GAIN_RAM_RDADDR_INT,
      clock => CLK_I,
      q => GAIN_RAM_DOUT_INT
   );
```

```
   SAMPLE_RAM : dpram_64kx64 PORT MAP (
      data => SAMPLE_RAM_DIN_INT,
      wren => SAMPLE_RAM_WREN_INT,
      wraddress => SAMPLE_RAM_WRADDR_INT,
      rdaddress => SAMPLE_RAM_RDADDR_INT,
      clock => CLK_I,
      q => SAMPLE_RAM_DOUT_INT
   );

   ------------------------------------------------------------------------
-
   -- debug status
   ------------------------------------------------------------------------
-

   DEBUG_O <= DEBUG_INT;

   DEBUG_INT(9) <= FAST_SAMPLE_MODE_INT;
   DEBUG_INT(8) <= FAST_SAMPLE_MODE_INT;

   ------------------------------------------------------------------------
-
   -- busy markers
   ------------------------------------------------------------------------
-

   MEM_ADDR_COPY_BUSY_INT(0) <= MEM_ADDR_COPY_BUSY_INT(1) xor
      MEM_ADDR_COPY_BUSY_INT(2);
   SIO_CLK_EN_O <= SIO_CLK_EN1_INT and SIO_CLK_EN2_INT;

   ------------------------------------------------------------------------
-
   -- fast MEMORY/SAMPLE Read/Write state machine
   ------------------------------------------------------------------------
-

   process
      variable MEM_ADR_INT : std_logic_vector(15 downto 0);
      variable MEM_LEN_INT : std_logic_vector(15 downto 0);
      variable DAC1_DAT_SAVED_INT : std_logic_vector(15 downto 0);
      variable VGA_GAIN_SAVED_INT : std_logic_vector(3 downto 0);
      variable CLOCK_COUNTER_INT : std_logic_vector(15 downto 0);
      variable TICK_COUNTER_INT : std_logic_vector(7 downto 0);
   begin

      wait until Falling_Edge(SCLK_I);

      -- reset
      if (RESET_I = '1') then
         FAST_SAMPLE_MODE_INT <= '0';
         MEM_ADR_INT := x"0000";
         MEM_LEN_INT := x"0000";
         FAST_SAMPLE_MODE_INT <= '0';
```

```
      MEM_ADDR_COPY_BUSY_INT(2) <= '0';
      SAMPLE_RAM_WREN_INT <= '0';
      SIO_CLK_EN1_INT <= '1';
      CLOCK_COUNTER_INT := x"0000";
      ADC1_DAT_INT <= x"0000";
      ADC2_DAT_INT <= x"0000";
      DAC1_DAT_O <= x"0000";
      TICK_COUNTER_INT := x"00";
   else
      case TICK_COUNTER_INT is -- fine-clock state machine
         when x"00" =>
            -- get coarse ADC, read lookup table (get has to
            -- occur when COUNT_I(0) is 0 -> falling clock edge)
            ADC2_DAT_INT <= ADC2_DAT_I and MASK_ADC_INT;
            OFFSET_RAM_RDADDR_INT <= x"00" & ADC2_DAT_I(15 downto 8)
               and MASK_ADC_INT(15 downto 8);
            GAIN_RAM_RDADDR_INT <= x"00" & ADC2_DAT_I(15 downto 8)
               and MASK_ADC_INT(15 downto 8);

            TICK_COUNTER_INT := TICK_COUNTER_INT +1;

         when x"01" => -- set offset DAC and VGA gain
            -- latch out values from lookup table output
            -- full fpadc mode
            if (FAST_SAMPLE_MODE_INT = '1') then -- output mem value
               DAC1_DAT_O <= OFFSET_RAM_DOUT_INT and MASK_DAC_INT;
               VGA_GAIN_O <= not GAIN_RAM_DOUT_INT;
            else -- output user value
               VGA_GAIN_O <= not VGA_GAIN_INT;
               DAC1_DAT_O <= DAC1_DAT_INT and MASK_DAC_INT;
            end if;

            -- this belongs in the next section, but it's ok to
            -- set it here
            -- set memory write addr
            SAMPLE_RAM_WRADDR_INT <= MEM_ADR_INT;

            TICK_COUNTER_INT := TICK_COUNTER_INT +1;

            -- wait until the remainder ADC stabilizes

         when x"0e" => -- get remainder ADC, write results to
            -- memory (get has to occur when COUNT_I(0) is
            -- 0 -> falling clock edge)
            ADC1_DAT_INT <= ADC1_DAT_I and MASK_ADC_INT;

            if (FAST_SAMPLE_MODE_INT = '1' and
               CLOCK_COUNTER_INT = CLOCK_DIV_INT) then
               SAMPLE_RAM_DIN_INT(15 downto 0) <= ADC1_DAT_I
                  and MASK_ADC_INT; -- write remainder adc value
               -- write coarse adc value
               SAMPLE_RAM_DIN_INT(31 downto 16) <= ADC2_DAT_INT;
```

```
            -- write offset dac value
            SAMPLE_RAM_DIN_INT(47 downto 32) <=
               OFFSET_RAM_DOUT_INT and MASK_DAC_INT;
            -- write gain value
            SAMPLE_RAM_DIN_INT(51 downto 48) <=
               GAIN_RAM_DOUT_INT;
            SAMPLE_RAM_DIN_INT(63 downto 52) <= x"000";
            SAMPLE_RAM_WREN_INT <= '1'; -- sram write
         end if;

         TICK_COUNTER_INT := TICK_COUNTER_INT +1;

      when x"0f" => -- set up fast sample cycle
         -- end sram write
         SAMPLE_RAM_WREN_INT <= '0';

         -- check for fast sample cycle request
         if (FAST_SAMPLE_MODE_INT = '0' and
            MEM_ADDR_COPY_BUSY_INT(0) = '1') then

            SIO_CLK_EN1_INT <= '0'; -- disable SIO

            MEM_ADR_INT := MEM_WRADR_INT;
            MEM_LEN_INT := MEM_WRLEN_INT;

            MEM_ADDR_COPY_BUSY_INT(2) <=
               MEM_ADDR_COPY_BUSY_INT(1);

            -- init sample timing division counter
            CLOCK_COUNTER_INT := x"0000";
            FAST_SAMPLE_MODE_INT <= '1';
         elsif (FAST_SAMPLE_MODE_INT = '1' and
            MEM_LEN_INT /= x"0000") then
            -- DMA ENGINE Advance

            if (CLOCK_COUNTER_INT = CLOCK_DIV_INT) then
               -- advance internal sram adr
               MEM_ADR_INT := MEM_ADR_INT + 1;
               -- advance internal sram count
               MEM_LEN_INT := MEM_LEN_INT - 1;
               CLOCK_COUNTER_INT := x"0000";
            else
               CLOCK_COUNTER_INT := CLOCK_COUNTER_INT +1;
            end if;
         elsif (FAST_SAMPLE_MODE_INT = '1') then
            FAST_SAMPLE_MODE_INT <= '0'; -- ok, stop
            -- done fast sample, enable SIO clk
            SIO_CLK_EN1_INT <= '1';
         end if;

         TICK_COUNTER_INT := x"00";
```

```
            when others =>
               TICK_COUNTER_INT := TICK_COUNTER_INT +1;
         end case;
      end if;
   end process ;

   ------------------------------------------------------------------------
-
   -- cmd processing
   ------------------------------------------------------------------------
-
   process
      variable COUNTER : integer range 0 to 15;
      variable STATE : integer range 0 to 15;
      variable COMMAND : std_logic_vector(7 downto 0);
      variable WAIT4_CMD_RX : std_logic;
      variable WAIT4_CMD_TX : std_logic;
      variable WAIT4_SIO_RX : std_logic;
      variable WAIT4_SIO_TX : std_logic;
   begin
      wait until Rising_Edge(CLK_I);

      --DEBUG_O(7 downto 0) <= COMMAND;

      -- reset
      if (RESET_I = '1') then
         COUNTER := 0;
         COMMAND := x"00";
         RXCMD_ACK_O <= '0';
         TXCMD_RDY_O <= '0';
         ADC3_TX_RDY_O <= '0';
         ADC3_RX_ACK_O <= '0';
         WAIT4_CMD_RX := '0';
         WAIT4_CMD_TX := '0';
         WAIT4_SIO_RX := '0';
         WAIT4_SIO_TX := '0';
         MEM_ADDR_COPY_BUSY_INT(1) <= '0';
         OFFSET_RAM_WREN_INT <= '0';
         GAIN_RAM_WREN_INT <= '0';
         SIO_CLK_EN2_INT <= '1';
         CLOCK_DIV_INT <= x"0000";
         MASK_ADC_INT <= x"ffff";
         MASK_DAC_INT <= x"ffff";

      elsif ((WAIT4_CMD_RX or WAIT4_CMD_TX or WAIT4_SIO_RX
         or WAIT4_SIO_TX) = '1') then
         if (WAIT4_CMD_RX = '1') then -- block until rxpkt block ready
            WAIT4_CMD_RX := RXCMD_RDY_I;
            RXCMD_ACK_O <= RXCMD_RDY_I;
         end if;
         if (WAIT4_CMD_TX = '1') then -- block until txpkt block acks
            WAIT4_CMD_TX := not TXCMD_ACK_I;
```

```
      TXCMD_RDY_O <= not TXCMD_ACK_I;
   end if;
   if (WAIT4_SIO_RX = '1') then -- block until sio block ready
      WAIT4_SIO_RX := ADC3_RX_RDY_I;
      ADC3_RX_ACK_O <= ADC3_RX_RDY_I;
   end if;
   if (WAIT4_SIO_TX = '1') then -- block until sio block acks
      WAIT4_SIO_TX := not ADC3_TX_ACK_I;
      ADC3_TX_RDY_O <= not ADC3_TX_ACK_I;
   end if;
-- command in progress
elsif (COUNTER /= 0) then
   -- read serial (precision ADC) cmd
   if RXCMD_DAT_I(255 downto 248) = x"20" then
      TXCMD_LEN_O <= x"04";
      TXCMD_DAT_O(255 downto 224) <= ADC3_RX_DAT_I;
      TXCMD_RDY_O <= '1';
      WAIT4_CMD_TX := '1';
   -- fetch value read from memory
   elsif (COMMAND = x"50" or COMMAND = x"51") then
      TXCMD_LEN_O <= x"04";
      if (COMMAND = x"50") then
         TXCMD_DAT_O(255 downto 224) <=
         SAMPLE_RAM_DOUT_INT(63 downto 32);
      else
         TXCMD_DAT_O(255 downto 224) <=
         SAMPLE_RAM_DOUT_INT(31 downto 0);
      end if;
      TXCMD_RDY_O <= '1';
      WAIT4_CMD_TX := '1';
   end if;

   COUNTER := COUNTER -1;
-- new command
elsif (RXCMD_RDY_I = '1') then
   RXCMD_ACK_O <= '1';
   WAIT4_CMD_RX := '1';
   COMMAND := RXCMD_DAT_I(255 downto 248);
   DEBUG_INT(7 downto 0) <= DEBUG_INT(7 downto 0) +1;

   -- mirror input
   if (RXCMD_DAT_I(255 downto 248) = x"08") then
      TXCMD_DAT_O <= RXCMD_DAT_I(247 downto 0) & x"00";
      TXCMD_LEN_O <= RXCMD_LEN_I;
      TXCMD_RDY_O <= '1';
      WAIT4_CMD_TX := '1';

   -- read ADC1, arg 1
   elsif (RXCMD_DAT_I(255 downto 248) = x"10") then
      TXCMD_LEN_O <= x"04";
      TXCMD_DAT_O(255 downto 224) <= x"0000" & ADC1_DAT_INT;
      TXCMD_RDY_O <= '1';
```

```
      WAIT4_CMD_TX := '1';

   -- read ADC2, arg 1
   elsif (RXCMD_DAT_I(255 downto 248) = x"11") then
      TXCMD_LEN_O <= x"04";
      TXCMD_DAT_O(255 downto 224) <= x"0000" & ADC2_DAT_INT;
      TXCMD_RDY_O <= '1';
      WAIT4_CMD_TX := '1';

   -- write DAC, arg=value
   elsif (RXCMD_DAT_I(255 downto 248) = x"12") then
      DAC1_DAT_INT <= RXCMD_DAT_I(231 downto 216);

   -- send SIO command
   elsif (RXCMD_DAT_I(255 downto 248) = x"20") then
      ADC3_TX_DAT_O <= RXCMD_DAT_I(231 downto 216);
      ADC3_TX_RDY_O <= '1';
      WAIT4_SIO_TX := '1';
      WAIT4_SIO_RX := '1';
      COUNTER := 1; -- continue later

   -- write offset correction table
   elsif (RXCMD_DAT_I(255 downto 248) = x"38") then
      OFFSET_RAM_WRADDR_INT <= x"00" &
         RXCMD_DAT_I(239 downto 232);
      OFFSET_RAM_DIN_INT <= RXCMD_DAT_I(231 downto 216);
      OFFSET_RAM_WREN_INT <= '1';

   -- write gain correction table
   elsif (RXCMD_DAT_I(255 downto 248) = x"39") then
      GAIN_RAM_WRADDR_INT <= x"00" & RXCMD_DAT_I(239 downto 232);
      GAIN_RAM_DIN_INT <= RXCMD_DAT_I(219 downto 216);
      GAIN_RAM_WREN_INT <= '1';

   -- init sample cycle
   elsif (RXCMD_DAT_I(255 downto 248) = x"40") then
      -- save write addr
      MEM_WRADR_INT <= RXCMD_DAT_I(247 downto 232);
      -- save write len
      MEM_WRLEN_INT <= RXCMD_DAT_I(231 downto 216);
      MEM_ADDR_COPY_BUSY_INT(1) <= not MEM_ADDR_COPY_BUSY_INT(2);

   -- fetch value read from memory
   elsif (RXCMD_DAT_I(255 downto 248) = x"50"
      or RXCMD_DAT_I(255 downto 248) = x"51") then
      -- save read addr
      MEM_RDADR_INT <= RXCMD_DAT_I(247 downto 232);
      -- save read len
      MEM_RDLEN_INT <= RXCMD_DAT_I(231 downto 216);
      SAMPLE_RAM_RDADDR_INT <= RXCMD_DAT_I(247 downto 232);
      COUNTER := 1;
```

```
        -- set VGA gain
        elsif (RXCMD_DAT_I(255 downto 248) = x"70") then
           -- write variable gain amp, arg=gain
           VGA_GAIN_INT <= RXCMD_DAT_I(219 downto 216);

        -- set INPUT MUX state
        elsif (RXCMD_DAT_I(255 downto 248) = x"71") then
           IN_MUX_O <= RXCMD_DAT_I(216); -- write MUX state

        -- set fast sampling divider
        elsif RXCMD_DAT_I(255 downto 248) = x"a0" then
           CLOCK_DIV_INT <= RXCMD_DAT_I(231 downto 216);
        -- set clock enables
        elsif RXCMD_DAT_I(255 downto 248) = x"a1" then
           SIO_CLK_EN2_INT  <= RXCMD_DAT_I(216);
        -- set adc mask
        elsif RXCMD_DAT_I(255 downto 248) = x"a2" then
           MASK_ADC_INT <= RXCMD_DAT_I(231 downto 216);
        -- set dac mask
        elsif RXCMD_DAT_I(255 downto 248) = x"a3" then
           MASK_DAC_INT <= RXCMD_DAT_I(231 downto 216);
        -- set lut shift
        elsif RXCMD_DAT_I(255 downto 248) = x"a4" then
           LUT_SHIFT_INT <= RXCMD_DAT_I(231 downto 216);
        else
        -- mirror input
           TXCMD_DAT_O <= RXCMD_DAT_I;
           TXCMD_LEN_O <= RXCMD_LEN_I;
           TXCMD_RDY_O <= '1';
           WAIT4_CMD_TX := '1';
        end if;
      else
         -- stop all memory ops
         OFFSET_RAM_WREN_INT <= '0';
         GAIN_RAM_WREN_INT <= '0';
      end if;
   end process;

end rtl ;
```

Top FPADC FPGA Project – fpadc_top.vhd

```
library IEEE ;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;

-- this is the top level design

entity fpadc_top is
 port (
   -- pads
   -- clocks
   CLK_OSC_PIN             : in std_logic;
   CLK_OSC2_PIN            : in std_logic;
   CLK_DTOA1_STRATIX_PIN   : out std_logic;
   CLK_DTOA2_STRATIX_PIN   : out std_logic;
   CLK_SRAM1_PIN           : out std_logic;
   CLK_SRAM2_PIN           : out std_logic;
   CLK_OPT_A2D_PIN         : out std_logic;
   CLK_DEBUGA_PIN          : out std_logic;
   CLK_DEBUGB_PIN          : out std_logic;
   CLK_2p_PIN              : in std_logic;
   CLK_EVALIO_OUT44_PIN    : out std_logic;

   -- rs232
   ROUT_PIN              : in std_logic;
   TIN_PIN               : out std_logic;

   -- buttons
-- SW0_PIN               : in std_logic;
   DEV_CLRn              : in std_logic;
   SW1_PIN               : in std_logic;
   SW2_PIN               : in std_logic;

   -- switches
   SW3p1_PIN             : in std_logic;
   SW3p2_PIN             : in std_logic;
   SW3p3_PIN             : in std_logic;
   SW3p4_PIN             : in std_logic;
   SW3p5_PIN             : in std_logic;
   SW3p6_PIN             : in std_logic;
   SW3p7_PIN             : in std_logic;
   SW3p8_PIN             : in std_logic;

   -- hex led disp
   HEX_0A_PIN            : out std_logic;
   HEX_0B_PIN            : out std_logic;
   HEX_0C_PIN            : out std_logic;
   HEX_0D_PIN            : out std_logic;
   HEX_0E_PIN            : out std_logic;
```

```
HEX_0F_PIN          : out std_logic;
HEX_0G_PIN          : out std_logic;
HEX_0DP_PIN         : out std_logic;
HEX_1A_PIN          : out std_logic;
HEX_1B_PIN          : out std_logic;
HEX_1C_PIN          : out std_logic;
HEX_1D_PIN          : out std_logic;
HEX_1E_PIN          : out std_logic;
HEX_1F_PIN          : out std_logic;
HEX_1G_PIN          : out std_logic;
HEX_1DP_PIN         : out std_logic;

-- leds
LED0_PIN            : out std_logic;
LED1_PIN            : out std_logic;

-- a/d 1
ATOD1_b0_PIN        : in std_logic;
ATOD1_b1_PIN        : in std_logic;
ATOD1_b2_PIN        : in std_logic;
ATOD1_b3_PIN        : in std_logic;
ATOD1_b4_PIN        : in std_logic;
ATOD1_b5_PIN        : in std_logic;
ATOD1_b6_PIN        : in std_logic;
ATOD1_b7_PIN        : in std_logic;
ATOD1_b8_PIN        : in std_logic;
ATOD1_b9_PIN        : in std_logic;
ATOD1_b10_PIN       : in std_logic;
ATOD1_b11_PIN       : in std_logic;

-- a/d 2
ATOD2_b0_PIN        : in std_logic;
ATOD2_b1_PIN        : in std_logic;
ATOD2_b2_PIN        : in std_logic;
ATOD2_b3_PIN        : in std_logic;
ATOD2_b4_PIN        : in std_logic;
ATOD2_b5_PIN        : in std_logic;
ATOD2_b6_PIN        : in std_logic;
ATOD2_b7_PIN        : in std_logic;
ATOD2_b8_PIN        : in std_logic;
ATOD2_b9_PIN        : in std_logic;
ATOD2_b10_PIN       : in std_logic;
ATOD2_b11_PIN       : in std_logic;

-- d/a 1
DTOA1_b13_PIN       : out std_logic;
DTOA1_b12_PIN       : out std_logic;
DTOA1_b11_PIN       : out std_logic;
DTOA1_b10_PIN       : out std_logic;
DTOA1_b9_PIN        : out std_logic;
DTOA1_b8_PIN        : out std_logic;
DTOA1_b7_PIN        : out std_logic;
```

```
   DTOA1_b6_PIN          : out std_logic;
   DTOA1_b5_PIN          : out std_logic;
   DTOA1_b4_PIN          : out std_logic;
   DTOA1_b3_PIN          : out std_logic;
   DTOA1_b2_PIN          : out std_logic;
   DTOA1_b1_PIN          : out std_logic;
   DTOA1_b0_PIN          : out std_logic;

   -- d/a 2
   DTOA2_b13_PIN           : out std_logic;
   DTOA2_b12_PIN           : out std_logic;
   DTOA2_b11_PIN           : out std_logic;
   DTOA2_b10_PIN           : out std_logic;
   DTOA2_b9_PIN            : out std_logic;
   DTOA2_b8_PIN            : out std_logic;
   DTOA2_b7_PIN            : out std_logic;
   DTOA2_b6_PIN            : out std_logic;
   DTOA2_b5_PIN            : out std_logic;
   DTOA2_b4_PIN            : out std_logic;
   DTOA2_b3_PIN            : out std_logic;
   DTOA2_b2_PIN            : out std_logic;
   DTOA2_b1_PIN            : out std_logic;
   DTOA2_b0_PIN            : out std_logic;


   -- digital i/o
   EVALIO_IN0_PIN        : in std_logic;
   EVALIO_IN1_PIN        : out std_logic;
   EVALIO_IN2_PIN        : in std_logic;
   EVALIO_IN3_PIN        : in std_logic;
   EVALIO_IN4_PIN        : out std_logic;
   EVALIO_IN5_PIN        : out std_logic;
   EVALIO_IN6_PIN        : out std_logic;
   EVALIO_IN7_PIN        : out std_logic;
   EVALIO_IN8_PIN        : out std_logic;
   EVALIO_IN9_PIN        : out std_logic;

   );
end fpadc_top;

architecture rtl of fpadc_top is
component fpadc_pktprocessor is
 port (
   -- standard signals
   CLK_I : in std_logic;
   RESET_I : in std_logic; -- global reset

   -- connection to fpadc module
   RXCMD_DAT_O : out std_logic_vector(255 downto 0);
   RXCMD_LEN_O : out std_logic_vector(7 downto 0);
   RXCMD_RDY_O : out std_logic; -- '1' - data present, '0' - idle
   RXCMD_ACK_I : in std_logic; -- '1' - rx can occur, '0' - busy
```

```
    TXCMD_DAT_I : in std_logic_vector(255 downto 0);
    TXCMD_LEN_I : in std_logic_vector(7 downto 0);
    TXCMD_RDY_I : in std_logic; -- '1' - issue tx, '0' - idle
    TXCMD_ACK_O : out std_logic; -- '1' - tx can be issued, '0' - busy

    -- connection to io module
    RX_DAT_I : in std_logic_vector(7 downto 0);
    RX_RDY_I : in std_logic; -- '1' - data present, '0' - idle
    RX_ACK_O : out std_logic; -- '1' - rx can occur, '0' - busy
    TX_DAT_O : out std_logic_vector(7 downto 0);
    TX_RDY_O : out std_logic; -- '1' - issue tx, '0' - idle
    TX_ACK_I : in std_logic -- '1' - tx can be issued, '0' - busy
    );
end component;

component fpadc_rs232 is
 port (
    -- standard signals
    CLK_I : in std_logic;
    RESET_I : in std_logic; -- global reset

    -- real io
    RX_PAD_I : in std_logic;
    TX_PAD_O : out std_logic;

    -- interface to processing module
    RX_DAT_O : out std_logic_vector(7 downto 0);
    RX_RDY_O : out std_logic; -- '1' - data present, '0' - idle
    RX_ACK_I : in std_logic; -- '1' - rx can occur, '0' - busy
    TX_DAT_I : in std_logic_vector(7 downto 0);
    TX_RDY_I : in std_logic; -- '1' - issue tx, '0' - idle
    TX_ACK_O : out std_logic -- '1' - tx can be issued, '0' - busy
    );
end component;

component fpadc_hexled is
 port (
    -- standard signals
    CLK_I : in std_logic;
    RESET_I : in std_logic; -- global reset

    -- real io
    HEXLED_O : out std_logic_vector(6 downto 0);

    -- interface to processing module
    DATA_I : in std_logic_vector(3 downto 0)
    );
end component;

component fpadc_main is
 port (
```

```
    -- standard signals
    CLK_I : in std_logic;
    RESET_I : in std_logic; -- global reset

    -- connection to fpadc module
    TXCMD_DAT_O : out std_logic_vector(255 downto 0);
    TXCMD_LEN_O : out std_logic_vector(7 downto 0);
    TXCMD_RDY_O : out std_logic; -- '1' - issue tx, '0' - idle
    TXCMD_ACK_I : in std_logic; -- '1' - tx can be issued, '0' - busy

    RXCMD_DAT_I : in std_logic_vector(255 downto 0) ;
    RXCMD_LEN_I : in std_logic_vector(7 downto 0);
    RXCMD_RDY_I : in std_logic; -- '1' - data present, '0' - idle
    RXCMD_ACK_O : out std_logic; -- '1' - rx can occur, '0' - busy
    SIO_CLK_EN_O : out std_logic; -- sio clock enable

    ADC1_DAT_I : in std_logic_vector(15 downto 0);
    ADC2_DAT_I : in std_logic_vector(15 downto 0);
    DAC1_DAT_O : out std_logic_vector(15 downto 0);
    VGA_GAIN_O : out std_logic_vector(3 downto 0);
    IN_MUX_O   : out std_logic;

    ADC3_TX_DAT_O : out std_logic_vector(15 downto 0);
    ADC3_TX_RDY_O : out std_logic; -- '1' - issue tx, '0' - idle
    ADC3_TX_ACK_I : in std_logic; -- '1' - tx can be issued, '0' - busy
    ADC3_RX_DAT_I : in std_logic_vector(31 downto 0);
    ADC3_RX_RDY_I : in std_logic; -- '1' - data present, '0' - idle
    ADC3_RX_ACK_O : out std_logic; -- '1' - rx can occur, '0' - busy

    DEBUG_O : out std_logic_vector(9 downto 0);
    SCLK_I : in std_logic
    );
end component;

component fpadc_sio is
 port (
    -- standard signals
    CLK_I : in std_logic;
    RESET_I : in std_logic; -- global reset

    -- real io
    ADC3_SCLK_I : in std_ulogic;
    ADC3_SDIN_I : in std_ulogic;
    ADC3_DRDY_I : in std_ulogic;
    ADC3_SDOUT_O : out std_ulogic;

    -- interface to processing module
    TX_DAT_I : in std_logic_vector(15 downto 0);
    TX_RDY_I : in std_logic; -- '1' - issue tx, '0' - idle
    TX_ACK_O : out std_logic; -- '1' - tx can be issued, '0' - busy
    RX_DAT_O : out std_logic_vector(31 downto 0);
    RX_RDY_O : out std_logic; -- '1' - data present, '0' - idle
```

```
    RX_ACK_I : in std_logic -- '1' - rx can occur, '0' - busy
    );
end component;

    -- internal signals
    signal GLOBAL_RESET_INT : std_logic;
    signal GLOBAL_RESET2_INT : std_logic;
    signal RX_DAT_INT : std_logic_vector(7 downto 0);
    signal RX_RDY_INT : std_logic;
    signal RX_ACK_INT : std_logic;
    signal TX_DAT_INT : std_logic_vector(7 downto 0);
    signal TX_RDY_INT : std_logic;
    signal TX_ACK_INT : std_logic;
    signal DISPLAY_INT : std_logic_vector(7 downto 0);
    signal HEXLED0_INT : std_logic_vector(6 downto 0);
    signal HEXLED1_INT : std_logic_vector(6 downto 0);
    signal COUNTER_INT : std_logic_vector(31 downto 0);
    signal ADC3_TX_DAT_INT : std_logic_vector(15 downto 0);
    signal ADC3_TX_RDY_INT : std_logic;
    signal ADC3_TX_ACK_INT : std_logic;
    signal ADC3_RX_DAT_INT : std_logic_vector(31 downto 0);
    signal ADC3_RX_RDY_INT : std_logic;
    signal ADC3_RX_ACK_INT : std_logic;
    signal RXCMD_DAT_INT : std_logic_vector(255 downto 0);
    signal RXCMD_LEN_INT : std_logic_vector(7 downto 0);
    signal RXCMD_RDY_INT : std_logic;
    signal RXCMD_ACK_INT : std_logic;
    signal TXCMD_DAT_INT : std_logic_vector(255 downto 0);
    signal TXCMD_LEN_INT : std_logic_vector(7 downto 0);
    signal TXCMD_RDY_INT : std_logic;
    signal TXCMD_ACK_INT : std_logic;
    signal ADC1_DAT_INT : std_logic_vector(15 downto 0);
    signal ADC2_DAT_INT : std_logic_vector(15 downto 0);
    signal DAC1_DAT_INT : std_logic_vector(15 downto 0);
    signal VGA_GAIN_INT : std_logic_vector(3 downto 0);
    signal IN_MUX_INT : std_logic;
    signal ADC3_SCLK_INT : std_logic;
    signal ADC3_SDIN_INT : std_logic;
    signal ADC3_DRDY_INT : std_logic;
    signal ADC3_SDOUT_INT : std_logic;
    signal DEBUG_INT : std_logic_vector(9 downto 0);
    signal SIO_CLK_EN_INT : std_logic;

begin

    -------------------------------------------------
    -- clocks
    -------------------------------------------------

    CLK_DTOA1_STRATIX_PIN <= COUNTER_INT(0); -- 40MHz dac clock
    CLK_DTOA2_STRATIX_PIN <= COUNTER_INT(0); -- 40MHz dac clock
    CLK_OPT_A2D_PIN <= COUNTER_INT(0); -- 40MHz adc clock
```

```
   EVALIO_IN9_PIN <= COUNTER_INT(2) and SIO_CLK_EN_INT; -- 10MHz SIO
clock -- JP19 pin 21 (G1)

   ------------------------------------------------------
   -- hardwired connections
   ------------------------------------------------------

   HEX_0DP_PIN <= not DEBUG_INT(9); --left
   HEX_1DP_PIN <= not DEBUG_INT(8); --right
   DISPLAY_INT <= DEBUG_INT(7 downto 0);

   (HEX_0A_PIN, HEX_0B_PIN, HEX_0C_PIN, HEX_0D_PIN,
      HEX_0E_PIN, HEX_0F_PIN, HEX_0G_PIN) <= not HEXLED0_INT;
   (HEX_1A_PIN, HEX_1B_PIN, HEX_1C_PIN, HEX_1D_PIN,
      HEX_1E_PIN, HEX_1F_PIN, HEX_1G_PIN) <= not HEXLED1_INT;

   ADC1_DAT_INT <= (
      ATOD1_b11_PIN, ATOD1_b10_PIN, ATOD1_b9_PIN, ATOD1_b8_PIN,
      ATOD1_b7_PIN, ATOD1_b6_PIN, ATOD1_b5_PIN, ATOD1_b4_PIN,
      ATOD1_b3_PIN, ATOD1_b2_PIN, ATOD1_b1_PIN, ATOD1_b0_PIN,
      '0', '0', '0', '0');

   ADC2_DAT_INT <= (
      ATOD2_b11_PIN, ATOD2_b10_PIN, ATOD2_b9_PIN, ATOD2_b8_PIN,
      ATOD2_b7_PIN, ATOD2_b6_PIN, ATOD2_b5_PIN, ATOD2_b4_PIN,
      ATOD2_b3_PIN, ATOD2_b2_PIN, ATOD2_b1_PIN, ATOD2_b0_PIN,
      '0', '0', '0', '0');

   (DTOA1_b13_PIN, DTOA1_b12_PIN,
      DTOA1_b11_PIN, DTOA1_b10_PIN, DTOA1_b9_PIN, DTOA1_b8_PIN,
      DTOA1_b7_PIN, DTOA1_b6_PIN, DTOA1_b5_PIN, DTOA1_b4_PIN,
      DTOA1_b3_PIN, DTOA1_b2_PIN, DTOA1_b1_PIN, DTOA1_b0_PIN) <=
DAC1_DAT_INT(15 downto 2);

   EVALIO_IN1_PIN <= ADC3_SDOUT_INT; -- JP19 pin 5 (C3), cmd out
   ADC3_DRDY_INT <= EVALIO_IN2_PIN; -- JP19 pin 7 (D3), rdy in
   ADC3_SCLK_INT <= EVALIO_IN3_PIN; -- JP19 pin 9 (D2), clk in
   ADC3_SDIN_INT <= EVALIO_IN0_PIN; -- JP19 pin 3 (C2), cmd in
   -- JP19 pin 19 (G2), -- JP19 pin 17 (F1),
   -- JP19 pin 15 (F2), -- JP19 pin 13 (E2)
   (EVALIO_IN8_PIN, EVALIO_IN7_PIN, EVALIO_IN6_PIN, EVALIO_IN5_PIN) <=
      VGA_GAIN_INT;
   EVALIO_IN4_PIN <= IN_MUX_INT; -- JP19 pin 11 (E3)

   ------------------------------------------------------
   -- reset process
   ------------------------------------------------------

   process
      variable RESET_COUNTER_INT : std_logic_vector(7 downto 0);
   begin
      wait until Rising_Edge(COUNTER_INT(15));
```

```
      if (RESET_COUNTER_INT /= x"81") then
          GLOBAL_RESET_INT <= RESET_COUNTER_INT(6);
       RESET_COUNTER_INT := RESET_COUNTER_INT +1;
   end if;
end process;

-------------------------------------------------------
-- counter process
-------------------------------------------------------

process
begin
   wait until Rising_Edge(CLK_OSC_PIN);

   COUNTER_INT <= COUNTER_INT +1;

   LED0_PIN <= COUNTER_INT(24);
   LED1_PIN <= not COUNTER_INT(24);
end process;

-------------------------------------------------------
-- objects
-------------------------------------------------------

FPADCMAIN : fpadc_main
port map(
   CLK_I => CLK_OSC_PIN,
   RESET_I => GLOBAL_RESET_INT,
   TXCMD_DAT_O => TXCMD_DAT_INT,
   TXCMD_LEN_O => TXCMD_LEN_INT,
   TXCMD_RDY_O => TXCMD_RDY_INT,
   TXCMD_ACK_I => TXCMD_ACK_INT,
   RXCMD_DAT_I => RXCMD_DAT_INT,
   RXCMD_LEN_I => RXCMD_LEN_INT,
   RXCMD_RDY_I => RXCMD_RDY_INT,
   RXCMD_ACK_O => RXCMD_ACK_INT,
   ADC1_DAT_I => ADC1_DAT_INT,
   ADC2_DAT_I => ADC2_DAT_INT,
   DAC1_DAT_O => DAC1_DAT_INT,
   VGA_GAIN_O => VGA_GAIN_INT,
   IN_MUX_O => IN_MUX_INT,
   ADC3_TX_DAT_O => ADC3_TX_DAT_INT,
   ADC3_TX_RDY_O => ADC3_TX_RDY_INT,
   ADC3_TX_ACK_I => ADC3_TX_ACK_INT,
   ADC3_RX_DAT_I => ADC3_RX_DAT_INT,
   ADC3_RX_RDY_I => ADC3_RX_RDY_INT,
   ADC3_RX_ACK_O => ADC3_RX_ACK_INT,
   SIO_CLK_EN_O => SIO_CLK_EN_INT,
   DEBUG_O => DEBUG_INT,
   SCLK_I => COUNTER_INT(0)
);
```

```
PKTPROC : fpadc_pktprocessor
port map(
   CLK_I => CLK_OSC_PIN,
   RESET_I => GLOBAL_RESET_INT,
   RXCMD_DAT_O => RXCMD_DAT_INT,
   RXCMD_LEN_O => RXCMD_LEN_INT,
   RXCMD_RDY_O => RXCMD_RDY_INT,
   RXCMD_ACK_I => RXCMD_ACK_INT,
   TXCMD_DAT_I => TXCMD_DAT_INT,
   TXCMD_LEN_I => TXCMD_LEN_INT,
   TXCMD_RDY_I => TXCMD_RDY_INT,
   TXCMD_ACK_O => TXCMD_ACK_INT,
   RX_DAT_I => RX_DAT_INT,
   RX_RDY_I => RX_RDY_INT,
   RX_ACK_O => RX_ACK_INT,
   TX_DAT_O => TX_DAT_INT,
   TX_RDY_O => TX_RDY_INT,
   TX_ACK_I => TX_ACK_INT
);

SIO : fpadc_sio
port map(
   CLK_I => CLK_OSC_PIN,
   RESET_I => GLOBAL_RESET_INT,
   ADC3_SCLK_I => ADC3_SCLK_INT,
   ADC3_SDIN_I => ADC3_SDIN_INT,
   ADC3_DRDY_I => ADC3_DRDY_INT,
   ADC3_SDOUT_O => ADC3_SDOUT_INT,
   TX_DAT_I => ADC3_TX_DAT_INT,
   TX_RDY_I => ADC3_TX_RDY_INT,
   TX_ACK_O => ADC3_TX_ACK_INT,
   RX_DAT_O => ADC3_RX_DAT_INT,
   RX_RDY_O => ADC3_RX_RDY_INT,
   RX_ACK_I => ADC3_RX_ACK_INT

);

RS232 : fpadc_rs232
port map (
   CLK_I => CLK_OSC_PIN,
   RESET_I => GLOBAL_RESET_INT,
   RX_PAD_I => ROUT_PIN,
   TX_PAD_O => TIN_PIN,
   RX_DAT_O => RX_DAT_INT,
   RX_RDY_O => RX_RDY_INT,
   RX_ACK_I => RX_ACK_INT,
   TX_DAT_I => TX_DAT_INT,
   TX_RDY_I => TX_RDY_INT,
   TX_ACK_O => TX_ACK_INT
);
```

```
   HEXLED0 : fpadc_hexled
   port map(
      CLK_I => CLK_OSC_PIN,
      RESET_I => GLOBAL_RESET_INT,
      HEXLED_O => HEXLED0_INT,
      DATA_I   => DISPLAY_INT(7 downto 4)
   );

   HEXLED1 : fpadc_hexled
   port map(
      CLK_I => CLK_OSC_PIN,
      RESET_I => GLOBAL_RESET_INT,
      HEXLED_O => HEXLED1_INT,
      DATA_I   => DISPLAY_INT(3 downto 0)
   );

end;
```

Software – C Sourcecode

Organization

The Control Application source code is organized as follows:

```
|-- Debug
|   `-- ctlapp.exe
|-- Release
|   `-- ctlapp.exe
|-- StdAfx.cpp
|-- StdAfx.h
|-- WaveformDlg.cpp
|-- WaveformDlg.h
|-- ctlapp.aps
|-- ctlapp.clw
|-- ctlapp.cpp
|-- ctlapp.dsp
|-- ctlapp.dsw
|-- ctlapp.h
|-- ctlapp.ncb
|-- ctlapp.opt
|-- ctlapp.plg
|-- ctlapp.rc
|-- ctlappDlg.cpp
|-- ctlappDlg.h
|-- fpadc.cpp
|-- fpadc.h
|-- res
|   |-- Thumbs.db
|   |-- ctlapp.ico
|   `-- ctlapp.rc2
|-- resource.h
`-- samples.dat
```

FPADC Library - fpadc.cpp

```
#include <math.h>

/*
  FPADC support routines
*/
#include "stdafx.h"
#include "ctlapp.h"

#include "fpadc.h"

/*
   HIGH LEVEL FUNCTIONALITY
*/

/* variables */
int fpadc_clock_divider = 4; /*freq=(80/16)MHz/(1+div)*/
int fpadc_num_bits = 16;
HANDLE CommPort = INVALID_HANDLE_VALUE;

/* calibration tables */
unsigned int offset_table[CORRECTION_TABLE_SIZE],
gain_table[CORRECTION_TABLE_SIZE];

double
   /* user set unsigned dac offset for given gain */
   domain_offset[4] = {0.030, 0.120, 0.360, 0.810},
   /* user set gain switching point based on coarse dac value */
   domain_threshold[4] = {0.060, 0.180, 0.540, 1.000};

#if CALIBRATION_VALUES_SAVED /* calibrated values */

double
   /* measured relative gain difference as seen by remainder or
      calibration dac */
   gain_value[4] = {7.51, 3.82, 1.93, 1},
   /* multiplicant needed to obtain calibration ADC value based on
      remainder ADC value */
   calib_remadc_to_caladc_factor=111.2234,
   /* multiplicant needed to obtain calibration ADC value based on
      offset DAC value */
   calib_ofsdac_to_caladc_factor=220.7286;
int
   /* measured residual bias seen by calibration adc when input=0 */
   calib_bias_caladc[4] = {325987, 166688, 250243, 46520};

#else /* uncalibrated values */

double
   /* measured relative gain difference as seen by remainder or
```

```
      calibration dac */
   gain_value[4] = {8, 4, 2, 1},
   /* multiplicant needed to obtain calibration ADC value based on
      remainder ADC value */
   calib_remadc_to_caladc_factor=128,
   /* multiplicant needed to obtain calibration ADC value based on
      offset DAC value */
   calib_ofsdac_to_caladc_factor=256;
int
   /* measured residual bias seen by calibration adc when input=0 */
   calib_bias_caladc[4] = {0, 0, 0, 0};

#endif
/*
   Previously obtained calibration values:

   ofsdac_to_caladc_factor remadc_to_caladc_factor
   220.9231 111.2234
   Index ofsdacThresh   Gain  caladc_Bias
   0  0.0250   0.0500   7.6378   323080
   1  0.1000   0.1500   3.8954   166688
   2  0.3000   0.4500   1.9472   250243
   3  0.7250   1.0000   1.0000   46520
   Done.
*/

/* helper variables */
int i, j, k, s;

/* initialize the serial communication port */
unsigned int fpadc_init() {
   DCB   dcb;
   COMSTAT comstat;

   int i;
   unsigned char Packet[1] = {0};
   unsigned long NumWritten, NumRead, dummy;

   if (CommPort != NULL) {
      CloseHandle(CommPort);
   }

   CommPort = CreateFile("\\\\.\\COM1",
   GENERIC_READ | GENERIC_WRITE,
   0,
   NULL,
   OPEN_EXISTING,
   0,//FILE_FLAG_OVERLAPPED,
   NULL);

  /* Set communication state */
   if (GetCommState(CommPort, &dcb))
```

```
   {
      dcb.BaudRate = CBR_115200;
      dcb.fBinary = TRUE;
      dcb.fParity = FALSE;
      dcb.fOutxCtsFlow = FALSE;
      dcb.fOutxDsrFlow = FALSE;
      dcb.fDtrControl = DTR_CONTROL_DISABLE;
      dcb.fDsrSensitivity = FALSE;
      dcb.fTXContinueOnXoff = FALSE;
      dcb.fOutX = FALSE;
      dcb.fInX = FALSE;
      dcb.fErrorChar = FALSE;
      dcb.fNull = FALSE;
      dcb.fRtsControl = RTS_CONTROL_DISABLE;
      dcb.fAbortOnError = FALSE;
      dcb.ByteSize = 8;
      dcb.Parity = NOPARITY;
      dcb.StopBits = ONESTOPBIT;
      if (!SetCommState(CommPort, &dcb))
      {
         return -1;
      }
      Sleep(1000);
   }
   else {
      return -1;
   }

   /* reset io state machine */
   Packet[0] = 0;
   for (i=0; i<257; i++) {
      WriteFile(CommPort, &Packet[0], 1, &NumWritten, NULL);
   }

   Sleep(1000);

   /* reset port */
   do {
      ClearCommError(CommPort, &dummy, &comstat);

      // check number of bytes waiting in rx queue
      if (comstat.cbInQue>0) {
         ReadFile(CommPort, &Packet[0], 1, &NumRead, NULL);
      }
   } while (comstat.cbInQue>0);

   return 0;
}

/* update/reset calibration data table */
void fpadc_calibrate(int mode, CProgressCtrl *progressbar)
{
```

```
int i, j, val, val2;
int caladc_val1, caladc_val2, remadc_val1, remadc_val2, dac_val,
    dummy1, dummy2;

/* gain_bias_cadc */

fpadc_write_cmd(SET_SIO_CLOCK_ENABLE, 1);
fpadc_write_cmd(SET_IN_MUX, 1);

for (i=0; i<4; i++) {
    fpadc_write_cmd(SET_VGA_GAIN, i);
    fpadc_write_ofsdac(0);

   fpadc_read_caladc();fpadc_read_caladc();fpadc_read_caladc();fpadc
   _read_caladc();

   val=0;
   for (j=0; j<100; j++)
      val += fpadc_read_caladc();
    calib_bias_caladc[i]=val/j;
}

/* gain_value */
fpadc_write_ofsdac(0);
for (i=0; i<4; i++) {
   dac_val = 0x0800*(1<<i);

   fpadc_write_cmd(SET_VGA_GAIN, i);
   fpadc_write_ofsdac(-dac_val);

   fpadc_read_caladc();fpadc_read_caladc();fpadc_read_caladc();fpadc
   _read_caladc();

   caladc_val1=0;
   remadc_val1=0;
   for (j=0; j<100; j++) {
      caladc_val1 += fpadc_read_caladc();
      remadc_val1 += fpadc_read_remadc();
   }
   caladc_val1=caladc_val1/j;
   remadc_val1=remadc_val1/j;
   fpadc_write_ofsdac(+dac_val);

   fpadc_read_caladc();fpadc_read_caladc();fpadc_read_caladc();fpadc
   _read_caladc();

   caladc_val2=0;
   remadc_val2=0;
   for (j=0; j<100; j++) {
      caladc_val2 += fpadc_read_caladc();
      remadc_val2 += fpadc_read_remadc();
```

```
    }
    caladc_val2=caladc_val2/j;
    remadc_val2=remadc_val2/j;

    gain_value[i]=double(abs(caladc_val1
       -caladc_val2))/double(dac_val*2);
}

for (i=0; i<4; i++) {
    gain_value[i]=gain_value[i]/gain_value[3];
}

/* calib_ofsdac_to_caladc_factor */

i=3;
dac_val = 0x0800*(1<<i);

fpadc_write_cmd(SET_VGA_GAIN, i);
fpadc_write_ofsdac(-dac_val);

fpadc_read_caladc();fpadc_read_caladc();fpadc_read_caladc();fpadc_re
ad_caladc();

caladc_val1=0;
remadc_val1=0;
for (j=0; j<100; j++) {
    caladc_val1 += fpadc_read_caladc();
    remadc_val1 += fpadc_read_remadc();
}
caladc_val1=caladc_val1/j;
remadc_val1=remadc_val1/j;
fpadc_write_ofsdac(+dac_val);

fpadc_read_caladc();fpadc_read_caladc();fpadc_read_caladc();fpadc_re
ad_caladc();

caladc_val2=0;
remadc_val2=0;
for (j=0; j<100; j++) {
    caladc_val2 += fpadc_read_caladc();
    remadc_val2 += fpadc_read_remadc();
}
caladc_val2=caladc_val2/j;
remadc_val2=remadc_val2/j;

calib_ofsdac_to_caladc_factor=double(abs(caladc_val1
    -caladc_val2))/double(abs(dac_val*2));

/* calib_remadc_to_caladc_factor */

i=3;
dac_val = 0x0800*(1<<i);
```

```
   fpadc_write_cmd(SET_VGA_GAIN, i);

   fpadc_write_ofsdac(-dac_val);

   fpadc_read_caladc();fpadc_read_caladc();fpadc_read_caladc();fpadc_re
   ad_caladc();

   caladc_val1=0;
   remadc_val1=0;
   for (j=0; j<100; j++) {
      caladc_val1 += fpadc_read_caladc();
      remadc_val1 += fpadc_read_remadc();
   }
   caladc_val1=caladc_val1/j;
   remadc_val1=remadc_val1/j;
   fpadc_write_ofsdac(+dac_val);

   fpadc_read_caladc();fpadc_read_caladc();fpadc_read_caladc();fpadc_re
   ad_caladc();

   caladc_val2=0;
   remadc_val2=0;
   for (j=0; j<100; j++) {
      caladc_val2 += fpadc_read_caladc();
      remadc_val2 += fpadc_read_remadc();
   }
   caladc_val2=caladc_val2/j;
   remadc_val2=remadc_val2/j;

   calib_remadc_to_caladc_factor=double(abs(caladc_val1
      -caladc_val2))/double(abs(remadc_val1-remadc_val2));
}

/* generate exponent data table
   mode = 0 : no offset/no gain
   mode = 1 : linear offset/no gain
   mode = 2 : step offset/step gain
*/
void fpadc_lut_gen(int mode, CProgressCtrl *progressbar)
{
   double sign, adcval, dacval;
   int shift=0, shift2=0,  done;

   #define CORRECTION_TABLE_HALF_SIZE (CORRECTION_TABLE_SIZE>>1)

   /* i: value of Remainder ADC,
      j: lookup table index to corresponding i
      sign: sign correnSponding to value of i
   */
   for (i=-CORRECTION_TABLE_HALF_SIZE; i<CORRECTION_TABLE_HALF_SIZE;
        i++) {
```

```
    if (i>=0) {
       j = i;
    } else {
       j = i + CORRECTION_TABLE_SIZE;
    }

    if (i-shift>=0) {
       sign=+1.0;
    } else {
       sign=-1.0;
    }

    switch (mode) {
    case 0:
      offset_table[j] = 0;
      gain_table[j] = 0;
      break;

    case 1:
      offset_table[j] = i*0x160;
      gain_table[j] = 0;
      break;

    case 2:
       /* find the highest gain that will ensure
          that the ADC value is not out of bounds
       */
       done=0;
       for (int s=0; (s<4) && (done==0); s++) {
          if (abs(i-shift)<int((domain_threshold[s])
              *CORRECTION_TABLE_HALF_SIZE)
             || (s==3)) {
             dacval =
                calib_bias_caladc[s]/calib_ofsdac_to_caladc_factor
                +sign*double(1.0*domain_offset[s])*0x8000;

             offset_table[j]=
                int(dacval+(32768/(1<<fpadc_num_bits)))&0xffff;
             gain_table[j]=s;
             done=1; /* end */
          }
       }
      break;
    }
}

/* write table to fpga */
for (i=0; i<CORRECTION_TABLE_SIZE; i++) {
   fpadc_write_cmd(WRITE_OFFSET_TABLE,
      (i<<16)|((0x8000+offset_table[i])&0xffff));
   fpadc_write_cmd(WRITE_GAIN_TABLE, (i<<16)|gain_table[i]);
```

```
      if (progressbar!=NULL) {
         progressbar->SetPos((i*100)/CORRECTION_TABLE_SIZE);
      }
   }
}

/* Perform FPADC sampling cycle - save samples to a software buffer
*/
void fpadc_sample(double sample_buffer[], unsigned int num_samples, int
device, CProgressCtrl *progressbar)
{
   signed int i, val, val2, val3;
   static unsigned sval=0;
   double prev;
   long int mantissa, exponent;

   /* gather sample values */
   switch (device) {
      case FPADC_DEV_NONE:
         for (i=0; i<num_samples; i++) {
            sample_buffer[i]=0;
         }
         break;
      case FPADC_DEV_ADC_REMAINDER_SLOW:
         fpadc_write_cmd(SET_SIO_CLOCK_ENABLE, 0);
         fpadc_write_cmd(WRITE_OFFSET_DAC, 0x8000);
         fpadc_write_cmd(SET_IN_MUX, 1);

         val2=0; val3=0;
         for (i=0; i<num_samples; i++) {
            fpadc_write_cmd(WRITE_OFFSET_DAC, val2+0x8000);
            if (val2==0 && val3<60) {
               val3++;
            } else {
               val2+=0x100;
               val3=0;
            }
            fpadc_write_cmd(READ_REMAINDER_ADC, 0);
            val = fpadc_read_val()&REMAINDER_ADC_MASK;
            /* 2's complement to binary */
            if ((val&0x00008000)!=0) {
               val = val | 0xffff0000;
            }
            sample_buffer[i]=double(val)/65536;

            if (progressbar!=NULL) {
               progressbar->SetPos((i*100)/num_samples);
            }
         }
         fpadc_write_cmd(SET_SIO_CLOCK_ENABLE, 1);
         break;
```

```
case FPADC_DEV_ADC_COARSE_SLOW:
   fpadc_write_cmd(SET_SIO_CLOCK_ENABLE, 0);
   fpadc_write_cmd(WRITE_OFFSET_DAC, 0x8000);
   fpadc_write_cmd(SET_IN_MUX, 1);
   for (i=0; i<num_samples; i++) {
      fpadc_write_cmd(WRITE_OFFSET_DAC, i*256);
      fpadc_write_cmd(READ_COARSE_ADC, 0);
      val = fpadc_read_val()&COARSE_ADC_MASK;
      /* 2's complement to binary */
      if ((val&0x00008000)!=0) {
         val = val | 0xffff0000;
      }
      sample_buffer[i]=double(val)/65536;

      if (progressbar!=NULL) {
         progressbar->SetPos((i*100)/num_samples);
      }
   }
   fpadc_write_cmd(SET_SIO_CLOCK_ENABLE, 1);
   break;

case FPADC_DEV_ADC_REMAINDER_FAST:
   fpadc_write_cmd(SET_IN_MUX, 0);

   fpadc_write_cmd(SAMPLE_CYCLE, num_samples);
   Sleep(100);

   for (i=0; i<num_samples; i++) {
      fpadc_read_sample_mem(i, NULL, (unsigned int *)&val);
      val = val &0xffff;
      /* 2's complement to binary */
      if ((val&0x00008000)!=0) {
         val = val | 0xffff0000;
      }
      sample_buffer[i] = double(val)/65536;

      if (progressbar!=NULL) {
         progressbar->SetPos((i*100)/num_samples);
      }
   }
   break;

case FPADC_DEV_ADC_COARSE_FAST:
   fpadc_write_cmd(SET_IN_MUX, 0);

   fpadc_write_cmd(SAMPLE_CYCLE, num_samples);
   Sleep(100);

   for (i=0; i<num_samples; i++) {
      fpadc_read_sample_mem(i, NULL, (unsigned int *)&val);
      val = (val >>16)&0xffff;
```

```
            /* 2's complement to binary */
            if ((val&0x00008000)!=0) {
               val = val | 0xffff0000;
            }
            sample_buffer[i] = double(val)/65536;

            if (progressbar!=NULL) {
               progressbar->SetPos((i*100)/num_samples);
            }
         }
         break;

      case FPADC_DEV_FPADC:
         fpadc_write_cmd(SET_IN_MUX, 0);

         int remainder_adc_val, coarse_adc_val, offset_dac_val,
            vga_val, sign;
         FILE *FHandle;

         fpadc_write_cmd(SAMPLE_CYCLE, num_samples);
         Sleep(10);

         FHandle = fopen("c:\\temp\\samples.dat", "wt");

         for (i=0; i<num_samples; i++) {
            /* read sample values from FPGA */
            fpadc_read_sample_mem(num_samples-i-1,
               (unsigned int *)&val2, (unsigned int *)&val);

            remainder_adc_val = val&0xffff;
            if ((remainder_adc_val&0x00008000)!=0) {
               remainder_adc_val = remainder_adc_val | 0xffff0000;
            }
            coarse_adc_val = val>>16;
            if ((coarse_adc_val&0x00008000)!=0) {
               coarse_adc_val = coarse_adc_val | 0xffff0000;
            }
            offset_dac_val = (val2&0xffff)-0x8000;
            vga_val = (val2>>16)&0x3;

            if (offset_dac_val<=0) sign=+1;
            else sign=-1;

            mantissa = double(double(
               double(remainder_adc_val)*calib_remadc_to_caladc_factor
               +double(offset_dac_val)*calib_ofsdac_to_caladc_factor
               *gain_value[vga_val]
                  -calib_bias_caladc[vga_val]
                  +(1<<(23-fpadc_num_bits))
                  )*(double(8>>vga_val)/gain_value[vga_val]));

            exponent = vga_val;
```

```
            sample_buffer[i] = double(mantissa)*double(1<<exponent);

            /* scale to fit display */
            sample_buffer[i] = sample_buffer[i]/(256*256*256*8);

            fprintf(FHandle, "%i\t%i\t%i\t%i\t%i\n",
               i, coarse_adc_val, remainder_adc_val, offset_dac_val,
                vga_val*10000);

            if (progressbar!=NULL) {
               progressbar->SetPos((i*100)/num_samples);
            }
         }

         fclose(FHandle);

         break;

      case FPADC_DEV_ADC_CALIBRATION:
         fpadc_write_cmd(SET_IN_MUX, 1);
         fpadc_write_ofsdac(0);
         prev = -1;

         for (i=0; i<640+0*num_samples; i+=2) {
            fpadc_write_ofsdac(i*200);
            // dummy read result
            val=fpadc_read_caladc();
            // read result
            val=fpadc_read_caladc();

            if (prev == -1) prev = double(val);

            sample_buffer[i+0] = double((val+prev)/2);
            sample_buffer[i+1] = double(val);
            prev = double(val);

            if (progressbar!=NULL) {
               progressbar->SetPos((i*100)/640);
            }
         }

         val=int(sample_buffer[i-1]);

         for (i=640; i<num_samples; i++) {
            sample_buffer[i] = double(val);
         }

         break;
   }
}
```

```
int fpadc_read_caladc() {
   int retval=(fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_DOR2,
      3))&CALIBRATION_ADC_MASK)-((CALIBRATION_ADC_MASK+1)>>1);

   return retval;
}

int fpadc_read_remadc() {
   int retval=(fpadc_write_cmd_read_val(READ_REMAINDER_ADC,
       0)&REMAINDER_ADC_MASK);

   if ((retval&(1+((REMAINDER_ADC_MASK)>>1)))!=0)
      retval |= ~(REMAINDER_ADC_MASK);

   return retval;
}

int fpadc_read_crsadc() {
   int retval=(fpadc_write_cmd_read_val(READ_COARSE_ADC,
      0)&COARSE_ADC_MASK);

   if ((retval&(1+((REMAINDER_ADC_MASK)>>1)))!=0)
      retval |= ~(REMAINDER_ADC_MASK);

   return retval;
}

void fpadc_write_ofsdac(int dacval) {
   fpadc_write_cmd(WRITE_OFFSET_DAC,
      (dacval+((1+OFFSET_DAC_MASK)>>1))&OFFSET_DAC_MASK);
}

/*
   LOW LEVEL FUNCTIONALITY
*/

/* initialize ads1210 on the Serial IO bus on the FPGA */
void ads1210_setup()
{
   int init_done=0, num_tries=0, a;

   while (init_done==0) {

      /* setup */
      int cmd2, decimation, s_freq;

      /*
      Data Rate Decimation @ 10MHz
      (Hz) Ratio DR12 DR11 DR10 DR9 DR8 DR7 DR6 DR5 DR4 DR3 DR2 DR1 DR0
      1000 19    0    0    0    0   0   0   0   0   1   0   0   1   1
      500  38    0    0    0    0   0   0   0   1   0   0   1   1   0
      250  77    0    0    0    0   0   0   1   0   0   1   1   0   1
```

```
      100   194   0    0    0    0    0    1    1    0    0    0    0    1    0
      60    325   0    0    0    0    1    0    1    0    0    0    1    0    1
      50    90    0    0    0    0    1    1    0    0    0    0    1    1    0
      20    976   0    0    0    1    1    1    1    0    1    0    0    0    0
      10    1952  0    0    1    1    1    1    0    1    0    0    0    0    0
      */

      s_freq = 10;
      decimation = (194*10)/(s_freq);

      // cmd3 - BIAS on, REFO on, RESULTS 2s complement, UB biplr,
      // BD msb, MSB msb, SDL sdout, DSYNC reset
      fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_CMD3,
         1<<7|1<<6|0<<5|1<<1|1<<0));
      fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_CMD2,
         cmd2=(0<<5|2<<2|0<<0))); // cmd2 - MODE norm, GAIN 1, CHANNEL
1
      // cmd1 - TURBO MODE 1, DR12-DR8
      fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_CMD1,
         0<<5|((decimation&0x1f00)>>8)));
      // cmd0 - DR7-DR0 - xxxHz sampling @ 10MHz clk
      fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_CMD0,
(decimation&0xff)));

      /*fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_CAL2, 0x0));
      fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_CAL1, 0x0));
      fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_CAL0, 0x0));
      fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_FSCAL2, 0x0));
      fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_FSCAL1, 0x0));
      fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_FSCAL0, 0x0));*/

      // cmd2 - MODE selfcalib
      fpadc_sio_cmd(ADS1210_WRITE(ADS1210_ADR_CMD2, (cmd2|(1<<5))));

      fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_DOR2, 3));
      fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_DOR2, 3));
      init_done=1;
   }
}

/* Send a command packet to the FPGA */
unsigned int fpadc_write_cmd(unsigned int cmd, unsigned int arg) {
   unsigned char Packet[257];
   unsigned long PacketLen, NumWritten;
   int timer=0;

   Packet[5] = cmd;
   Packet[4] = (arg>>24)&0xff;
   Packet[3] = (arg>>16)&0xff;
   Packet[2] = (arg>>8)&0xff;
   Packet[1] = (arg)&0xff;
```

```
   PacketLen = 6;
   Packet[0] = PacketLen-1;

   WriteFile(CommPort, &Packet[0], PacketLen, &NumWritten, NULL);
   FlushFileBuffers(CommPort);

   return 0;
}

/* Read the response packet from the FPGA */
unsigned int fpadc_read_val(void) {
   int timer=0;
   unsigned char Packet[257];
   unsigned long PacketLen, NumRead=0, val, dummy;
   COMSTAT comstat;
   static Abort=0;

   if (Abort!=0) return -1;

   while (NumRead<4) {
      ClearCommError(CommPort, &dummy, &comstat);

      // check number of bytes waiting in rx queue
      if (comstat.cbInQue>=4) {
         ReadFile(CommPort, &Packet[0], 4, &NumRead, NULL);
      } else {
         Sleep(1);
         timer++;

         if (timer>FPADC_CMD_TIMEOUT) {
            Abort=1;
            AfxMessageBox("ABORTING: Communication Error. Try resetting
               the board.");
            exit(-1);
         }
      }
   }

   val = Packet[0]<<24 | Packet[1]<<16 | Packet[2]<<8 | Packet[3];

   return val;
}

/* Issue a command to the FPGA and read the response */
unsigned int fpadc_write_cmd_read_val(unsigned int cmd,
   unsigned int arg) {
   fpadc_write_cmd(cmd, arg);
   return fpadc_read_val();
}

/* Read up to 64 bits of the stored sample value from the FPGA */
void fpadc_read_sample_mem(unsigned int index, unsigned int *val_hi,
```

```
   unsigned int *val_lo) {

   if (val_lo!=NULL) {
      *val_lo=fpadc_write_cmd_read_val(READ_SAMPLE_LO_TABLE,
index<<16);
   }
   if (val_hi!=NULL) {
      *val_hi=fpadc_write_cmd_read_val(READ_SAMPLE_HI_TABLE,
index<<16);
   }
}

unsigned int fpadc_get_status(unsigned int mask) {
   return 0;
}

/* Issue an Serial IO command to the FPGA */
unsigned int fpadc_sio_cmd(unsigned int cmd) {
   int timer=0;
   unsigned int ret;

   ret=fpadc_write_cmd_read_val(SIO_ADS1210_CMD, cmd);

   /* adjust sample value from 2s complement to straight binary */
   if (cmd==ADS1210_READ(ADS1210_ADR_DOR2, 3)) {
      ret = ret&0x00ffffff;
      /* do we have a 24 bit negative value - expand it to 32 bit */
      if ((ret&0x00800000)!=0) {
         ret = ret | 0xff000000;
      }

      /* convert 2s comp to binary by adding 0x00800000 */
      ret += 0x00800000;
   }

   return ret;
}

/* perform a 16 bit wide bitflipping operation -
   i.e. MSb becomes LSb, etc */
unsigned int bitflip16(unsigned int value) {
   unsigned int result=0;

   for (int i=0; i<16; i++) {
      result*=2;
      if ((value&(1<<i))!=0) {
         result++;
      }
   }

   return result;
}
```

FPADC Library Header - fpadc.h

```
/*
 * Definition for fpadc board
 */

/*
 * fpadc commands
 */

#define READ_REMAINDER_ADC        0x10
#define READ_COARSE_ADC           0x11
#define WRITE_OFFSET_DAC          0x12
#define SIO_ADS1210_CMD           0x20
#define WRITE_OFFSET_TABLE        0x38
#define WRITE_GAIN_TABLE          0x39
#define SAMPLE_CYCLE              0x40
#define READ_SAMPLE_HI_TABLE      0x50
#define READ_SAMPLE_LO_TABLE      0x51
#define SET_VGA_GAIN              0x70
#define SET_IN_MUX                0x71
#define SET_SAMPLING_CLK_DIVIDER  0xa0
#define SET_SIO_CLOCK_ENABLE      0xa1
#define SET_ADC_MASK              0xa2
#define SET_DAC_MASK              0xa3

/*
  Command return value masks
*/

#define ADC_MASK                  0x0000ffff
#define COARSE_ADC_MASK           0x0000ffff
#define REMAINDER_ADC_MASK        0x0000ffff
#define CALIBRATION_ADC_MASK      0x00ffffff
#define OFFSET_DAC_MASK           0x0000ffff


/*
  FPADC command status masks
*/
#define STATUS_FPADCCMD_BUSY      0x80000000
#define STATUS_MEMWRITE_BUSY      0x40000000
#define STATUS_MEMREAD_BUSY       0x20000000
#define STATUS_SIOCMD_BUSY        0x10000000
#define STATUS_ALL                0xffffffff

/*
  ADS1210 ADS definitions
*/

#define ADS1210_INS(RW, MB, AD)
```

```
      ((((RW&1)<<7)|(((MB-1)&3)<<5)|((AD&15)<<0))<<8)
#define ADS1210_READ(AD, MB)      (ADS1210_INS(1, MB, AD))
#define ADS1210_WRITE(AD, VAL)    (ADS1210_INS(0, 1, AD)|(VAL&0xff))

/* address map for serial ADC */
#define ADS1210_ADR_DOR2   0
#define ADS1210_ADR_DOR1   1
#define ADS1210_ADR_DOR0   2
#define ADS1210_ADR_CMD3   4
#define ADS1210_ADR_CMD2   5
#define ADS1210_ADR_CMD1   6
#define ADS1210_ADR_CMD0   7
#define ADS1210_ADR_CAL2   8
#define ADS1210_ADR_CAL1   9
#define ADS1210_ADR_CAL0   10
#define ADS1210_ADR_FSCAL2 12
#define ADS1210_ADR_FSCAL1 13
#define ADS1210_ADR_FSCAL0 14

/* table sizes */
#define CORRECTION_TABLE_SIZE 256

/* maximum value of lut_gen mode */
#define MAX_LUT_GEN_MODE 2

/* device definitions for fpadc_sample() device */
#define FPADC_DEV_NONE                    0
#define FPADC_DEV_ADC_REMAINDER_SLOW      1
#define FPADC_DEV_ADC_COARSE_SLOW         2
#define FPADC_DEV_ADC_REMAINDER_FAST      3
#define FPADC_DEV_ADC_COARSE_FAST         4
#define FPADC_DEV_FPADC                   5
#define FPADC_DEV_ADC_CALIBRATION         6

/* low level communication constants */
#define FPADC_CMD_TIMEOUT  1000 /* in 0.01 sec intervals */

/* externs */
extern int fpadc_clock_divider;
extern int fpadc_num_bits;
extern void ads1210_setup();
extern unsigned int offset_table[CORRECTION_TABLE_SIZE],
gain_table[CORRECTION_TABLE_SIZE];
extern double domain_offset[4], domain_threshold[4], gain_value[4],
   calib_remadc_to_caladc_factor, calib_ofsdac_to_caladc_factor;
extern int calib_bias_caladc[4];//, gain_bias_correction_ofsdac[4],
   domain_offset_correction_caladc[4][2];
/* high-level */
void fpadc_calibrate(int mode, CProgressCtrl *progressbar);
void fpadc_lut_gen(int mode, CProgressCtrl *progressbar);
void fpadc_sample(double sample_buffer[], unsigned int num_samples,
   int device, CProgressCtrl *progressbar);
```

```
int fpadc_match_caladc(int final_adc_val);
int fpadc_match_remadc(int final_adc_val, int &low, int &high);

/* low-level */
unsigned int fpadc_init();
unsigned int fpadc_write_cmd(unsigned int cmd, unsigned int arg);
unsigned int fpadc_read_val(void);
unsigned int fpadc_write_cmd_read_val(unsigned int cmd,
   unsigned int arg);
void fpadc_read_sample_mem(unsigned int index, unsigned int *val_hi,
   unsigned int *val_lo);
unsigned int fpadc_get_status(unsigned int mask);
unsigned int fpadc_sio_cmd(unsigned int cmd);
unsigned int bitflip16(unsigned int value);
int fpadc_read_caladc();
int fpadc_read_crsadc();
int fpadc_read_remadc();
void fpadc_write_ofsdac(int dacval);
```

FPADC Application Main Window - ctlappDlg.cpp

```
// ctlappDlg.cpp : implementation file
//

#include "stdafx.h"
#include "ctlapp.h"
#include "ctlappDlg.h"
#include "WaveformDlg.h"

#include <stdio.h>
#include <time.h>
#include <math.h>

#include "fpadc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

unsigned int BoardInitialized = 0;

/////////////////////////////////////////////////////////////////////////////
/
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
   CAboutDlg();

// Dialog Data
   //{{AFX_DATA(CAboutDlg)
   enum { IDD = IDD_ABOUTBOX };
   //}}AFX_DATA

   // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAboutDlg)
   protected:
   virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
   //}}AFX_VIRTUAL

// Implementation
protected:
   //{{AFX_MSG(CAboutDlg)
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};
```

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
   //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAboutDlg)
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
   //{{AFX_MSG_MAP(CAboutDlg)
      // No message handlers
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
/
// CCtlappDlg dialog

CCtlappDlg::CCtlappDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CCtlappDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CCtlappDlg)
   m_console_text = _T("");
   //}}AFX_DATA_INIT
   // Note that LoadIcon does not require a subsequent DestroyIcon in
Win32
   m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CCtlappDlg::ConsoleClear(void)
{
   CCtlappDlg::m_console_text="";
   CCtlappDlg::m_console.Clear();
   CCtlappDlg::m_console.Invalidate(TRUE);
}

void CCtlappDlg::ConsoleAdd(CString Text)
{
   CCtlappDlg::m_console_text=CCtlappDlg::m_console_text+Text;
   CCtlappDlg::m_console.SetWindowText(CCtlappDlg::m_console_text);
   CCtlappDlg::m_console.Invalidate(TRUE);
}

void CCtlappDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CCtlappDlg)
```

```
    DDX_Control(pDX, IDC_EDIT1, m_console);
    DDX_Text(pDX, IDC_EDIT1, m_console_text);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCtlappDlg, CDialog)
    //{{AFX_MSG_MAP(CCtlappDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_COMMAND(ID_COMMAND_ACQUIRE, OnCommandAcquire)
    ON_COMMAND(ID_COMMAND_CALIBRATE, OnCommandCalibrate)
    ON_COMMAND(ID_COMMAND_RESET, OnCommandReset)
    ON_COMMAND(ID_COMMAND_DEBUGFUNCTION1, OnCommandDebugfunction1)
    ON_COMMAND(ID_COMMAND_DEBUGFUNCTION2, OnCommandDebugfunction2)
    ON_COMMAND(ID_COMMAND_DEBUGFUNCTION3, OnCommandDebugfunction3)
    ON_COMMAND(ID_COMMAND_DEBUGFUNCTION4, OnCommandDebugfunction4)
    ON_COMMAND(ID_COMMAND_EXIT, OnCommandExit)
    ON_COMMAND(ID_HELP_ABOUT, OnHelpAbout)
    ON_COMMAND(ID_WINDOW_RAWDATA, OnWindowRawdata)
    ON_COMMAND(ID_WINDOW_WAVEFORM, OnWindowWaveform)
    ON_COMMAND(ID_COMMAND_DEBUGFUNCTION5, OnCommandDebugfunction5)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
/
// CCtlappDlg message handlers

BOOL CCtlappDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }
```

```
   // Set the icon for this dialog.  The framework does this
automatically
   //  when the application's main window is not a dialog
   SetIcon(m_hIcon, TRUE);        // Set big icon
   SetIcon(m_hIcon, FALSE);       // Set small icon

   // TODO: Add extra initialization here
   CONSOLE_CLEAR;
   CONSOLE_ADD("Self-Calibrating Floating Point ADC control application
ready.\r\n");

   // 'click' the reset function
   OnCommandReset();

   return TRUE;  // return TRUE  unless you set the focus to a control
}

void CCtlappDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
   if ((nID & 0xFFF0) == IDM_ABOUTBOX)
   {
      CAboutDlg dlgAbout;
      dlgAbout.DoModal();
   }
   else
   {
      CDialog::OnSysCommand(nID, lParam);
   }
}

void CCtlappDlg::OnPaint()
{
   if (IsIconic())
   {
      CPaintDC dc(this); // device context for painting

      SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

      // Center icon in client rectangle
      int cxIcon = GetSystemMetrics(SM_CXICON);
      int cyIcon = GetSystemMetrics(SM_CYICON);
      CRect rect;
      GetClientRect(&rect);
      int x = (rect.Width() - cxIcon + 1) / 2;
      int y = (rect.Height() - cyIcon + 1) / 2;

      // Draw the icon
      dc.DrawIcon(x, y, m_hIcon);
   }
   else
   {
      CDialog::OnPaint();
```

```
    }
}

// The system calls this to obtain the cursor to display while the
// user drags the minimized window.
HCURSOR CCtlappDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CCtlappDlg::OnCommandAcquire()
{
    CONSOLE_CLEAR;
    CONSOLE_ADD("Acquiring...\r\n");
    CONSOLE_ADD("Done.\r\n");
}

void CCtlappDlg::OnCommandCalibrate()
{
    CONSOLE_CLEAR;
    CONSOLE_ADD("Calibrating...\r\n");
    CONSOLE_ADD("Done.\r\n");
}

void CCtlappDlg::OnCommandReset()
{
    char buffer[1024];
    unsigned int val, i;

    BoardInitialized = 0;

    CONSOLE_CLEAR;
    CONSOLE_ADD("Resetting...\r\n");

    CONSOLE_ADD("- Initializing communication link.\r\n");

    if (fpadc_init()!=0) {
        CONSOLE_ADD("- ERROR: Could not initialize comm!\r\n");
        return;
    }

    fpadc_write_cmd(SET_SAMPLING_CLK_DIVIDER, 0);
    fpadc_write_cmd(SET_SIO_CLOCK_ENABLE, 1);

    fpadc_write_cmd(SET_SAMPLING_CLK_DIVIDER, fpadc_clock_divider);

    CONSOLE_ADD("- Initializing ADC/DAC masks\r\n");
    fpadc_write_cmd(SET_ADC_MASK, bitflip16((1<<fpadc_num_bits)-1));
    fpadc_write_cmd(SET_DAC_MASK, bitflip16((1<<16)-1));

    CONSOLE_ADD("- Initializing VGA\r\n");
    fpadc_write_cmd(SET_VGA_GAIN, 0);
```

```
    fpadc_write_cmd(WRITE_OFFSET_DAC, 0x8000);
    fpadc_write_cmd(SET_IN_MUX, 0);

    Sleep(100);

    CONSOLE_ADD("- Initializing ADS1210 ADC\r\n");
    ads1210_setup();

    CONSOLE_ADD("- Initializing FPADC lookup tables\r\n");

    CONSOLE_ADD("Done.\r\n");

    BoardInitialized = 1;
}

void CCtlappDlg::OnCommandDebugfunction1()
{
    char buffer[1024];
    unsigned int val=0xdeadbeef;
    int a;

    CONSOLE_CLEAR;

    if (BoardInitialized==0) {
       CONSOLE_ADD("ERROR: Board not initialized\r\n");
       return;
    }

    CONSOLE_ADD("Sample memory dump\r\n");
    for (a=0; a<256; a++) {
       if (a%8==0) {
          sprintf(buffer, "%04X: ", a);
          CONSOLE_ADD(buffer);
       }

       fpadc_read_sample_mem(a, NULL, (unsigned int*)&val);
       sprintf(buffer, "%08X ", val);
       CONSOLE_ADD(buffer);

       if (a%8!=7) {
          CONSOLE_ADD(" ");
       } else {
          CONSOLE_ADD("\r\n");
       }
    }
    if (a%8!=7) {
       CONSOLE_ADD("\r\n");
    }
    CONSOLE_ADD("\r\n");

    CONSOLE_ADD("Done.\r\n");
}
```

```
void CCtlappDlg::OnCommandDebugfunction2()
{
   static int led_state=0;
   char buffer[1024];

   CONSOLE_CLEAR;

   if (BoardInitialized==0) {
      CONSOLE_ADD("ERROR: Board not initialized\r\n");
      return;
   }

   led_state++;
   if(led_state>3) led_state=0;

   CONSOLE_ADD("Setting LED to ");
   sprintf(buffer, "%X ", led_state);
   CONSOLE_ADD(buffer);
   CONSOLE_ADD("\r\n");

   fpadc_write_cmd(SET_LED, led_state);

   CONSOLE_ADD("Done.\r\n");
}

void CCtlappDlg::OnCommandDebugfunction3()
{
   int a, val1, val2;
   char buffer[1024];

   CONSOLE_CLEAR;

   if (BoardInitialized==0) {
      CONSOLE_ADD("ERROR: Board not initialized\r\n");
      return;
   }

   CONSOLE_ADD("FPADC dump:\r\n");

   for (a=0; a<16; a++) {
      if (a%8==0) {
         sprintf(buffer, "%04X: ", a);
         CONSOLE_ADD(buffer);
      }

      val1=fpadc_write_cmd_read_val(READ_REMAINDER_ADC,
         0)&REMAINDER_ADC_MASK;
      val2=fpadc_write_cmd_read_val(READ_COARSE_ADC,
0)&COARSE_ADC_MASK;

      sprintf(buffer, "%08X ", val1<<16 | val2);
```

```
      CONSOLE_ADD(buffer);
      if (a%8!=7) {
         CONSOLE_ADD(" ");
      } else {
         CONSOLE_ADD("\r\n");
      }
   }


   CONSOLE_ADD("Done.\r\n");
}

void CCtlappDlg::OnCommandDebugfunction4()
{
   unsigned int val, a;
   char buffer[1024];
   unsigned int vals[1024];

   CONSOLE_CLEAR;

   if (BoardInitialized==0) {
      CONSOLE_ADD("ERROR: Board not initialized\r\n");
      return;
   }

   fpadc_write_cmd(SET_VGA_GAIN, 0);
   fpadc_write_cmd(WRITE_OFFSET_DAC, 0);
   fpadc_write_cmd(SET_IN_MUX, 1);

   val = fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_CMD3, 4));
   CONSOLE_ADD("CMD: "); itoa(val, buffer, 16); CONSOLE_ADD(buffer);
   CONSOLE_ADD("\r\n");
   val = fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_CMD3, 4));
   CONSOLE_ADD("CMD: "); itoa(val, buffer, 16); CONSOLE_ADD(buffer);
   CONSOLE_ADD("\r\n");

   val = fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_CAL2, 3));
   CONSOLE_ADD("CAL: "); itoa(val, buffer, 16); CONSOLE_ADD(buffer);
   CONSOLE_ADD("\r\n");
   val = fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_CAL2, 3));
   CONSOLE_ADD("CAL: "); itoa(val, buffer, 16); CONSOLE_ADD(buffer);
   CONSOLE_ADD("\r\n");

   val = fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_FSCAL2, 3));
   CONSOLE_ADD("FSCAL: "); itoa(val, buffer, 16); CONSOLE_ADD(buffer);
   CONSOLE_ADD("\r\n");
   val = fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_FSCAL2, 3));
   CONSOLE_ADD("FSCAL: "); itoa(val, buffer, 16); CONSOLE_ADD(buffer);
   CONSOLE_ADD("\r\n");

   // read values
   for (a=0; a<64; a++) {
```

```
      fpadc_write_cmd(WRITE_OFFSET_DAC, -a*1024);
      // dummy read
      vals[a] = fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_DOR2, 3));
      // real read
      vals[a] = fpadc_sio_cmd(ADS1210_READ(ADS1210_ADR_DOR2, 3));
   }

   for (a=0; a<64; a++) {
      if (a%8==0) {
         sprintf(buffer, "Sample %04X: ", a);
         CONSOLE_ADD(buffer);
      }

      sprintf(buffer, "%08X ", vals[a]);
      CONSOLE_ADD(buffer);
      if (a%8!=7) {
         CONSOLE_ADD(" ");
      } else {
         CONSOLE_ADD("\r\n");
      }
   }

   CONSOLE_ADD("Done.\r\n");
}

void CCtlappDlg::OnCommandDebugfunction5()
{
   char buffer[1024];
   int i;

   CONSOLE_CLEAR;

   if (BoardInitialized==0) {
      CONSOLE_ADD("ERROR: Board not initialized\r\n");
      return;
   }

   fpadc_calibrate(0,0);
   fpadc_lut_gen(2,0);

   /* print out the calibration info */
   sprintf(buffer,
      "ofsdac_to_caladc_factor\tremadc_to_caladc_factor\r\n");
   CONSOLE_ADD(buffer);
   sprintf(buffer, "%.4f\t%.4f\r\n",calib_ofsdac_to_caladc_factor,
      calib_remadc_to_caladc_factor);
   CONSOLE_ADD(buffer);

   sprintf(buffer, "Index\tofsdac\tThresh\tGain\tcaladc_Bias\r\n");
   CONSOLE_ADD(buffer);
   for (i=0; i<4; i++) {
      sprintf(buffer, "%i\t%.4f\t%.4f\t%.4f\t%i\r\n",
```

```
            i,
            domain_offset[i], domain_threshold[i], gain_value[i],
            calib_bias_caladc[i]);
         CONSOLE_ADD(buffer);
      }

      CONSOLE_ADD("Done.\r\n");
}

void CCtlappDlg::OnCommandExit()
{
      CCtlappDlg::DestroyWindow();
}

void CCtlappDlg::OnHelpAbout()
{
      CAboutDlg dlgAbout;
      dlgAbout.DoModal();
}

void CCtlappDlg::OnWindowRawdata()
{
}

void CCtlappDlg::OnWindowWaveform()
{

      if (BoardInitialized==0) {
         CONSOLE_CLEAR;
         CONSOLE_ADD("ERROR: Board not initialized\r\n");
         return;
      }

      CWaveformDlg dlgWaveform;
      dlgWaveform.DoModal();
}
```

FPADC Application Graph Window - WaveformDlg.cpp

```
// WaveformDlg.cpp : implementation file
//
#include "stdafx.h"
#include "ctlapp.h"
#include "WaveformDlg.h"
#include <math.h>
#include <stdlib.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#include "fpadc.h"

/* local defines */
#define ID_PAINT_TIMER  1000
#define NUM_SAMPLES 10001 /* we're ignoring the 1st sample */

static double sample_buffer[NUM_SAMPLES];

int DisplayDevice = FPADC_DEV_NONE;
char *DisplayString = "No Source Selected";
double LowVal=0, MidVal=0, HiVal=0;
int TimedRefresh = 0;
int DCFilterEnabled = 0;
int AutoScaleEnabled = 0;
int ResampleOnPaint = 1;
double Offset=0, Zoom=1;
int DACtest = 0;
int controlsNeedUpdate = 1;

/////////////////////////////////////////////////////////////////////////////
/
// CWaveformDlg dialog

CWaveformDlg::CWaveformDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CWaveformDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CWaveformDlg)
    //}}AFX_DATA_INIT

    controlsNeedUpdate = 1;
}

void CWaveformDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
```

```
    //{{AFX_DATA_MAP(CWaveformDlg)
    DDX_Control(pDX, IDC_BITS, m_Bits);
    DDX_Control(pDX, IDC_BITSDISP, m_BitsDisp);
    DDX_Control(pDX, IDDACOUT, m_DACOut);
    DDX_Control(pDX, IDMUX2, m_Mux2Button);
    DDX_Control(pDX, IDMUX1, m_Mux1Button);
    DDX_Control(pDX, IDCALIBRATE, m_CalibrateButton);
    DDX_Control(pDX, IDC_COMMPROGRESS, m_CommProgress);
    DDX_Control(pDX, IDC_CLOCKDISP, m_ClockDisp);
    DDX_Control(pDX, IDC_ZOOMDISP, m_ZoomDisp);
    DDX_Control(pDX, IDC_OFFSETDISP, m_OffsetDisp);
    DDX_Control(pDX, IDC_SPEED, m_Speed);
    DDX_Control(pDX, IDC_ZOOM, m_Zoom);
    DDX_Control(pDX, IDC_OFFSET, m_Offset);
    DDX_Control(pDX, IDC_DCFILTER, m_DcFilter);
    DDX_Control(pDX, IDC_AUTOSCALE, m_AutoScale);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CWaveformDlg, CDialog)
    //{{AFX_MSG_MAP(CWaveformDlg)
    ON_WM_PAINT()
    ON_WM_TIMER()
    ON_WM_CANCELMODE()
    ON_WM_CREATE()
    ON_WM_DESTROY()
    ON_BN_CLICKED(IDADC1S, OnAdc1s)
    ON_BN_CLICKED(IDADC2S, OnAdc2s)
    ON_BN_CLICKED(IDAFPADC, OnAfpadc)
    ON_BN_CLICKED(IDAuto, OnAuto)
    ON_BN_CLICKED(IDTRIGGER, OnTrigger)
    ON_BN_CLICKED(IDAADCCALIB, OnAadccalib)
    ON_BN_CLICKED(IDC_DCFILTER, OnDcfilter)
    ON_BN_CLICKED(IDC_AUTOSCALE, OnAutoscale)
    ON_BN_CLICKED(IDCALIBRATE, OnCalibrate)
    ON_NOTIFY(NM_RELEASEDCAPTURE, IDC_OFFSET, OnSetOffset)
    ON_NOTIFY(NM_RELEASEDCAPTURE, IDC_ZOOM, OnSetZoom)
    ON_WM_CAPTURECHANGED()
    ON_NOTIFY(NM_CUSTOMDRAW, IDC_OFFSET, OnCustomdrawOffset)
    ON_NOTIFY(NM_CUSTOMDRAW, IDC_ZOOM, OnCustomdrawZoom)
    ON_NOTIFY(NM_RELEASEDCAPTURE, IDC_SPEED, OnSetSpeed)
    ON_BN_CLICKED(IDSAVE, OnSave)
    ON_BN_CLICKED(IDADC1DMA, OnAdc1dma)
    ON_BN_CLICKED(IDADC2DMA, OnAdc2dma)
    ON_BN_CLICKED(IDMUX1, OnMux1)
    ON_BN_CLICKED(IDMUX2, OnMux2)
    ON_BN_CLICKED(IDDACOUT, OnDacout)
    ON_NOTIFY(NM_RELEASEDCAPTURE, IDC_BITS, OnReleasedcaptureBits)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////////////////////////////////////////
/
// CWaveformDlg message handlers

void CWaveformDlg::OnPaint()
{
   unsigned int num_samples=NUM_SAMPLES;
   unsigned int x, y, init, i, y2, ncrossings, crossing,
      crossingfound=0, height, midpoint;
   double freq, period, top, bottom, mean, average, min, max,
      intaverage, zoom, val;
   CRect rect;
    char status[1024];

   if (controlsNeedUpdate == 1) {
      /* set up the sliders */
      m_Zoom.SetRangeMin(0, false);
      m_Zoom.SetRangeMax(100, false);
      m_Zoom.SetTicFreq(10);
      sprintf(status, "%.2f", Zoom);
      m_ZoomDisp.SetWindowText(status);

      m_Offset.SetRangeMin(-100, false);
      m_Offset.SetRangeMax(100, false);
      m_Offset.SetTicFreq(10);
      sprintf(status, "%.2f", Offset);
      m_OffsetDisp.SetWindowText(status);

      m_Speed.SetRangeMin(0, false);
      m_Speed.SetRangeMax(29, false);
      m_Speed.SetTicFreq(2);
      m_Speed.SetPos(sqrt(fpadc_clock_divider));
      sprintf(status, "SClk:%.6fMhz",
         double((40.0/16.0)/(1+double(fpadc_clock_divider))));
      m_ClockDisp.SetWindowText(status);

      m_Bits.SetRangeMin(1, false);
      m_Bits.SetRangeMax(16, false);
      m_Bits.SetTicFreq(1);
      m_Bits.SetPos(fpadc_num_bits);
      sprintf(status, "%i bits", fpadc_num_bits);
      m_BitsDisp.SetWindowText(status);

      m_CommProgress.SetPos(0);
      m_CommProgress.SetRange(0, 100);

      controlsNeedUpdate =0;
   }

   /* get the sample values */
   if (ResampleOnPaint!=0) {
      fpadc_sample(sample_buffer, num_samples, DisplayDevice,
```

```
      &m_CommProgress);
   ResampleOnPaint = 0;
}

/* dc filter */
if (DCFilterEnabled!=0) {
   average=0;

   for (i=0; i<num_samples; i++) {
      average += sample_buffer[i];
   }
   average /= i;
   intaverage=unsigned int(average);
   for (i=0; i<num_samples; i++) {
      val = unsigned int(sample_buffer[i]);
      val += -intaverage;
   }
}

/* auto sizing filter */
if (AutoScaleEnabled!=0) {
   min =  1;
   max = -1;

   for (i=0; i<num_samples; i++) {
      if (min>(sample_buffer[i])) min=sample_buffer[i];
      if (max<(sample_buffer[i])) max=sample_buffer[i];
   }

   zoom = 1/(max-min+1);

   for (i=0; i<num_samples; i++) {
      sample_buffer[i] = sample_buffer[i]*zoom;
   }
}

CPaintDC dc(this); // device context for painting

CWnd* pWnd=GetDlgItem(IDC_WAVE);
pWnd->GetWindowRect(&rect);
ScreenToClient(&rect);

sprintf(status, "%+.2fV", (HiVal*Offset)+HiVal/Zoom);
dc.TextOut(rect.left, rect.top+16, status);

sprintf(status, "%+.2fV", (HiVal*Offset)+MidVal/Zoom);
dc.TextOut(rect.left, (rect.top+rect.bottom)/2, status);

sprintf(status, "%+.2fV", (HiVal*Offset)+LowVal/Zoom);
dc.TextOut(rect.left, rect.bottom-16, status);

init=0;
```

```
   i=0;
   top = -1;
   bottom = 1;
   mean = 0;

   y2 = 0;
   ncrossings = 0;
   crossing = 0;
   period = 0;
   height=(rect.Height()-24);
   midpoint=(rect.bottom-4)-(height/2);

   for (x=rect.left; signed int(x)<rect.right; x++) {

      if (i<num_samples) {
         y=unsigned int(midpoint-((Zoom*(sample_buffer[i++]
            -Offset))*height));
      } else y=0;

      if (init==0) {
         dc.MoveTo(x, y);
      } else {
         dc.LineTo(x-1, y);
         dc.LineTo(x, y);
      }

      if (y>top) top=double(y);
      if (y<bottom) bottom=double(y);

      if (init==0) init=1;

      /* freq calculation */
      mean = (top+bottom)/2;

      if (y<mean && y2>=mean) {
         if (crossingfound>2) {
            ncrossings++;
            period += abs(x-crossing);
         } else crossingfound++;

         crossing=x;
      }
      y2 = y;
   }

   if (ncrossings==0) freq=0;
   else freq = 8250000/(period/ncrossings);

   sprintf(status, "%s", DisplayString);
   dc.TextOut(rect.left, rect.top, status);
}
```

```
#define REPAINT(RESAMPLE)                                    \
{                                                            \
   ResampleOnPaint = RESAMPLE; CRect rect;                   \
   CWnd* pWnd=GetDlgItem(IDC_WAVE);                          \
   pWnd->GetWindowRect(&rect);                               \
   ScreenToClient(&rect);                                    \
   InvalidateRect(rect, TRUE);                               \
}

void CWaveformDlg::OnTimer(UINT nIDEvent)
{
   if (nIDEvent == ID_PAINT_TIMER && TimedRefresh!=0) {
      REPAINT(1)
   }

   CDialog::OnTimer(nIDEvent);
}

void CWaveformDlg::OnCancelMode()
{
   CDialog::OnCancelMode();
}

int CWaveformDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CDialog::OnCreate(lpCreateStruct) == -1)
      return -1;

   if (SetTimer(ID_PAINT_TIMER, 500, NULL) == 0)
   {
      AfxMessageBox("Failed to create timer");
      return -1;
   }

   return 0;
}

void CWaveformDlg::OnDestroy()
{
   CDialog::OnDestroy();
   KillTimer(ID_PAINT_TIMER);
}

void CWaveformDlg::OnAdc1s()
{
   DisplayDevice = FPADC_DEV_ADC_REMAINDER_SLOW;
   DisplayString = "REMAINDER ADC (slow)";
   LowVal = -1.0;
   MidVal = 0.0;
   HiVal =  +1.0;
   REPAINT(1)
}
```

```
void CWaveformDlg::OnAdc2s()
{
   DisplayDevice = FPADC_DEV_ADC_COARSE_SLOW;
   DisplayString = "COARSE ADC (slow)";
   LowVal = -1.0;
   MidVal = 0.0;
   HiVal =  +1.0;
   REPAINT(1)
}

void CWaveformDlg::OnAfpadc()
{
   DisplayDevice = FPADC_DEV_FPADC;
   DisplayString = "FPADC";
   LowVal = -1.0;
   MidVal = 0.0;
   HiVal =  +1.0;
   REPAINT(1)
}

void CWaveformDlg::OnAadccalib()
{
   DisplayDevice = FPADC_DEV_ADC_CALIBRATION;
   DisplayString = "CALIBRATION ADC";
   LowVal = -1.0;
   MidVal = 0.0;
   HiVal =  +1.0;
   REPAINT(1)
}

void CWaveformDlg::OnAdc1dma()
{
   DisplayDevice = FPADC_DEV_ADC_REMAINDER_FAST;
   DisplayString = "REMAINDER ADC (fast)";
   LowVal = -1.0;
   MidVal = 0.0;
   HiVal =  +1.0;
   REPAINT(1)
}

void CWaveformDlg::OnAdc2dma()
{
   DisplayDevice = FPADC_DEV_ADC_COARSE_FAST;
   DisplayString = "COARSE ADC (fast)";
   LowVal = -1.0;
   MidVal = 0.0;
   HiVal =  +1.0;
   REPAINT(1)
}

void CWaveformDlg::OnAuto()
```

```
{
   // enable auto-refresh
   if (TimedRefresh != 1) {
      TimedRefresh = 1;
      REPAINT(1)
   }
}

void CWaveformDlg::OnTrigger()
{
   // disable auto-refresh
   if (TimedRefresh == 1) {
      TimedRefresh = 0;
   } else {
      REPAINT(1)
   }
}

void CWaveformDlg::OnCancel()
{
   DisplayDevice = FPADC_DEV_NONE;
   DisplayString = "Null Source";

   CDialog::OnCancel();
}

void CWaveformDlg::OnDcfilter()
{
   DCFilterEnabled = m_DcFilter.GetCheck();
   REPAINT(1)
}

void CWaveformDlg::OnAutoscale()
{
   AutoScaleEnabled = m_AutoScale.GetCheck();
   REPAINT(1)
}

void CWaveformDlg::OnCalibrate()
{
   static int mode=2;
   char name[256], number[256];

   strcpy(name, "Lut (=");
   itoa(mode, number, 10);
   strcat(name, number);
   strcat(name, ")");
   m_CalibrateButton.SetWindowText(name);

   fpadc_lut_gen(mode++, &m_CommProgress);

   if (mode==MAX_LUT_GEN_MODE+1) mode=0;
```

```
}

void CWaveformDlg::OnSetOffset(NMHDR* pNMHDR, LRESULT* pResult)
{
   Offset = double(m_Offset.GetPos())/double(m_Offset.GetRangeMax()
      -m_Offset.GetRangeMin());
   controlsNeedUpdate = 1;
   REPAINT(0)
   *pResult = 0;
}

void CWaveformDlg::OnSetZoom(NMHDR* pNMHDR, LRESULT* pResult)
{
   Zoom = double(m_Zoom.GetPos())/double(m_Zoom.GetRangeMax()
      -m_Zoom.GetRangeMin())*9.0+1.0;
   controlsNeedUpdate = 1;
   REPAINT(0)
   *pResult = 0;
}

void CWaveformDlg::OnCaptureChanged(CWnd *pWnd)
{
   CDialog::OnCaptureChanged(pWnd);
}

void CWaveformDlg::OnCustomdrawOffset(NMHDR* pNMHDR, LRESULT* pResult)
{
   *pResult = 0;
}

void CWaveformDlg::OnCustomdrawZoom(NMHDR* pNMHDR, LRESULT* pResult)
{
   *pResult = 0;
}

void CWaveformDlg::OnSetSpeed(NMHDR* pNMHDR, LRESULT* pResult)
{
   fpadc_clock_divider = m_Speed.GetPos();
   fpadc_clock_divider = fpadc_clock_divider * fpadc_clock_divider;
   fpadc_write_cmd(SET_SAMPLING_CLK_DIVIDER, fpadc_clock_divider);
   controlsNeedUpdate = 1;
   REPAINT(0)
   *pResult = 0;
}

void CWaveformDlg::OnSave()
{
   CString File;
   FILE *FHandle;
   int i, savedTimedRefresh;

   CFileDialog FileDialog(false, "dat", "samples.dat",
```

```
      OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT, "dat");

   savedTimedRefresh = TimedRefresh;
   TimedRefresh = 0;

   if (FileDialog.DoModal() == IDOK)
   {
      File = FileDialog.GetPathName();
      FHandle = fopen(File, "wt");

      if (!FHandle) {
         ::MessageBox ( NULL, "Couldn't create file",
            "ERROR", MB_OK | MB_ICONHAND | MB_SETFOREGROUND );
      } else {
         for (i=1; i<NUM_SAMPLES; i++) {
            fprintf(FHandle, "%lf\n", sample_buffer[i]);
         }
         fclose(FHandle);
      }
   }

   TimedRefresh = savedTimedRefresh;
}

void CWaveformDlg::OnMux1()
{
   static int mode=0;
   char name[256], number[256];

   strcpy(name, "VgaMux (=");
   itoa(mode, number, 10);
   strcat(name, number);
   strcat(name, ")");
   m_Mux1Button.SetWindowText(name);
   fpadc_write_cmd(SET_VGA_GAIN, mode);

   mode++;
   if (mode==16) mode=0;
}

void CWaveformDlg::OnMux2()
{
   static int mode=0;
   char name[256], number[256];

   strcpy(name, "InMux (=");
   itoa(mode, number, 10);
   strcat(name, number);
   strcat(name, ")");
   m_Mux2Button.SetWindowText(name);
   fpadc_write_cmd(SET_IN_MUX, mode);
```

```
    mode++;
    if (mode==2) mode=0;
}

void CWaveformDlg::OnDacout()
{
    static int val=0;
    char name[256], number[256];

    strcpy(name, "DAC 0x");
    itoa(val, number, 16);
    strcat(name, number);
    m_DACOut.SetWindowText(name);
    fpadc_write_cmd(WRITE_OFFSET_DAC, val);

    val+=0x1000;
    if (val>=0x10000) val=0;
}

void CWaveformDlg::OnReleasedcaptureBits(NMHDR* pNMHDR, LRESULT*
pResult)
{
    fpadc_num_bits = m_Bits.GetPos();
    fpadc_write_cmd(SET_ADC_MASK, bitflip16((1<<fpadc_num_bits)-1));
    controlsNeedUpdate = 1;
    REPAINT(0)

    *pResult = 0;
}
```