
Université d'Ottawa · University of Ottawa

# Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Shumin SHEN

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

M. Sc. (Systems Science)

GRADE - DEGREE

Systems Science Program

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

A Floating-Point Analog-to-Digital Converter

V. Groza

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

A. El Saddik

E. Petriu

J-M. De Koninck, Ph D

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

DEAN OF THE FACULTY OF GRADUATE AND POSTODORAL STUDIES

# A Floating-Point Analog-to-Digital Converter

By

Shumin Shen

A Thesis
Presented to School of Graduate Studies and Research
In partial fulfillment of the
Requirements for the degree of

Mater of Science

Master program in Systems Science
University of Ottawa

Ottawa, Ontario, Canada, 2004

*Your file Votre référence*
*ISBN: 0-494-01607-8*
*Our file Notre référence*
*ISBN: 0-494-01607-8*

# Abstract

The floating-point analog-to-digital (A/D) converter has been proposed for achieving large dynamic range signals. The architecture of this class of A/D converters handles separately the dynamic range and resolution such that high-resolution imposed by wide dynamic range can be avoided.

This thesis studies the floating-point analog-to-digital converter (FP-ADC). The first attempt is to analyze the parallel architecture of the floating-point converter, which is our research base. The characteristics and specifications of the floating-point A/D converter are described. Simulations of the parallel architecture of the floating-point A/D converter were conceived, run and presented here to support the theoretically derived FP-ADC transfer characteristics.

After analyzing the parallel architecture of the floating-point A/D converter, the following work is to provide a way of minimizing the conversion time as well as keeping the precision of the floating point A/D converter (FP-ADC) by implementing the parallel architecture with Field Programmable Gate Arrays (FPGA). This method makes full use of the remarkable precision of the uniform quantizers, as well as minimizing the conversion time similar to the speed of nonlinear converters. That is, it can achieve a higher acquisition rate while preserving the highest possible resolution. The thesis also gives the performance analysis of the FPGA-based floating-point analog-to-digital converter.

The thesis presents the design and practical implementation of the parallel FP-ADC, based on a FPGA and other hybrid components-of-the-shelf. The correctness of the design was verified by computer simulation, while the functionality of the implemented FP-ADC was tested on a test bench controlled by a PC.

# Acknowledgement

As with most large projects, this research would not have been possible without considerable guidance and support. I would like to acknowledge those who have enabled me to complete this work and my years of graduate study.

I would like to express my sincere gratitude to my supervisor Professor Dr. Voicu Groza for his valuable advices and guidance and for his unfailing patience with me during the course of this research.

I would also like to thank my professors and colleagues for the valuable discussions we had and for enduring my persistent questioning.

Finally, I would like to acknowledge the patience and understanding of my family and friends during my studies.

# Acronyms

| | |
|---|---|
| A/D | Analog-to-Digital |
| DMA | Direct Memory Access |
| DNL | Differential Nonlinearity |
| ENOB | Effective Number of Bits |
| FP-ADC | Floating-Point Analog-to-Digital Converter |
| FPGA | Field Programmable Gate Array |
| GCU | Gain Control Unit |
| INL | Integral Nonlinearlity |
| LSB | Least Significant Bit |
| LUT | Look-Up-Table |
| MSB | Most Significant Bit |
| PDF | Probability Density Functions |
| PGA | Programmable Gain Amplifier |
| PLL | Phase Locked Loop |

SNR Signal-to-Noise

UART Universal Asynchronous Receiver and Transmitter

VGA Variable Gain Amplifier

# Table of Contents

# Table of Figures

# List of Tables

# Chapter 1

# Introduction

Analog-to-digital (A/D) converters are the key elements of any system that uses digital techniques to communicate the analog "real world". They provide the link between the analog world and digital systems. Since the extensive use of analog and mixed analog-digital operations, A/D converters often appear as the bottleneck in the data processing applications, limiting the overall speed or precision. New technology applications such as digital signal processing and telecommunication need high-resolution, high-speed A/D converters at very low power consumption and the minimum silicon cost. Current CMOS trends show supply voltages decrease faster than the relevant device mismatch parameters. Therefore the design of high-resolution converters is continually becoming more difficult. It is often simply impossible to achieve high enough resolution to cover the full dynamic range.

The rapidly proliferating use of digital signal processing has resulted in a steadily increasing demand for higher sampling rates and lower power dissipation in high-resolution A/D converters, which are applied in the areas such as wireless telecommunication systems, test and measurement equipments. [2][6] Floating-point A/D converters have been shown to be a very useful means of providing a large dynamic range in applications by decoupling the signal resolution and dynamic range specifications and alleviating the problems of noise limited power consumption and

device mismatch tolerances. Thus, they have been historically used in the instrumentation of experiments where the acquired signal is of a non-repeatable nature. Furthermore, they have been specifically used for large dynamic range data acquisition in high-energy physics instrumentation such as electromagnetic calorimeters for detectors in colliding-beam machines. [8][11][14]

The conversion time and precision are the two main problems in the floating-point A/D converter. With the demand of high-resolution and high speed A/D converters in the new technology applications, how to minimize the conversion time as well as keeping the precision of the floating-point A/D converter have recently caught researchers' attention. The literature review of the floating-point A/D converter is discussed in Chapter 2.

## 1.1 Motivation

The motivation of the work that follows is to provide a way of minimizing the conversion time as well as keeping the precision of the floating point A/D converter (FP-ADC) by implementing the parallel architecture with Field Programmable Gate Array (FPGA). This method can make full use of the remarkable precision of the uniform quantizers as well as minimizing the conversion time similar to the nonlinear converters. That is, it can achieve a higher acquisition rate while preserving the highest possible resolution. The reason why FPGA is used to implement the parallel architecture is elaborated in chapter 2.

In pursuing our objectives above, MATLAB is used in the analyzing of the characteristics of the parallel architecture of FP-ADC and the simulation of FP-ADC in

In pursuing our objectives above, MATLAB is used in the analyzing of the characteristics of the parallel architecture of FP-ADC and the simulation of FP-ADC in statistical view. Moreover, the MATLAB serial port interface is used to directly access the hardware device (ALTERA APEX DSP board) via RS232 port.

The FPGA-based system design has two main elements: two 10-bit A/D converters (Analog Devices Inc., AD9203) and ALTERA AP20KE200EBC652-1X FPGA. ALTERA Quartus® Version 2.0 is used for VHDL programming, simulation, synthesis, place and routing.

## 1.2 Synopsis

### 1.2.1 Contributions

The results presented in this thesis offer one approach to minimizing the conversion time as well as keeping the precision of the floating point A/D converters by implementing the parallel architecture with Field Programmable Gate Array (FPGA) in ALTERA EP20K200EBC652-1X.

The principal contributions of this research are:

1. Development of a VHDL program that can realize the implementation of the parallel architecture of the floating-point A/D converter and the UART (universal asynchronous receiver and transmitter) serial communication between PC and FPGA via RS232 serial port.

2. Design and characterization of the circuits, which operate two 10-bit A/D converters (AD9203) with maximum sampling rate of 40 MSPS and a PGA (programmable Gain amplifier).

3. The performance comparison of the simulation of the parallel architecture of the floating-point A/D converter in statistic view and the FPGA hardware implementation.

4. Proposal of the predictive FPGA implementation floating-point A/D converter and its performance evaluation.

These contributions are reviewed in the final chapter of this work.

### 1.2.2 Thesis Overview

The thesis is organized as follows:

Chapter 2 begins with an introduction of the floating-point number systems that includes IEEE floating-point standards and the literature review of a collection of the floating-point A/D conversion techniques. A thorough survey of the general architectures of the floating A/D converters is given. The comparison of characteristics of these floating-point A/D converters is presented.

Chapter 3 introduces the theory knowledge of the parallel architecture of the floating-point A/D converter. The characteristics and specifications of the floating-point A/D converter are described. And the simulation of the parallel architecture of the floating-point A/D converter is presented in statistic view.

Chapter 4 presents the architecture of the FPGA system's components. The detailed description of the architecture of the system components is discussed. And the hardware circuits to implement the parallel architecture FP-ADC with PGA and two A/D converters are designed and characterized.

Chapter 5 gives the detailed analysis of the testing. The performance comparison of the simulation of hardware implementation of FP-ADC and that of FP-ADC in statistical view is discussed, and the experimental testing results with MATLAB analysis are presented.

Finally, Chapter 6 summarizes the work presented and suggests areas for future works.

# Chapter 2

# Background Material

This chapter begins with an introduction on floating-point number systems, which includes IEEE floating-point standards. Many excellent texts [1][2][3][4] exist on the topic of the floating-point numbers and their properties, and may be consulted for additional details. The remainder of the chapter focuses on a literature review of a collection of floating-point analog-to-digital conversion techniques that are used to efficiently obtain signals within a high dynamic range. Before that, the basic A/D converter specifications are introduced.

## 2.1 Floating-point number systems

The evolution of floating-point representation in computers has a long history. There were no standards for how floating-point numbers were stored and how standardized floating-point number calculations were performed in the earlier days. With the availability of the computer hardware to make floating-point calculations faster by the 1970s, the major problem with floating point calculations of that time was the lack of standards. Different manufacturers' floating-point libraries and hardware supported different standards for rounding, approximating, performing calculations, handling exceptions. This diversity made it difficult to write portable codes.

Inspired by Dr. William Kahan's research of the costs associated with different ways of representing and calculating using floating-point numbers, many companies including Intel, Motorola, IBM and DEC formed a committee under the aegis of IEEE to standardize floating-point arithmetic in computers. This was the IEEE 754 standard.

In floating-point number systems, any number $X$ can be represented as

$$X = sign(X) \cdot M \cdot \beta^e \tag{2-1}$$

Where $sign(X)$ is the signum-function, $M$ is called mantissa, $e$ is called exponent and $\beta$ is the base of the floating-point number system.

Within the IEEE standard, exponent is defined by

$$e = \lfloor \log_\beta |X| \rfloor \tag{2-2}$$

Where $\lfloor . \rfloor$ is the floor-function. With the definition, mantissa is in the interval $M \in [1, \beta)$

When $\beta = 2$, binary floating-point numbers are represented as

$$X = sign(X) \cdot M \cdot 2^e \tag{2-3}$$

Where mantissas are limited to interval $M \in [1,2)$, exponents are then defined

$$e = \lfloor \log_2 |X| \rfloor \tag{2-4}$$

### 2.1.1 The IEEE floating-point Standard

IEEE Standard 754 is a standard for binary floating-point arithmetic[3][4], which specifies number formats, basic operations, conversions, and exceptional conditions.

The basic number format sizes in IEEE 754 for representing floating-point values are 32 bits (single precision) and 64 bits (double precision). Here, we only discuss the single-precision floating numbers. The 32 bits used in single precision are divided into three separate groups: bits 0 through 22 forms the mantissa, bits 23 through 30 form the exponent, and bit 31 is the sign bit.

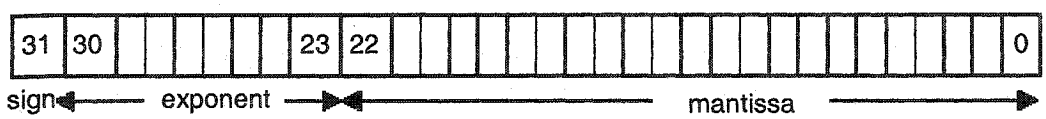


Figure 2.1 IEEE format of floating point numbers

These bits form the floating point number, $X$ , by the following relation:

$$X = (-1)^S \cdot M \cdot 2^{BE-127} \tag{2-5}$$

The term $(-1)^S$ simply means that the sign bit, $S$ , is 0 for a positive number and 1 for a negative number. The variable, $BE$ , is number between 0 and 255 represented by the eight exponent bits. Subtracting 127 from this number allows the exponent term to run from $2^{-127}$ to $2^{128}$. The number 127 is a bias for IEEE single-precision floats. Because the exponent field needs to represent both positive and negative exponents, a

*bias* is added to the actual exponent in order to get the stored exponent. Thus, an exponent of zero means that the number 127 is stored in the exponent field. A stored value of 200 indicates an exponent of (200-127), or 73. For reasons discussed, exponents of -127 (all 0s) and +128 (all 1s) are reserved for special numbers.

The mantissa, $M$ , is formed from the 23 bits as a binary fraction. It represents the precision bits of the number. It is composed of an implicit leading bit and the fraction bits. In order to maximize the quantity of representable numbers, floating-point numbers are typically stored in the *normalized* form, where the radix point is basically put after the first non-zero digit. Normalized representation of binary mantissa is generally used, that is, there is only one nonzero digit to the left of the decimal point (called a binary point in base 2). Since the only nonzero number that exists in base two is 1, the leading digit in the mantissa will always be a 1, and therefore does not need to be stored. As a result, the mantissa has effectively 24 bits of resolution, by way of 23 fraction bits.

The 23 stored bits, referred to by the notation: $m_{22}$, $m_{21}$, $\ldots\ldots m_0$, form the mantissa in normalized representation according to:

$$M = 1.\, m_{22}\, m_{21}\, m_{20} \ldots\ldots m_1\, m_0 . \tag{2-6}$$

In other words, $M = 1 + m_{22} * 2^{-1} + m_{21} * 2^{-2} + m_{20} * 2^{-3} + \ldots\ldots$ (2-7)

Zero is treated as a special number. It is not directly representable in the straight format due to the assumption of a leading 1. For zero, the exponent and mantissa bits are all zeros. The sign bit could be '1' or '0'. Note that -0 and +0 are distinct values, though they both compare as equal.

If the exponent is all 0s, but the fraction is non-zero, then the value is called *denormalized*, which does *not* have an assumed leading 1 before the binary point. Thus, this represents a number $(-1)^{s} \cdot 0.M \cdot 2^{-127}$, where $s$ is the sign bit and $M$ is the fraction.

### 2.1.2 Floating-point number properties

In floating-point representation, mantissa and exponent correspond to accuracy and dynamic range respectively. A finite precision binary floating-point number can be represented with $m+E$ bits, where $m$ is the number of mantissa bits and $E$ is the number of exponent bits. As discussed before, the first bit of the mantissa in the normalized representation for binary floating-point is assumed leading 1 and not represented. And the sign bit represents the signed number. Therefore, signed mantissas with $m$ bits accuracy are represented with $m$ bits.

The range of numbers of a floating-point number system is determined by the smallest and the largest numbers and can be represented with the full accuracy of the number system. The number range of the binary floating-point is

$$\frac{1}{2}2^{e_{\min}} \le |X| < 2^{e_{\max}} \tag{2-8}$$

Where the minimum and the maximum exponents are usually chosen to be $e_{\min} = -2^{E-1}+1$ and $e_{\max} = 2^{E-1}$.

The dynamic range of the floating-point number system is defined by the ratio of the largest and smallest number in (2-8).

Numbers not in number range (2-8) are called either under- or overflowed.

## 2.2 Analog-to-digital converter specifications

Analog-to-digital converter is a device that converts the continuous-time signals into discrete-time, binary-coded form. Full specifications of the performance of A/D converters require a large number of parameters, some of which are defined differently by different manufactures. The discussion below follows a number of important parameters. This section attempts to provide a general understanding of A/D converter specifications. For a complete set of specifications, the literatures [36][37] and manufacturer's data sheets can be referred to.

### 2.2.1 Basic A/D conversion relationship

Perhaps the most fruitful way of indicating the relationship between analog and digital quantities involved in a conversion is to plot a graph. Figure 2.2 shows the graph of quantization function $y = Q(X)$ of an ideal $b$-bit A/D converter, ($b = 3$). Since all values of the analog input are presumed to exist, they must be quantized by partitioning the continuum into 8 discrete ranges. All analog values within a given range are represented by the same digital code, which corresponds to the nominal mid-range value.

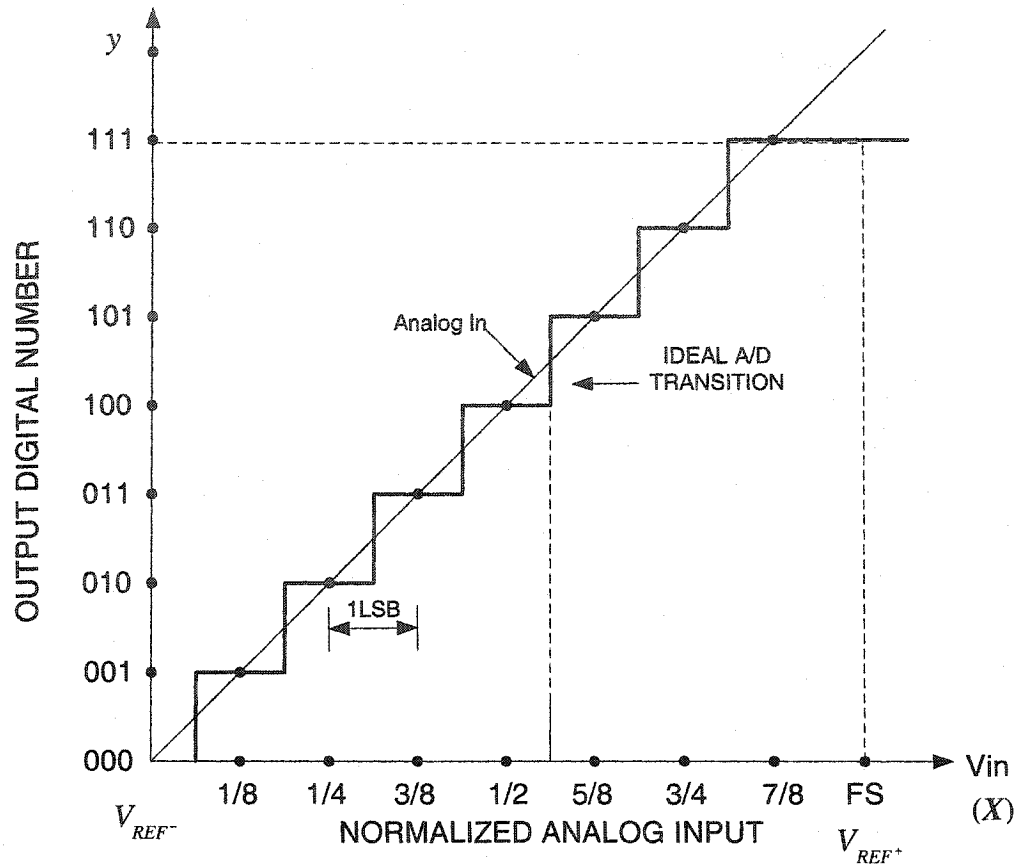


Figure 2.2 Conversion ideal relationships in a 3-bit A/D converter

### 2.2.2 Operational characteristics of A/D converter

#### 2.2.2.1 Full-Scale Range

Full-Scale Range $V_{FS}$ is the input range of voltages over which the A/D converter will digitize that input. For example: $V_{REF^+} = 3.5V$ and $V_{REF^-} = 1.5V$, $V_{FS} = V_{REF^+} - V_{REF^-} = 3.5V - 1.5V = 2.0V$

#### 2.2.2.2 Quantization Step

Quantization Step (Δ) is the magnitude of the step size of A/D converters transition functions. It also refers to the bit of the digital output code that has the smallest weight, Least Significant Bit (LSB).

For a ***b***-bit converter, the Quantization Step or LSB is defined as $\Delta = LSB = \frac{V_{FS}}{2^b}$.

Since the biggest binary number that can be represented with ***b*** bits is $2^b$-1, the largest value of the analog input signal that can be converted correctly to digital is $(2^b - 1) \cdot \Delta = V_{FS} - \Delta$, i.e., $V_{REF^+} - \Delta$ for $V_{REF^-} = 0$

#### 2.2.2.3 Most Significant Bit

The Most Significant Bit (MSB) is the bit of the digital output code that has the largest weight. Its value is the half of the full-scale range.

#### 2.2.2.4 Input Bandwidth

Input Bandwidth specifies the highest frequency analog signal, which an A/D converter can convert and still meet its performance specifications. Usually, A/D converters are required to convert signals with bandwidths of up to ½ their sampling rate (Nyquist rate). A/D converters must use anti-aliasing filters to remove all signals above the Nyquist frequency.

### 2.2.3 A/D amplitude response descriptive terminology

The descriptive terminology for A/D converter describes the characteristics of an ideal function and the actual function.

The quantization characteristic conveniently represents the ideal A/D conversion function: the specific input/output amplitude mapping between the continuous set of input amplitudes and the discrete set of output amplitudes. The output amplitudes are called quantization levels.

The input transition levels are the measured input levels at which the output transitions occur. The threshold level associated with a given output code-word transition is a suitably defined level corresponding to that output transition. The quantization level is defined as the amplitude levels midway between adjacent threshold levels.

A uniform A/D converter is characterized by equal spacing between adjacent ideal threshold levels, whereas a nonuniform A/D converter has an ideal threshold level spacing, which varies with threshold level. Figure 2.3 illustrates the two commonly used ideal quantization characteristics. A quantization level with value zero distinguishes the MIDTREAD characteristic (Figure 2.3(b)), and the MIDRISER characteristic (Figure 2.3(a)) has a threshold level with value zero. Unlike the MIDRISER characteristic, the MIDTREAD characteristic gives an output that is insensitive to the infinitesimal input changes about zero and is therefore generally preferred. Both characteristics introduce a maximum error of $\frac{\Delta}{2}$, where $\Delta$ is the step between adjacent threshold levels corresponding to the given quantization step.

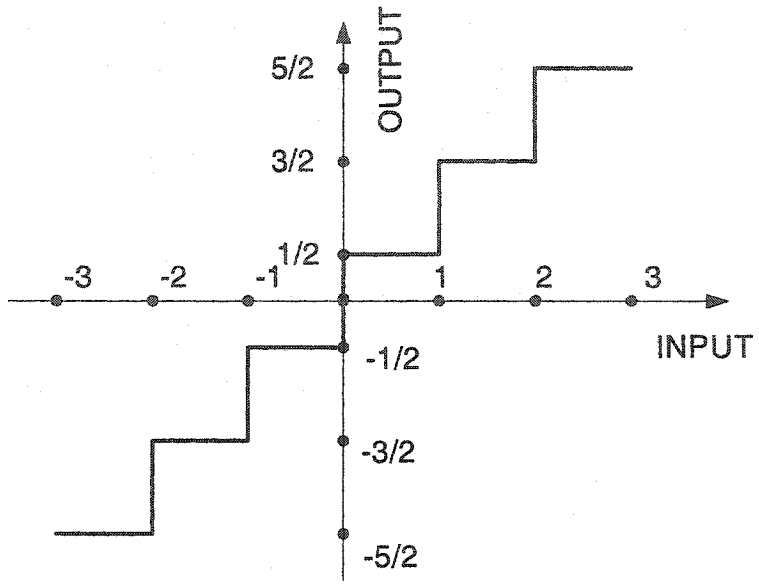


Figure 2.3 (a) MID-RISER A/D quantization characteristics

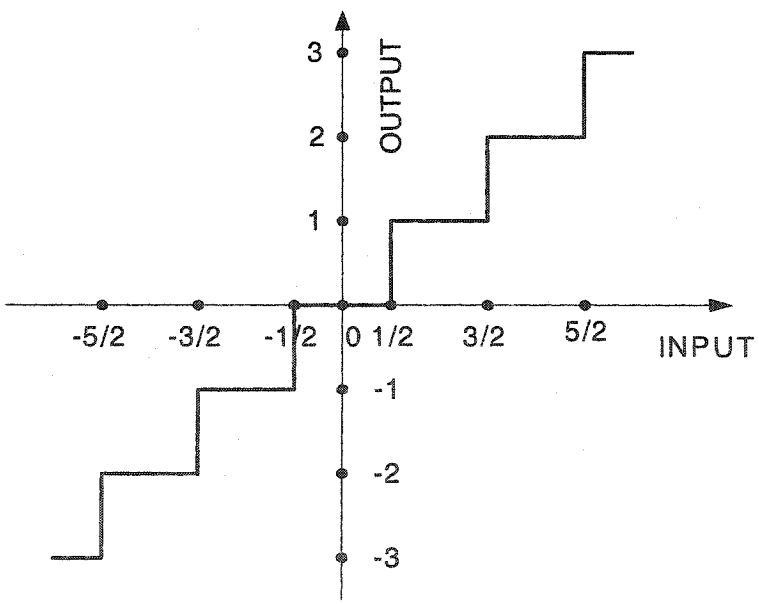


Figure 2.3 (b) MID-TREAD A/D quantization characteristics

### 2.2.4 Accuracy parameters of A/D converters

#### 2.2.4.1 Resolution

Resolution is the smallest analog increment corresponding to 1 LSB (Least Significant Bit). For converters, resolution is normally expressed in bits ($b$), where the numbers of analog levels is equal to $2^b$ bits.

#### 2.2.4.2 Dynamic Range

Dynamic Range is the ratio of the largest input that can be converted to the smallest step size of the converter. For example, an 8-bit A/D converter with an input range of zero to four volts has a quantization step size of $\Delta = (4V - 0V)/2^8 = 15.625mV$. Therefore, the dynamic range ratio is $V_{FS}/\Delta = 4.0V/15.625mV = 256 = 2^8$. This number can also be expressed in decibels as $20\log 256 = 48dB$, or it can be expressed as a power of 2 (number of bits), that is, 8-bit A/D converter.

#### 2.2.4.3 Quantization Error

Quantization Error ($\varepsilon$) is the error inherent in all A/D conversions. If $Q(X)$ is the analog equivalent of the quantized signal, then the quantization error is defined as

$$\varepsilon = Q(X) - X \tag{2-9}$$

Since even an "ideal" converter has finite resolution, any analog voltage that falls between two adjacent output codes will result in output code that is inaccurate by up to ½

LSB, i.e., $\varepsilon \in (-\Delta/2, \Delta/2)$. For the quantization error of the floating-point analog-to-digital converter, Chapter 3 will discuss it in details.

#### 2.2.4.4 Signal-to-noise Ratio

Signal-to-noise (SNR) is the ratio of the signal power to the total noise power at the output (usually measured for a sinusoidal input). The Signal-to-noise represents the dynamic performance of A/D converters.

As known, there is a quantization error denoted by $\varepsilon$ in the approximation effect of A/D converters. To formulate the impact of quantization noise on the performance, $\varepsilon$ is assumed that (1) it is a random variable uniformly distributed between $-\frac{\Delta}{2}$ and $+\frac{\Delta}{2}$, and (2) independent of the analog input. The quantization noise power can then be expressed as the mean square of $\varepsilon$ :

$$\varepsilon^2 = \frac{1}{\Delta}\int_{-\frac{\Delta}{2}}^{+\frac{\Delta}{2}} \varepsilon^2 d\varepsilon \approx \frac{\Delta^2}{12} \tag{2-10}$$

For a $b$ -bit A/D converter, the peak signal-to-noise ratio $SNR$ at the output is:

$$SNR = \frac{2^{2b-3}\cdot\Delta^2}{\Delta^2/12} = \frac{3}{2}\cdot 2^{2b} \tag{2-11}$$

Which, when expressed in decibels, becomes $SNR = 6.02\cdot b + 1.76$ dB

The SNR of the floating-point analog-to-digital quantizer will be discussed in Chapter 3.

#### 2.2.4.5 Effective Number of Bits

Effective Number of Bits (ENOB) is a measure of overall A/D performance under dynamic conditions. Due to the quantization noise, an ideal [n] bit A/D converter will have an effective number of bits that is less than n. Cumulative effects of many errors, missing codes, integral nonlinearity – all contribute to a lower effective number of bits. ENOB can be calculated from the signal-to-noise (SNR) obtained from the dynamic testing.

$$ENOB = \frac{SNR - 1.76}{6.02} \tag{2-12}$$

For a perfect 12-bit A/D converter, $ENOB = \frac{72dB - 1.76}{6.02} = 11.68$ bits.

#### 2.2.4.6 Differential Nonlinearity Error

Differential Nonlinearity (DNL) is a measure of how uniform the transfer function step sizes are. Each step size is compared to the ideal step size. Any difference in magnitude is DNL. The ideal step size is LSB.

#### 2.2.4.7 Integral Nonlinearity Error

Integral Nonlinearlity (INL) is the maximum deviation of the input/output characteristic from a straight line passed through its end point (Figure 2.4). The overall difference plot is called the INL.

Figure 2.4 shows INL, DNL and the following two definitions. These terms describe the static behavior of A/D converters.

Offset Error identifies the deviation from the ideal location of the lowest transition level on the A/D converter transfer function.

Gain Error is a measure of the deviation from the ideal of the slope of the A/D converter transfer function. There are several ways that manufacturers determine the slope of the transfer function. Figure 2.4 shows gain error specified as the deviation from the ideal location of the highest transition level.

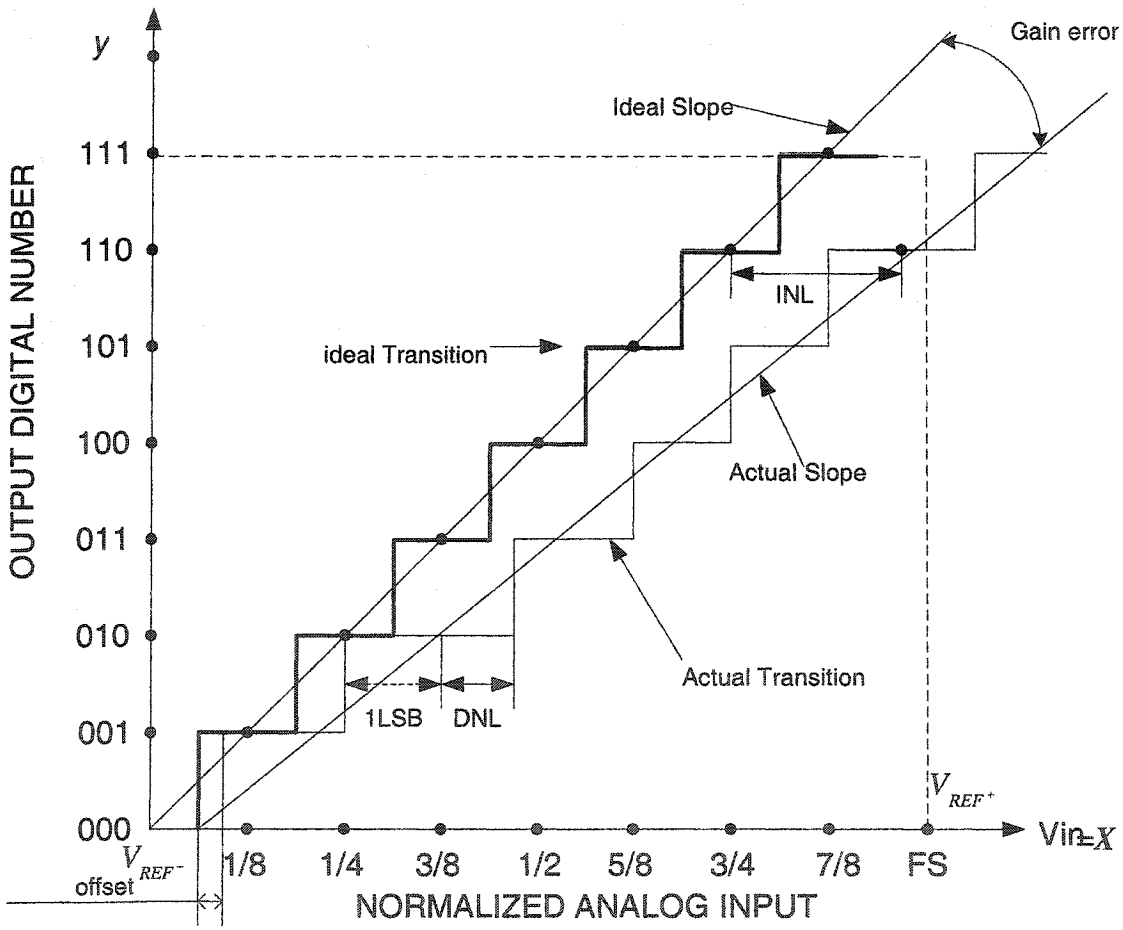


Figure 2.4 Illustration of DNL, INL, Offset, Gain error

## 2.3 Floating-point quantizer properties

The benefits obtained with floating-point number representation in digital signal processing are large dynamic range and constant signal-to-noise ratio (SNR) over a wide dynamic range. SNR is defined by the ratio of the signal power and the noise power: $SNR(dB) = 10\log\left(\frac{\sigma_x^2}{\sigma_\varepsilon^2}\right)$, where $\sigma_x^2$ is the variance of the signal, and $\sigma_\varepsilon^2$ is the variance of the noise.

Signal-to-noise ratio for a floating–point quantizer $SNR_{float}(dB)$ is similarly defined in [2] and [6]:

$$SNR_{float}(dB) = 10\log\left(\frac{\sigma_x^2}{\sigma_\varepsilon^2}\right) \tag{2-13}$$

Roundoff errors are inevitable in realizations of digital signal processing due to the quantization of the data samples to a finite accuracy. In floating-point rounding, mantissas and exponents are rounded separately. In the basic analysis of roundoff error, it is assumed that there are no under- or overflows from the number range. Then roundoff error is resulted only due to rounding of mantissa, because exponents are integers. Roundoff errors in floating-point cases can be modeled as multiplicative errors. Relative roundoff error is defined as

$$\varepsilon_r = \frac{\varepsilon}{X} = \frac{X_q - X}{X} = \frac{(M_q - M)\cdot 2^e}{M\cdot 2^e} = \frac{M_q - M}{M} = \frac{M_e}{M} \tag{2-14}$$

Where $M_q$ and $M$ are quantized and infinite precision mantissa, respectively, and $M_e$ is mantissa roundoff error. Mantissa error is assumed to be independent of mantissa.

The quantized finite precision floating-point number $X_q$ is:

$$X_q = X \cdot (1+\varepsilon_r) = X + \varepsilon_r \cdot X \tag{2-15}$$

Where $X$ is the infinite precision number and $\varepsilon_r$ is the relative roundoff error.

Because the relative roundoff error $\sigma_{\varepsilon_r}^2$ is a multiplicative error and it was assumed to be independent of the signal, the variance of the roundoff error is:

$$\sigma_\varepsilon^2 = \sigma_x^2 \cdot \sigma_{\varepsilon_r}^2 \tag{2-16}$$

$$SNR_{float}(dB) = 10\log\left(\frac{\sigma_x^2}{\sigma_x^2 \cdot \sigma_{\varepsilon_r}^2}\right) = 10\log\left(\frac{1}{\sigma_{\varepsilon_r}^2}\right) \tag{2-17}$$

The distribution of quantization error can be derived when the probability density functions (PDF) of mantissa error Me and mantissa M are known. The distribution of mantissa error depends on the rounding strategy chosen. If symmetric rounding is used, mantissa error can be assumed to be uniformly distributed in the interval $\left[-\frac{\Delta}{2}, \frac{\Delta}{2}\right)$, where $\Delta = 2^{-m}$ is the quantization step of normalized mantissas.

The variance of relative error when it possesses a certain PDF is $\sigma_{\varepsilon_r}^2 = \frac{\Delta^2}{8 \cdot \ln 2}$. How to derive this equation can be referred to [2] and [6].

Therefore, $SNR_{float} = 6.02 \cdot m + 7.44$ (2-18)

For floating-point numbers, the number of significant bits (mantissa bits) is constant over the dynamic range. The number of exponent bits defines the dynamic range. As is shown in equation (2-14), when the signal is scaled down, it has no remarkable effect on the $SNR_{float}$. It is constant.

SNR for a bipolar fixed-point quantizer with $(b_{fix} + 1)$ bits is called $SNR_{fixed}(dB)$, which is in [7, pp122-123].

The quantization error $\sigma_\varepsilon^2 = \frac{\Delta^2}{12} = \frac{\left(\frac{V_{FS}}{2^{b_{fix}+1}}\right)^2}{12} = \frac{V_{FS}^2}{48 \cdot 2^{2 \cdot b_{fix}}}$ (2-19)

$$SNR_{fixed}(dB) = 10 \log 10\left(\frac{48.2^{2b_{fix}} \sigma_x^2}{V_{FS}^2}\right) = 6.02 b_{fix} + 16.81 - 20 \log 10\left(\frac{V_{FS}}{\sigma_x}\right) \quad (2\text{-}20)$$

Where $V_{FS}$ is the full-scale amplitude of the quantizer. It can be seen that $SNR_{fixed}$ is not constant depending on the actual signal distribution.

The comparison between accuracies of 15+1 bit fixed-point (1 bit for sign) and 12+4 bit floating-point with normalized number representation can be seen in Figure 2.5. Mantissa is expected represented with 12 bits, while exponent with 4 bits. It is easily seen

that the $SNR_{float}$ is almost constant when the number of mantissa bits is certain, while the $SNR_{fixed}$ increases when the signal is scaled down. The consequence of this difference in $SNR$ can be consulted in reference [2].

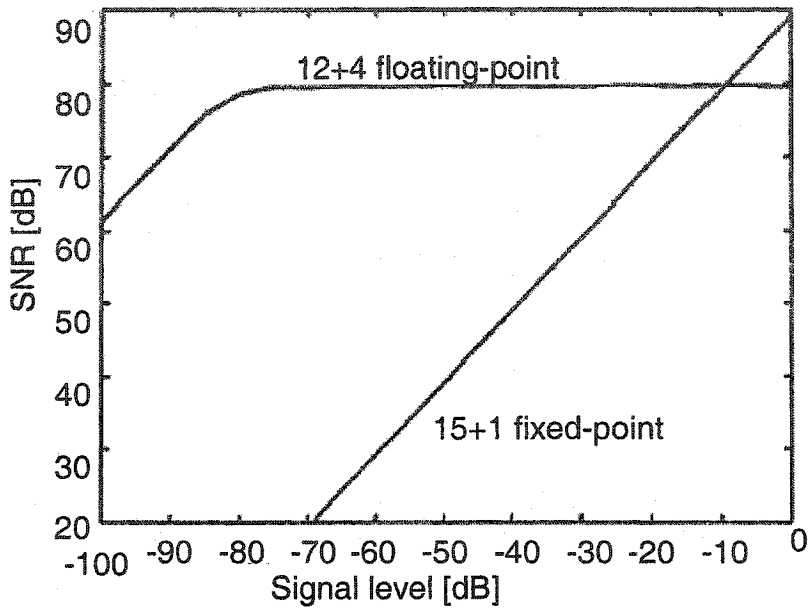


Figure 2.5 SNR as a function of signal level for Gaussian noise.

## 2.4 Review of the floating-point analog-to-digital converters

There has long been the need for low cost, high-resolution and high dynamic range A/D converters in many applications such as process control, other low or medium-speed applications, wireless communications systems, instrumentation for high-energy physics experiments, radar, and test and measurement equipment.[7][8][9][10] However, many applications do not require high resolution at large amplitudes, that is, they do not require high-resolution and high dynamic range simultaneously. In fact, many times a large

number of bits (high-resolution) are required simply to provide dynamic range rather than higher absolute accuracy at all levels.

In addition, low power A/D converters are required for many power sensitive applications. Modern CMOS trends toward lower supply voltages are reducing the available signal range, increasing the required sampling capacitances to achieve a certain signal-to-noise ratio (SNR). Therefore, the design of high-resolution converters is becoming more difficult. In high-speed converters, it is often not possible to achieve a high enough resolution to cover the full dynamic range. [12] In these cases, it would be rational if the A/D converters offer the same resolution for large and small signals within the dynamic range in order to reduce power and cost.

Based on the properties of floating-point number systems, a floating-point analog-to-digital (A/D) converter has been proposed for this purpose of achieving large dynamic range signals. It is an approach to a wide dynamic range where dynamic range and resolution are handled separately so that high-resolution imposed by wide dynamic range can be avoided.

Floating-point A/D converters can decouple the conversion precision and dynamic range specifications and alleviate the problems of noise limited power consumption and device mismatch tolerances. In floating-point representation, accuracy (mantissa) and dynamic range (exponent) are controlled independently. In a floating-point A/D converter, signal resolution is kept relatively constant over a dynamic range in the same manner as a floating-point number representation.

There are many types of floating-point A/D converters: charge integration, cyclic or algorithmic, and floating-point A/D converters employing a VGA (variable gain amplifier) followed by uniform A/D converters, etc. [13] However, they can be generally classified into two large "classic" approaches.

One method is based on non-uniform quantizers where the input signals are quantized to steps that are spaced non-uniformly. One way is to do the A/D conversions based on the close resemblance of floating-point and logarithmic number representations. This method is similar to standard logarithmic, $\mu$-law or A-law companding techniques.[5] This is the known logarithmic amplifier solution. The floating-point A/D conversion method consists of several parts: analog logarithmic nonlinearity amplifier, $1+b_{fix}$ bit fixed-point A/D converter and Look-Up-Table (LUT) that performs the conversion from logarithmic to floating-point. The amplifier is used for compressing the signal amplitude in order to extend the dynamic range and a LUT is used for producing linear digital output code. The exponent is obtained directly from the integer part of logarithmic representation, and the mantissa is obtained from the fractional part of the logarithmic representation by calculation with LUT.

This classic nonuniform quantizer method has its own sets of advantages and disadvantages. As we discussed before, many applications do not require high resolution at large amplitudes. In fact, many times a large number of bits is required simply to provide dynamic range, not necessarily to provide higher accuracy at all levels. For standard uniform A/D converters, there may not be any systemic advantage to quantizing the large signals as finely as the small ones and there may also be a significant increase of

the cost to producing a large number of bits. So, for the same number of bits and equal dynamic range, it is possible that the non-uniform method can perform well at the system level. It could overcome the problem of the redundancy of digital data at higher voltages that is a problem in the high-resolution linear A/D converters when increasing the number of bits (i.e. lowering the quantizing steps) to attempt to improve the signal to quantizing error ratio with the increasing input voltages. However, because of the non-uniform nature of the quantization, the precision of nonlinear A/D converters that implement these quantizers is not outstanding, and the quantization errors are still to be worsened by the finite length of registers used in the later process of floating-point coding.

A second classic method is based on a uniform quantization A/D converter connected to the acquired signal through a programmable gain amplifier (PGA). There are two types of such FP-ADCs. One is called sequential floating-point A/D converter or two-cycle floating-point A/D converter. The detailed information can be obtained from the references [7][8][9][17][18][19][20][23]. Figure 2.6 shows the flowchart of floating-point A/D conversion. It produces two parts of digital output data consisting of an exponent and a mantissa. According to the input range, the full-scale range of the conversion is determined by the value of the exponent and this full-scale value is converted with resolution corresponding to the bits of mantissa.

In [7][8][9], the floating-point A/D technique employs the variable gain amplification of the input voltage. The gain of the variable gain amplifier (VGA) varies in the steps by powers of 2, depending upon the input voltage range, which attempt to

bring the amplified signal in the upper-half of the converter range. The only difference among [7], [8] and [9] is the hardware implementation, that is, FP-ADC in [7] is with a self-calibration circuit D/A converter, FP-ADC in [8] is with VGA and FP-ADC in [9] is with a cyclic charged-balancing algorithm instead of hardware.

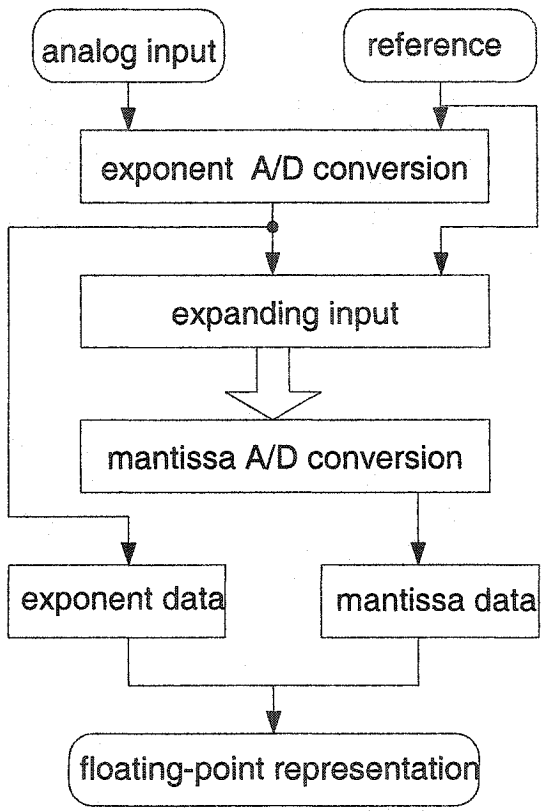


Figure 2.6 Flowchart of the floating-point A/D conversion

The architecture of the sequential floating-point A/D converter is shown in Figure 2.7. It consists of an analog-to-digital converter, a programmable amplifier (PGA) and a gain control unit (GCU). The A/D converter is a midtread uniform quantizer. The quantization step is $\Delta$. The full-scale input range is $[-V_{FS}/2, V_{FS}/2)$. The mantissa $y_m$ is expressed with $m$ bits, and the exponent $e$ is expressed with $E$ bits. In a typical approach of the floating-point A/D converter, the conversion employs two basic operations in

addition to uniform quantization. First, the range of the magnitude of the input signal is detected from the A/D output and the range is encoded as an $E$ bit exponent. Then, the PGA gain is set correspondingly and the signal is scaled according to the value of the exponent. After that, the quantization determines the $m$ bit mantissa. The total resolution of the converter is $E+m$ bits.

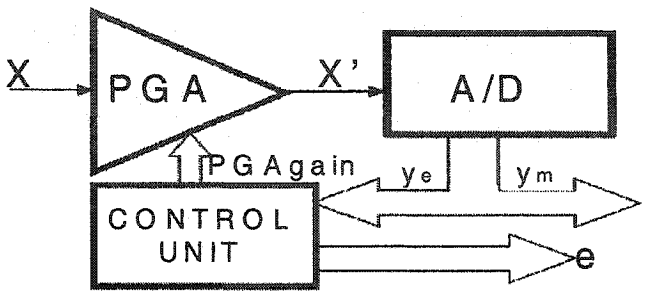


Figure 2.7 Block diagram of the sequential floating-point A/D converter

The PGA gain is adjusted accordingly, to bring the input voltage in the range of ½ full-scale to full-scale value. This ensures the optimal use of the converter's full-scale range. In the first cycle, the input voltage is converted as it is and the PGA gain is decoded from this value $y_e$, also determining the exponent part. In the next cycle, the input signal is amplified by the PGA so that the output signal from PGA is in the upper half of A/D conversion range. A second cycle conversion is carried out to measure the mantissa. The conversion is performed at this gain to determine the mantissa part.

This solution has its advantages and disadvantages. The advantages are that it preserves the high precision of uniform A/D converters and avoids separate coding stages. But the obvious drawbacks are that the conversion time is doubled and the sampling rate is halved. The architecture of the pipelined floating-point A/D converter

presented in [23] shortens the conversion time by overlapping the fine second quantization cycle over the previous coarse first quantization. Moreover, a reduced resolution is needed for exponent acquisition. The architecture with two A/D converters (one for coarse quantization and the other for fine quantization) was proposed in [17], [18], and [25]. This solution can further speed up the conversion process. In this solution, the two quantizers are still connected in the cascade and the two quantization cycles are executed separately. The speed improvement is mainly obtained from using a variant of flash A/D converters for the exponent acquisition. In general, the sequential architecture of FP-ADC doubles the conversion time even by using the fastest A/D conversion techniques.

Based on a uniform quantizer that preserves the precision of the sequential floating-point A/D converter, while minimizing the conversion time close to one of the non-uniform one-cycle quantizers, a new architecture of the floating-point analog-to-digital converter is proposed. This architecture is called the parallel architecture of the floating-point analog-to-digital converter. The architecture of this kind of the floating-point A/D converter combines the advantages of the above two "classic" floating-point A/D converters. It consists of two A/D converters that work simultaneously: one determines the exponent, while the other one, which is connected to the quantizer input over a programmable gain amplifier (PGA), finds the mantissa. The PGA gain is based on the prediction of the input signal. If the predicted exponent coincides with the currently acquired one, the conversion result is delivered immediately; if not, the mantissa is acquired again with the PGA gain reset to the most recently acquired exponent. This can be referred to in the references [20], [21], and [22]. Obviously this kind of the floating-

point A/D converter can make full use of the remarkable precision of the uniform quantizers and the minimum conversion time of the non-uniform quantizers. Figure 2.8 shows the architecture. The details are discussed in Chapter 3.

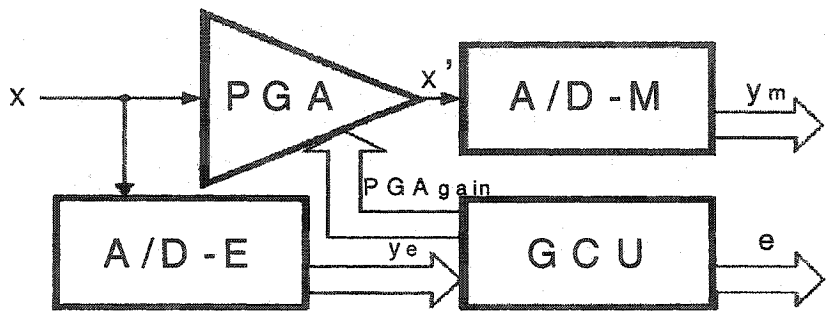


Figure 2.8 Block diagram of parallel floating-point A/D converter

This new parallel architecture of the floating-point analog-to-digital converter is our research base. The implementation of this kind of FP-ADC is resorted to Field Programmable Gate Arrays (FPGAs) to minimize the conversion speed.

FPGA devices are field programmable and are based on Flash, SRAM, EEPROM or Anti-Fuse connectivity. The most successful FPGA devices are based on SRAM. This is because the other memory types mentioned above are less dense in terms of area than SRAM. In addition, there are some types of connectivity that are one-time programmable (i.e. Anti-Fuse) so they are not very flexible.

FPGA development is, in some way, similar to ASIC development. The program language for FPGA development can be HDL (Hardware Description Language). The most common HDLs used for FPGA Design are Verilog and VHDL. After describing an

FPGA design in an HDL, a Synthesizer tool can effectively convert Verilog or VHDL into specific primitives, which exist in an FPGA family. FPGA devices allow rapid design prototyping. They offer more dense logic and less tedious wiring work than discrete chip designs and faster turnaround than the standard cell, or full-custom design fabrication. Due to the outstanding speed of FPGA, the implementation of the parallel architecture of the floating-point analog-to-digital converter with FPGA can effectively increase the sample acquisition rate.

## 2.5 Summary

This chapter presents an introduction to the floating-point number systems including IEEE floating-point standards. Furthermore, the basic A/D converter specifications are discussed which give general technical information about the A/D converter. The literature review of a collection of the floating-point analog-to-digital conversion techniques is discussed. A thorough survey of the general architectures of the floating A/D converters is given. The two main popular techniques of the floating-point A/D converters are introduced. Since there are some disadvantages in these main techniques, the new parallel architecture of the floating-point A/D converter is proposed, which is the main foundation of this research work.

# Chapter 3

# Floating-Point Analog-to-Digital Converter

This research work refers to a new parallel architecture of floating-point analog-to-digital converters, which employs uniform quantizers to keep the precision of the sequential floating-point A/D converter and minimizes the conversion time close to the characteristics of the non-uniform one-cycle quantizer. The basic knowledge of this kind of the floating-point A/D converter is introduced in this Chapter. The characteristics and specifications of the floating-point A/D converter are also described. The simulation of the parallel floating-point A/D converter is presented in statistic view.

## 3.1 Floating-point analog-to-digital converter

### 3.1.1 Architecture

The architecture of the parallel floating-point A/D converter consists of two A/D converters that work simultaneously: one determines the exponent; while the other one, which is connected to the quantizer input over a programmable gain amplifier (PGA), finds the mantissa. The PGA gain is based on the prediction of the input signal. If the current acquired exponent coincides with the predicted one, the conversion result is

delivered immediately; if not, the mantissa is acquired again and the PGA gain is reset to the most recently acquired exponent. This converter architecture is called parallel floating-point A/D converter. [18] It is easily seen that this kind of floating-point A/D converter can make full use of the high precision of the uniform quantizers and the minimum conversion time of the non-uniform quantizers.

The architecture of the floating-point A/D converter is shown in Figure 3.1. [18]

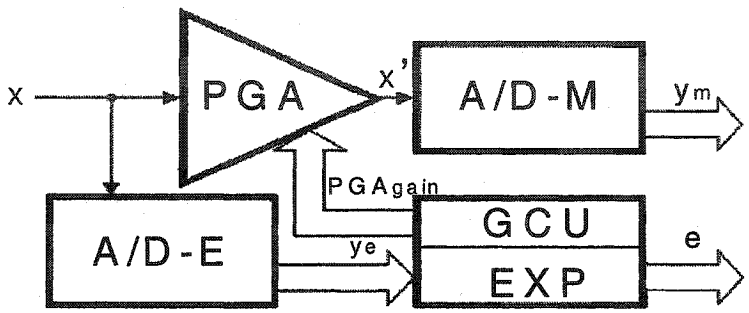


Figure 3.1 Block diagram of the parallel floating-point A/D converter

It consists of two linear A/D converters (A/D-E and A/D-M), one programmable gain amplifier (PGA), one gain control unit (GCU) and one exponent unit (EXP). The A/D-M measures the mantissa, while A/D-E performs a coarse conversion that is used to determine the exponent of the quantized signal. Both A/D converters have the same input range $[-V_{FS}/2, V_{FS}/2)$.

Both conversion cycles are executed in parallel: the A/D-E executes the coarse conversion ($y_e$); simultaneously, the A/D-M determines the mantissa ($y_m$). The current PGA gain is set to a value that is predicted by the Gain Control Unit (GCU) on the base

of the history of the quantized signal ( $X$ ). The GCU algorithm aims to maintain the amplified signal ( $X$ ') in the most significant half of the A/D converter's input quantization range, to minimize the measurement relative error.

The exponent $e$ is calculated in EXP from $y_e$. Based on the PGA gain, if this value coincides with the predicted one, that is, the current quantized signal $X$ ' is in the upper part of the measurement domain of A/D-M, the floating-point A/D converter provides the correct exponent and mantissa. If the predicted PGA gain drives the signal ( $X$ ') at the input of A/D-M out of the measuring domain, the overflow conversion value will increase the quantization overload noise; if the signal ( $X$ ') is in the lower part of the A/D-M input domain, the granular quantization noise is increased.

What are the quantization overload noise and the granular quantization noise?

As we know, a N-point quantizer $Q$ may be defined by specifying a set of $N+1$ decision levels $X_0$, $X_1$,..., $X_N$ and a set of $N$ output points $y_1$, $y_2$,..., $y_N$. When the value $X$ of an input sample lies in the $i$-th quantizing interval, namely $R_i = \{X_{i-1} < X < X_i\}$, the quantizer produces the output value $y_i$. The end levels $X_0$ and $X_N$ are chosen to be equal to the bounds of the Full-Scale Range $V_{FS}$ i.e., the smallest and largest values respectively.

The quantization process can be modeled as the addition of a random noise component $\varepsilon = Q(X) - X$ to the input sample as indicated in Figure 3.2. The quantization noise $\varepsilon$ is often approximated as being independent of the input samples when the number of levels is large.

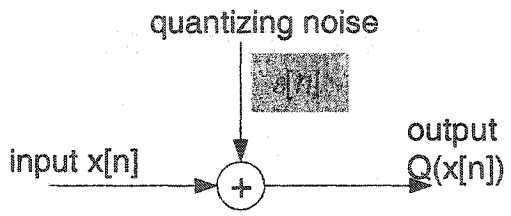


Figure 3.2 Additive noise model of the A/D quantizer

When the input sample lies within the interval $X_1 \leq X \leq X_{N-1}$, the output noise is described as granular noise. When the input lies outside this interval, the output is described as overload noise. In section 3.2.2, quantization error is discussed in details.

### 3.1.2 GCU algorithm

Gain Control Unit (GCU) algorithm implements the control path of the conversion process, which consists of two concurrent tasks: getting the exponent and the mantissa. The A/D-E and the A/D-M converters (Fig. 3.1) carry out these two processes, respectively. GCU determines the ***E*** bits exponent part and the correspondent PGA gain. The PGA gain is adjusted to a value that was previously predicted based on the history of the acquired signal to bring the output of the PGA in the range of ½ full-scale to full-scale value. This ensures the optimal use of the A/D-M converter's full-scale range to get the best accuracy of the A/D-M $m$ bit resolution. The gain prediction of the parallel floating-point A/D converter can be approached by the polynomial regressive extrapolation. [17] The most simple prediction is given by a zero-order extrapolator: if the absolute value of the current quantized signal $X' = X \cdot gain^{t-1}$ is in the ½ full-scale to the full-scale value of the A/D-M conversion range, the floating-point quantization is correct

and the A/D-E has acquired the same exponent as the preceding sample $gain^t = gain^{t-1}$ ; the GCU generates the termination logic signal (EOC) to indicate the end of conversion. If the previously predicted PGA gain ($gain^{t-1}$) has made the signal $X'$ out of the conversion range ($|X| > V_{FS}$) or in the lower half of the range of the A/D-M ($|X| < V_{FS}/2$), the GCU sets the PGA gain to the current acquired exponent $gain^t$ and A/D-M repeats the conversion to get the correct mantissa. After the second conversion, the A/D-M is back to the initial state and the termination signal (EOC) is generated.

Depending on the input voltage range, the PGA gain varies as powers of 2, attempting to bring the input of A/D-M in the upper-half of the converter range. Therefore, if the exponent part has $\boldsymbol{E}$ bits, it shall have $2^E$ distinct values, each of which represents a unique gain from 1 to $2^{2^E-1}$. The maximum gain is used when the input signal is less than $(V_{FS}/2)/(2^{2^E-1})$. This scaled up input signal is then digitized into $\boldsymbol{m}$-bit mantissa part through A/D-M according to the GCU theory, leading to a dynamic range of $m+2^E$ bits. In order to get the most of the resolution, $m$ was chosen such as $m = 2^E$.

For example, let us consider an input signal $V_{in} \in$ [0, 8v), with the number of bits that are used to represent the exponent $E$=2, and $m$=4 - the number of bits that are used to represent the mantissa. The PGA gain spans a range from 1 to $2^3 = 8$ (3 being the largest number that can be represented with $E$ = 2 bits) and it is found out from the output of A/D-E, a linear ADC that has a resolution of 4 bits. The PGA gain is set to maximum 8 when the input voltage is smaller than the quantization step $\Delta = \frac{8V}{2^4} = 0.5V$. Since the

A/D-M decision levels are resolved with a theoretic quantization error of $\pm\Delta/2 = \pm 0.25V$, a safety margin of 0.25V should be considered for the exponent calculation as the input signal value is in the bins of $2^e\Delta$. Each value of the exponent/gain defines a measurement domain with a precision of $\pm\ \Delta/2$. To avoid that the amplified signal exceeds the quantization limits, the measurement domains are established with a pre-emptive policy, as illustrated by the example of Table 3-1

Table 3.1: Floating-point converter's 2-bit exponent and 4 bit mantissa

| $V_{in}$ (Input voltage) | Gain | Exponent $e$ ($e_0e_1$) | Mantissa $M$ ($m_3m_2m_1m_0$) |
|---|---|---|---|
| 0 – 0.25v | 8 | 00 | 0000~1111 |
| 0.25-1.75v | 4 | 01 | 0000~1111 |
| 1.75-3.75v | 2 | 10 | 0000~1111 |
| 3.75-8v | 1 | 11 | 0000~1111 |

## 3.2 Characteristics of the floating-point A/D converter

The dynamic range, the signal-to-quantization noise ratio ($SNR$) and the conversion time are the most important characteristics of the floating-point A/D converter. The exponent is represented by $E$ bits, the mantissa is by $m$ bits. The performance of the parallel floating-point A/D converter with $E$ bits exponent and $m$

bits mantissas is compared with that of the fixed-point uniform quantizer with $b$ bits resolutions in the following.

### 3.2.1 Dynamic Range

Dynamic range is the ratio of the largest input that can be converted to the smallest step size of the converter. The floating-point A/D converter's dynamic is defined as the ratio between the largest admissible quantizer input signal and the smallest one that gives a nonzero quantized acquisition when the magnitude of the input signal is bounded to the quantization range. The exponent bits determine the dynamic range. For the bipolar signals, the two's complement code is usually used in uniform midtread quantizers to represent them. The largest positive number and the smallest negative number expressed with $b$ bits in two's complement are respectively:

$$y^{+} = 2^{b-1} - 1 \tag{3-1}$$

$$y^{-} = -2^{b-1} \tag{3-2}$$

When an acceptable approximation of $y^{+} \approx \left| y^{-} \right|$ is considered, the dynamic range of a $b$-bit resolution A/D converter can be expressed by:

$$D_{fix} = 2^{b-1} \tag{3-3}$$

In order to compare the floating-point quantizer and the fixed-point quantizer, the resolution of the fixed-point quantizer $b$ is assumed to be equal to that of the floating-point quantizer ($E + m$).

$$b = E + m \tag{3-4}$$

To express the exponent $e$ as an $E$-bit number, the A/D converter has to have a resolution $m \geq 2^E - 1$, with $m$ and $E$ natural numbers.

There is a relationship between the dynamic range of the floating-point A/D converter ($D_{floating}$) and that of the fixed-point quantizer ($D_{fix}$). $D_{floating}$ can be expressed by $D_{fix}$ in equation (3-5).

$$D_{floating} = 2^{2^E-1} \cdot 2^{m-1} = 2^{2^E-1} \cdot 2^{b-E-1} = 2^{2^E-E-1} \cdot D_{fix}$$

$$\frac{D_{floating}}{D_{fix}} = 2^{2^E-E-1} \tag{3-5}$$

A floating-point A/D converter can acquire signals with higher dynamics, compared with a uniform quantizer that has the same resolution. For example, for $b = 12$ bit uniform quantizer and $E + m = 12$ floating-point A/D converter, if the exponent is expressed by a 3-bit number, $E = 3$, $\frac{D_{floating}}{D_{fix}} = 16$; if the exponent $E = 4$, $\frac{D_{floating}}{D_{fix}} = 2048$.

It is easily seen that the dynamic range of the floating-point A/D converter is much higher than that of the uniform quantizer with the same resolution.

The dynamic range can also be expressed in decibels as $10\log 2^{E+m}\ dB$.

### 3.2.2 Quantization Error

Quantization error is the natural error that occurs when a signal is converted from analog to digital domain. The approximation or "rounding" effect in A/D converters is called quantization, and the difference between the original input and the digitized output is called the quantization error and is denoted by $\varepsilon$. This error decreases when the resolution increases, and its effect can be viewed as additive noise called quantization noise appearing at the ADC output.

To formulate the impact of quantization noise on the performance, $\varepsilon$ is assumed that (1) it is a random variable uniformly distributed between $-\frac{\Delta}{2}$ and $+\frac{\Delta}{2}$, and (2) independent of the analog input. The quantization noise power can then be expressed as the mean square of $\varepsilon$:

$$\varepsilon^2 = \frac{1}{\Delta}\int_{-\frac{\Delta}{2}}^{+\frac{\Delta}{2}} \varepsilon^2 d\varepsilon \approx \frac{\Delta^2}{12} \tag{3-6}$$

The input-output characteristic $Q(x)$ of the quantizer is ordinarily chosen to have odd symmetry with the symmetry assumption.

To minimize the overload noise of a uniform quantizer, its input range is usually set to an interval that reduces the probability of a quantized signal falling outside it.

A symmetric uniform quantizer can be fully described by specifying the number of quantization levels and either the step size $\Delta$ or the overload range of $V_{FS} = X_N = -X_0$. To avoid significant overload distortion, the overload range is chosen

to be a suitable multiple $L = V_{FS} / \sigma$ called the loading factor, as a function of the root mean square of the signal $\sigma$. A common choice is the four-sigma loading where $L = 4$. That is, the signal normally distributed with mean 0 and variance $\sigma$ is quantized with this uniform quantizer having an input range of $[-4\sigma, 4\sigma]$. [34][36] Because the total span of the quantizing range is $V_{FS} = 8\sigma$, there are $2^b$ levels in that range, with the step size:

$$\Delta = \frac{8 \cdot \sigma}{2^b} = \sigma \cdot 2^{-b+3} \tag{3-7}$$

As is known, the uniform quantizer is a basic common component to many A/D converters. It is assumed that the fixed-point quantizer has a $b$-bit resolution. The step size of the fixed-point quantizer is: $\Delta = V_{FS} / 2^b$.

The quantizer output level is always the nearest discrete neighbor level to the input, in a range of $\pm \Delta / 2$. The quantization function $Q(X)$ of a $b$ - bit resolution quantizer can be summarized by the following expression (3-8):

$$Q(X) = \begin{cases} \left(2^{b-1} - 1\right) \cdot \Delta ; & \text{for } X \geq \left(2^{b-1} - 1\right) \cdot \Delta - \dfrac{\Delta}{2} \\ k \cdot \Delta \quad \text{for } k \cdot \Delta - \dfrac{\Delta}{2} \leq X < k \cdot \Delta + \dfrac{\Delta}{2}, \text{ and } k \in \left\{\left(-2^{b-1} + 1\right), \ldots\ldots, \left(2^{b-1} - 2\right)\right\}; & \\ -2^{b-1} \cdot \Delta ; & \text{for } X < -2^{b-1} \cdot \Delta + \dfrac{\Delta}{2} \end{cases} \tag{3-8}$$

The quantizer absolute error is defined by

$$\varepsilon = Q(X) - X \tag{3-9}$$

The relative error is defined by

$$\varepsilon_r = \frac{\varepsilon}{X} = \frac{Q(X) - X}{X} \tag{3-10}$$

The quantizer output can be written as the sum of its input and the quantizer noise. Thus, in particular, if the input is a sequence of samples $x_n$, then $\varepsilon_n = Q(x_n) - x_n$ will be of interest.

Note that except for the inputs in the top or bottom regions of the input range, the magnitude of the error $\varepsilon$ is bound above by $\Delta/2$. If the input is confined to the ADC quantization range, i.e., $X \in \left[-2^{b-1}\cdot\Delta - \frac{\Delta}{2}, \left(2^{b-1}-1\right)\cdot\Delta + \frac{\Delta}{2}\right]$, the magnitude error is no larger than $\Delta/2$. If the input is outside of this range, then the quantizer magnitude error will be larger than $\Delta/2$ and the quantizer is called "overloaded". The range $X \in \left[-2^{b-1}\cdot\Delta - \frac{\Delta}{2}, \left(2^{b-1}-1\right)\cdot\Delta + \frac{\Delta}{2}\right]$ will be called no-overload range. In a $b$-bit quantizer, the no-overload range clearly has size $2^b\Delta$.

Let's normalize the quantizer output and input by $\Delta$ and rewrite (3-10):

$$\varepsilon_n = \frac{\varepsilon}{\Delta} = \frac{Q(X)}{\Delta} - \frac{X}{\Delta} \tag{3-11}$$

where $\varepsilon_n$ is the normalized quantization error.

The system of the normalized quantization error is:

$$\varepsilon_n = \begin{cases} \dfrac{X}{\Delta} - \left(2^{b-1} - 1\right); & \text{for } \dfrac{X}{\Delta} \geq \left(2^{b-1} - 1\right) - \dfrac{1}{2} \\ \dfrac{X}{\Delta} - k \quad \text{for } k - \dfrac{1}{2} \leq \dfrac{X}{\Delta} < k + \dfrac{1}{2}, \text{ and } k \in \left\{\left(-2^{b-1} + 1\right), \ldots\ldots, \left(2^{b-1} - 2\right)\right\}; & \\ \dfrac{X}{\Delta} - \left(-2^{b-1}\right); & \text{for } \dfrac{X}{\Delta} < -2^{b-1} + \dfrac{1}{2} \end{cases} \tag{3-12}$$

If the input is confined to the no-overload region, then the previous formulas can be abbreviated to:

$$\varepsilon_n = \frac{X}{\Delta} - k \text{ for } k - \frac{1}{2} \leq \frac{X}{\Delta} < k + \frac{1}{2}, \text{ and } k \in \left\{\left(-2^{b-1} + 1\right), \ldots\ldots, \left(2^{b-1} - 2\right)\right\}; \tag{3-13}$$

As known, every real number $a$ can be uniquely written in the form $a = \lfloor a \rfloor + < a >$ where $\lfloor a \rfloor$ is the greatest integer less than or equal to $a$, $< a >$ is the fractional part of $a$, $0 \leq a < 1$.

For the floating-point A/D converter with $E$-bit exponent and $m$-bit mantissa, the quantization function of the linear uniform A/D converter used to implement the floating-point A/D converter is given by the following equation (3-14):

$$y_m = \begin{cases} \left(2^{m-1}-1\right)\cdot\Delta \quad ; & \text{for } X > \left(2^{m-1}-1\right)\cdot\Delta - \dfrac{\Delta}{2}; \\ \left\lfloor \dfrac{X+\dfrac{\Delta}{2}}{\Delta} \right\rfloor \cdot \Delta \quad ; & \text{for } X \in \left[-2^{m-1}\cdot\Delta + \dfrac{\Delta}{2}, \left(2^{m-1}-1\right)\cdot\Delta - \dfrac{\Delta}{2}\right]; \\ -2^{m-1}\cdot\Delta \quad ; & \text{for } X < -2^{m-1}\cdot\Delta + \dfrac{\Delta}{2}; \end{cases} \quad (3\text{-}14)$$

From the equation (3-14), the absolute error in the non-overload region, i.e., $X \in \left[-V_{FS}/2, V_{FS}/2\right)$, can be expressed by:

$$\varepsilon = \frac{\Delta}{2} - \Delta \cdot \left\langle \frac{X+\dfrac{\Delta}{2}}{\Delta} \right\rangle \quad (3\text{-}15)$$

The equation of the normalized quantization error can be derived:

$$\varepsilon_n = \frac{\varepsilon}{\Delta} = \frac{1}{2} - \left\langle \frac{X+\dfrac{\Delta}{2}}{\Delta} \right\rangle \quad (3\text{-}16)$$

The relative error of the uniform quantizer is given in equation (3-17)

$$\varepsilon_r = \frac{\varepsilon}{X} = \frac{\Delta}{X}\left( \frac{1}{2} - \left\langle \frac{X+\dfrac{\Delta}{2}}{\Delta} \right\rangle \right). \quad (3\text{-}17)$$

For a normalized input range of $[-V_{FS}/2, V_{FS}/2) = [-4\sigma, 4\sigma) = [-4, 4)$, the normalized quantizer error $\frac{\varepsilon}{\Delta}$ of a uniform quantizer with $b$=4 and that of a floating-point A/D converter with $E = 2$ and $m = 4$ are shown in the Figure 3.3.

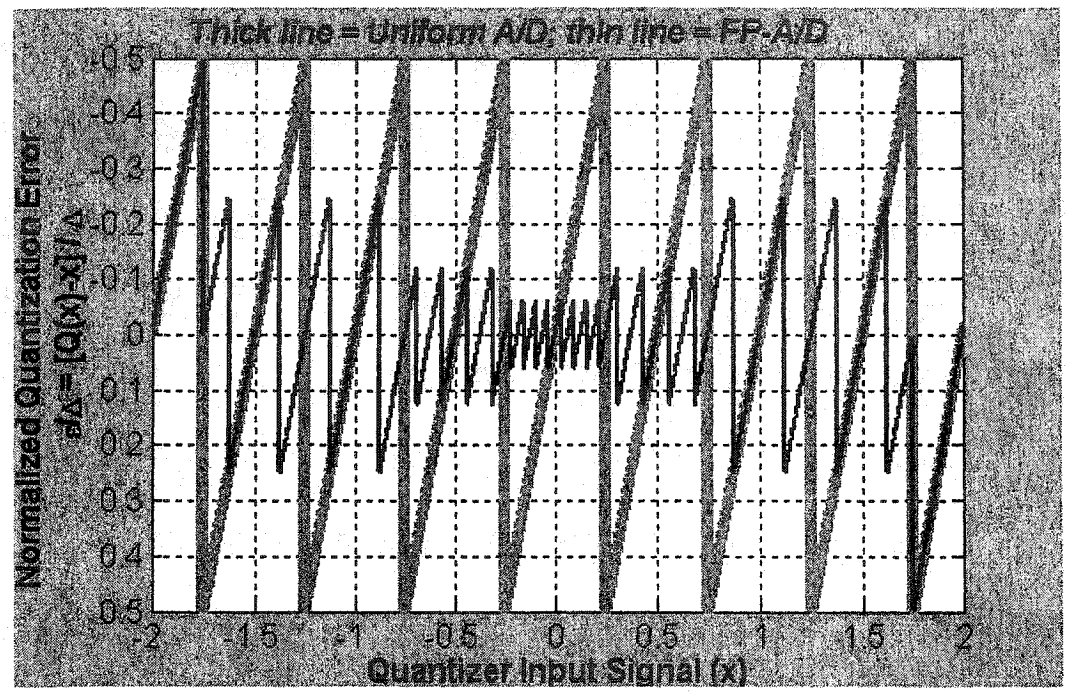


Figure 3.3 Normalized quantization errors: Uniform A/D and FP-A/D converters

As we discussed before, the magnitude of the absolute error $\varepsilon$ is no larger than $\Delta/2$ if the quantizer input range is confined to the range $X \in [-V_{FS}/2, V_{FS}/2]$. Because both quantizers have the same error beyond the interval [-2, 2], Figure 3.3 shows only the region [-2, 2]. It is easily seen that the floating-point A/D converter has a smaller quantization error than a uniform A/D converter.

The relative error $\varepsilon_r$ of the floating-point A/D converter has superiority over that of the uniform A/D converter. Figure 3.4 represents $\varepsilon_r$ of the two kinds of quantizers.

Because the smallest quantization step of the floating-point A/D (FP-A/D) converter is $\min \Delta_e = \Delta / 2^{\max E} \leq \Delta$, the highest value of the relative error $\varepsilon_r$ of the FP-A/D converter is exhibited only in a short interval $[-(\min \Delta_e)/2, (\min \Delta_e)/2]$ of Figure 3.4. It is also seen that the relative error $\varepsilon_r$ of the FP-ADC is markedly smaller than that of the uniform quantizer through the interval [-2, 2].

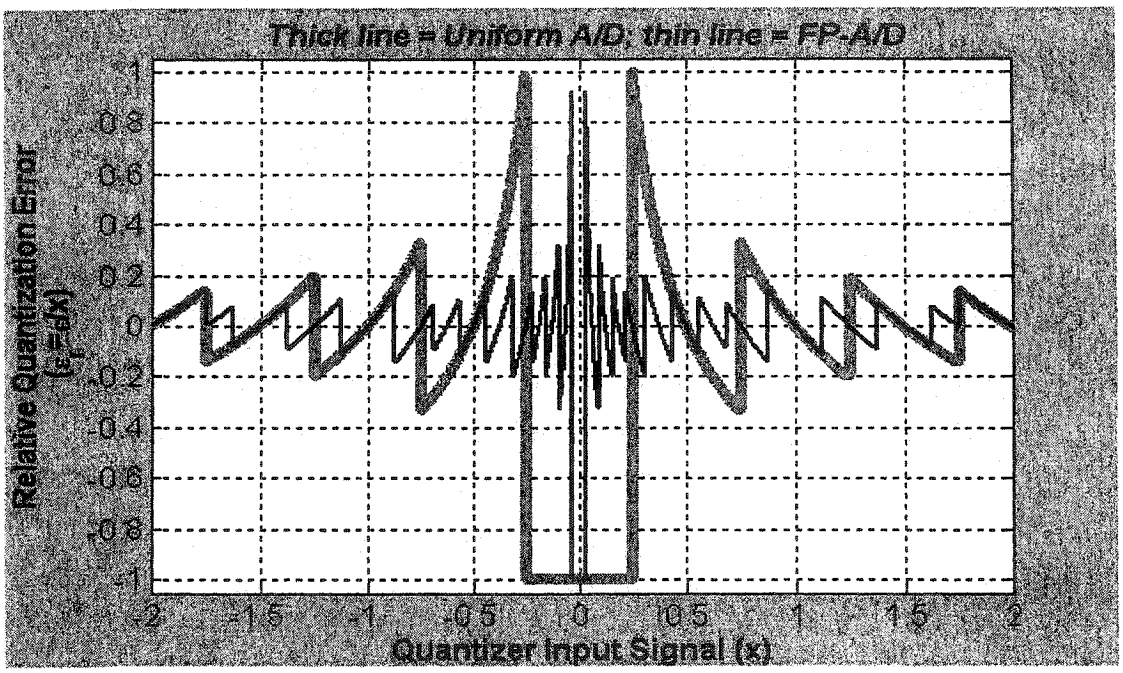


Figure 3.4 Relative quantization error $\varepsilon_r$: Uniform A/D and FP-A/D converters

### 3.2.3 Signal-to-Quantization Noise Ratio

Based on the model of Figure 3.2, the signal-to-quantization noise ratio (SNR) is defined by the ratio of the input signal power and the quantization noise power.

If the analog input is a sinusoid with amplitude $V_{FS}/2$, its total power is equal to $(V_{FS}^2/4)/2 = 2^{2b} \cdot \Delta^2/8$, where $V_{FS}$ is the input full-scale voltage quantization range, $b$

is a $b$-bit binary number of digital output, $\Delta = V_{FS} / 2^b$ is the quantization step, i.e., the minimum change in the input that can cause a change in the output and that corresponds to the least significant bit of the digital representation.

As discussed before, the quantization noise power is: $\varepsilon^2 \approx \frac{\Delta^2}{12}$.

Thus, peak signal-to-noise ratio $SNR$ at the output is:

$$SNR_p = \frac{2^{2b-3} \cdot \Delta^2}{\Delta^2 / 12} = \frac{3}{2} 2^{2b} \tag{3-18}$$

This equation is often used to describe the performance of a given $b$-bit uniform quantizer if the condition for the quantization noise to be uniform is observed. Formulating the above the equation, the signal-to-noise ratio of the uniform quantizer can be expressed in terms of its resolution $b$, as given by equation (3-19).

$$SNR_u = 3 \cdot 4^{b-2} \tag{3-19}$$

The input signal's probability density function $p(x)$ characterizes its statistical behavior. If the quantization step on the input range $\left[V_{REF^-}, V_{REF^+}\right]$ of the floating-point A/D converter is $\Delta_e$, the variance of the FP-ADC quantization error $\sigma_\varepsilon^2$ is given by (3-20).

$$\sigma_\varepsilon^2 = \int_{-\infty}^{\infty} \left[Q(x) - x\right]^2 \cdot p(x) \cdot dx = \int_{-\infty}^{V_{REF^-}} \left[-2^{m-1} \cdot \Delta - x\right]^2 \cdot p(x) \cdot dx +$$

$$+\sum_{e=0}^{2^E-1}\left\{\sum_{k=0}^{2^m-1}\left[\int_{x_k}^{x_{k+1}}\left(y_k\cdot\frac{2^e}{2^{2^E-1}}-x\right)^2\cdot p(x)\cdot dx\right]\right\}+\int_{V_{REF^+}}^{\infty}\left[(2^{m-1}-1)\cdot\Delta-x\right]^2\cdot p(x)\cdot dx \quad (3\text{-}20)$$

Where $[x_k, x_{k+1}]$ is the $k$-th quantization interval.

In (3-20), the first and the last terms represent the *overload noise*; the middle term is the *granularity noise*. As we discussed before, the contribution of the overload noise can be neglected for a Gaussian signal if the quantizer has a loading factor $L$=4. Therefore, the first and last terms in (3-19) can be discarded in our example, such that the middle term in (3-20) gives the quantization noise.

In order to compare the SNR of the floating-point quantization ($SNR_{FP-ADC}$) and that of the uniform quantizer ($SNR_u$), the same normal distribution signal observing the four-sigma loading rule ($L$=4) is applied to both the FP-ADC and the uniform A/D converter.

In the uniform quantization, $\forall x \in \{[X_k, X_{k+1}) | X_{k+1} - X_k| = \Delta\}$, $p(x) \cong p(X_k)$.

If both the uniform and FP-ADC quantizers are assumed to have the same input range $V_{FS} = V_{REF^+} - V_{REF^-}$, the step size of the FP-ADC $\Delta_e$ is smaller than that of uniform A/D converter $\Delta$. The FP-ADC quantization step is a function of the exponent and it varies for different measurement domains ($D_e$) that are determined by the exponent's values:

$$\Delta_e = \frac{\Delta}{2^{\max E - e}} \text{ on } \{D_e, e \in [0, 2^E - 1] \cap \mathrm{N}\}$$

For any smaller quantization interval of the FP-ADC, it is assumed that $p(x) \cong p(x_k),\ \forall x \in \left\{ [x_k, x_{k+1}) \middle| x_{k+1} - x_k = \Delta_e = \frac{\Delta}{2^{\max E - e}} \right\}$. Then (3-20) can be expressed as:

$$\sigma_e^2 = \sum_{e=0}^{2^E-1} \left\{ \sum_{k=0}^{2^m-1} \left[ \int_{x_k}^{x_{k+1}} (x_k - x)^2 \cdot p(x) \cdot dx \right] \right\} \cong \sum_{e=0}^{2^E-1} \left\{ \sum_{k=0}^{2^m-1} \left[ \frac{1}{3} (x - x_k)^3 \cdot p(x) \Big|_{x_k}^{x_{k+1}} \right] \right\} \cong$$

$$\sum_{e=0}^{2^E-1} \left\{ \sum_{k=0}^{2^m-1} \left[ \frac{1}{3} \Delta_e{}^3 \cdot p(x_{k+1}) \right] \right\} \cong \frac{1}{3} \sum_{e=0}^{2^E-1} \left\{ \sum_{k=0}^{2^m-1} \left[ \Delta_e{}^3 \cdot p(x_k) \right] \right\} \cong \frac{\Delta^2}{3} \left\{ \frac{1}{4 \cdot 2^{2^{E-1}}} \sum_{e=0}^{2^E-1} \left[ 2^{2e} \cdot \int_{\tau_e} p(x) \cdot dx \right] \right\}$$

$$= \frac{\Delta^2}{12} \cdot G(x) \qquad (3\text{-}21)$$

Where $\left\{ |x_{k+1} - x_k| = \Delta_e = \frac{\Delta}{2^{\max E - e}} \right\}$, $x_k = y_k \cdot \frac{2^e}{2^{2^{E-1}}}$,

For a normally distributed signal, the objective function $G(x)$ with mean 0, represents the ratio of the SNR of the floating-point quantizer over that of the uniform one.

$$G(x) = \frac{SNR_{FP-ADCt}}{SNR_u}$$

Provided both uniform and FP-ADC quantizers have the same resolutions $m = b$, it can be shown that $G(x) > 1$, i.e., $SNR_{FP-ADCt} > SNR_u$. For a uniform quantizer with a resolution of $b = 4$ and the FP-ADC quantizer with $E = 2, m = 4$, $G(x)$ is almost 5 when quantizing normally distributed signals with mean 0 and variance 1. The relationship of $SNR_{FP-ADC}$ and $SNR_u$ is shown in Figure 3.5.

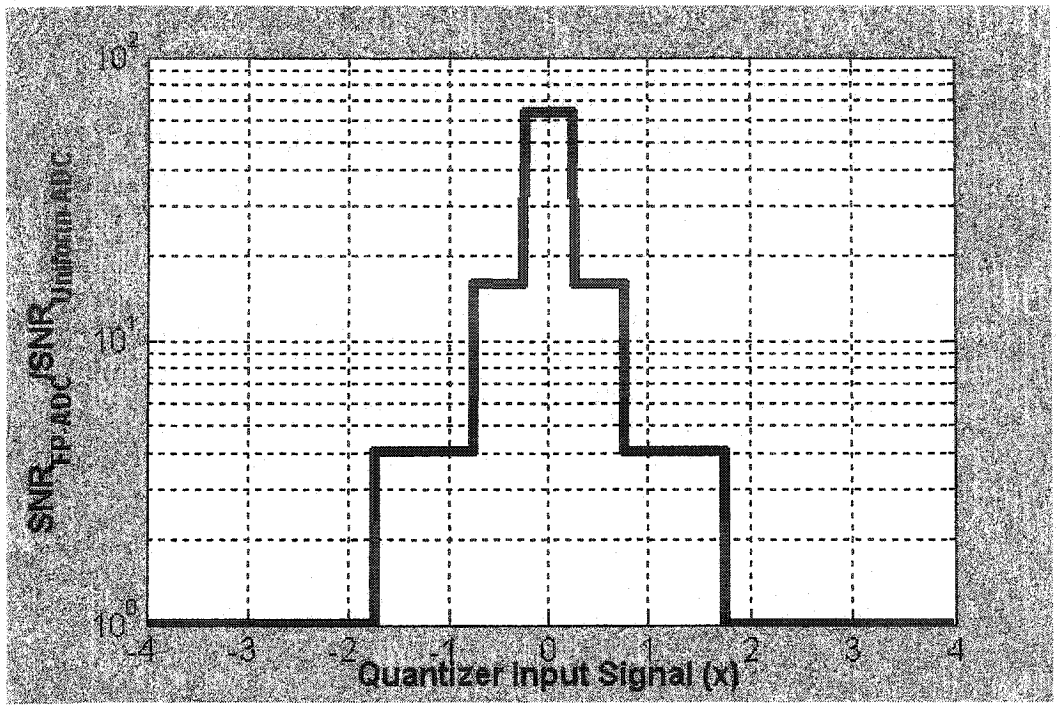


Figure 3.5 $SNR_{FP-ADC}$ VS $SNR_u$

It is easily seen that the smaller the signal is, the higher value $G(x)$ presents.

### 3.2.4 Conversion time

The conversion time, also called latency, of an A/D converter is the interval from the moment the analog input is sampled and until the associated digital output code is valid on the output.

The conversion time of the fixed-point converter $\tau_u$ is the sum of the propagation delays and setting times of the components in the analog path.

The acquisition time of a classic sequential floating-point A/D converter is twice $\tau_u$ because two uniform quantizers are used in the serial architecture of the floating-point A/D converter.

$$\tau_{FP-ADC} = 2 \cdot \tau_u \tag{3-22}$$

The average conversion time of the parallel architecture of the floating-point A/D converter is a function of the statistics of the acquired signal. If two consecutive samples of the quantizer input signal fall in the same measurement domain, the correct result is obtained in one conversion cycle $\tau_u$; if they fall in different measurement domains, a second conversion cycle with the appropriate gain is required and the conversion time doubles. It results that the average conversion time increases with the probability $P$ that two consecutive samples would fall in different measurement domains:

$$\tau_{(FP-ADC)_P} = \tau_u (1 + P) \tag{3-23}$$

## 3.3 Summary

This chapter introduced the new parallel architecture of the floating-point analog-to-digital converter. The characteristics and specifications of the floating-point A/D converter are described. And the simulation of the parallel architecture floating-point A/D converter is presented in statistic view.

# Chapter 4

# System Description

## 4.1 System Architecture

The whole system consists of two 10-bit Analog Devices Inc. AD9203 A/D converters, which implement the A/D-E and the A/D-M converters that get the exponent and the mantissa respectively; a Programmable Gain Amplifier (PGA) PGA203KP, which is a digitally controlled gain amplifier with binary model; an Altera APEX EP20K200EBC652-1X FPGA, which implements the control path of the parallel architecture of the floating-point analog-to-digital conversion FPADC. The main architecture of the system is shown in Figure 4.1.

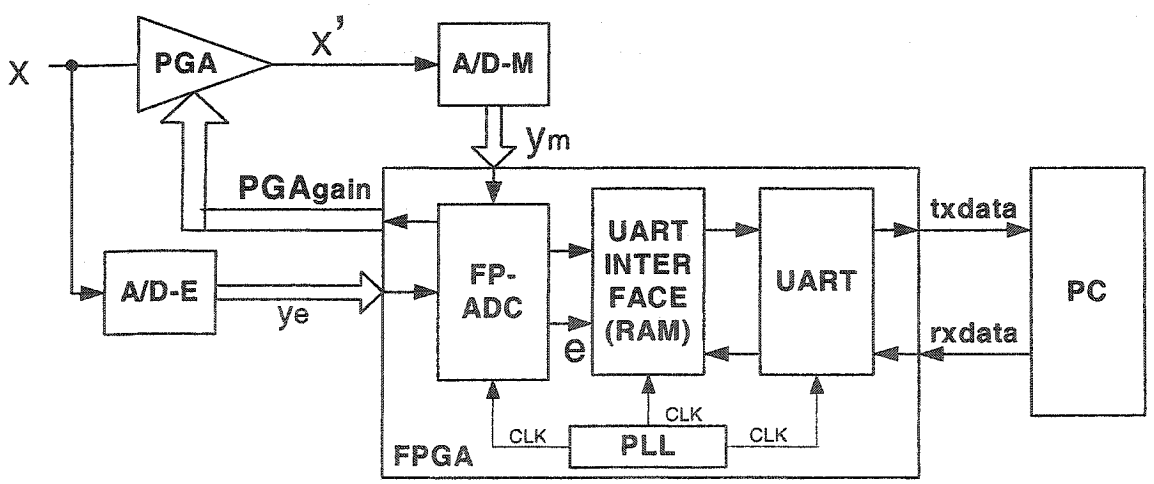


Figure 4.1 Block diagram of the whole system architecture

A/D-E and A/D-M converters are two AD9203 flash A/D converters. Input voltage is limited to 1V peak-to-peak to keep it within the linear portion of A/D converter's characteristics. The A/D subsystem has the following features:

The A/D converters produce 10-bit samples at a sample rate of 10 MSPS. The analog input to each A/D converter is single–ended. The output format of each A/D converter is two 's complement. The input to each A/D converter is used as AC coupled.

The EP20K200EBC652-1X FPGA device features 211,000 gates in a 652-pin Fineline BGA package. The device has 8320 logic cells and 106,496 RAM bits.

PGA203KP is selected as the PGA. It is a digitally controlled gain amplifier with binary model. The gains are 1, 2, 4 and 8. Gain selection is accomplished by the application of a 2-bit digital word to the gain select inputs. Table 4.1 shows the gains for the different possible values of the digital input word. A0 and A1 are two gain selection inputs.

Table 4.1 Software Gain Selections

| A1 | A0 | GAIN |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 2 |
| 1 | 0 | 4 |
| 1 | 1 | 8 |

The system works as follows:

When the FP-ADC is ready to begin a floating-point A/D conversion, the "start_adc" command is sent from PC, as an encoded message - "rxdata," via the RS232 port to the FPGA UART. The FP-ADC control unit in FPGA decodes this message and starts the floating-point A/D acquisition. Every floating-point conversion result is sent in a DMA (Direct Memory Access) manner to a buffer that is implemented as a dual-port memory block. When the number of the sample data reaches a certain number, such as 128, the floating-point A/D conversions stop and the FPGA sends the acquired data (the mantissa data and the computed exponents), serially, as "txdata," to the PC through the UART for further processing. The floating–point A/D conversion FPGA implementation part is discussed in the following.

## 4.2 FPGA Subsystem Description

From Figure 4.1, it is easily seen that the FPGA subsystem FPADC consists of four main modules: FP-ADC (floating-point analog-to-digital conversion) module, PLL (Phase Locked Loop) module, UART (universal asynchronous receiver and transmitter) module, and UART INTERFACE module. VHDL hardware programming language is used in this design. The main architecture of FPGA subsystem is illustrated in Figure 4.2. The detailed information about FPADC is discussed in section 4.3.

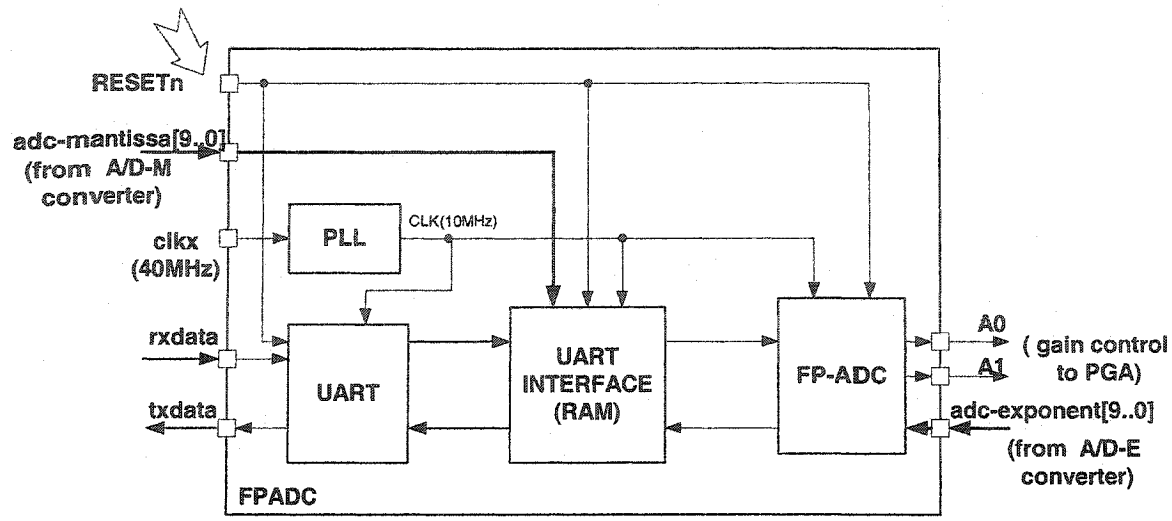


Figure 4.2 The main architecture of the FPGA subsystem FPADC

In VHDL design, FPADC.vhd is a top-level hierarchy module of the FPGA subsystem FPADC. All sub-modules are instantiated here. No logic is present in this module. It consists of 4 submodules: FP-ADC.vhd, CLKPLL.vhd, UART.vhd, UART_INTERFACE_ADC.vhd.

The FP-ADC is the entity of the floating-point A/D conversion. It provides the four logic functions: to implement the gain decoder, to get the current gain from the A/D-E converter, to generate the exponent from the gain value stored in the *PGAgain* register, to implement the control unit to correctly operate the A/D-E and the A/D-M. In VHDL, it corresponds to FP-ADC.vhd. It has three submodules: UART_DECODER.vhd, GAIN_DECODER.vhd and DUALCLOCL.vhd. Section 4.3 presents the information in details.

The PLL is the entity of the clock phase locked loop, which can provide the system clock signals to FP-ADC, UART, and UART INTERFACE. The main system clock has a frequency of 10 MHz and it is controlled by the PLL Unit fed by a 40MHz onboard

oscillator. In VHDL, CLKPLL.vhd is the corresponding core that has ClockLock and ClockBoost features. Section 4.4 gives the detailed information.

The UART is the entity of UART implemented in FPGA. The input data can be sent from a PC's serial port to the UART in the APEX DSP board via a RS232 cable. PC can receive the output data from the UART INTERFACE module and send the control command to FPGA through UART, as well. UART.vhd is a universal asynchronous receiver and transmitter core in VHDL. The detailed information can be referred in section 4.5.

The UART INTERFACE is the entity that implements the interface between the UART and the FP-ADC to realize the serial communication control and access to the on-chip memory RAM. This memory buffer is used to store temporarily the acquired data (mantissa part and the computed exponent part of the floating-point A/D conversions), and then to output these data to the UART. Uart_interface_adc.vhd is the corresponding module in VHDL. Section 4.6 should be referred to.

The system can be reset at any time by asserting the signal "*RESETn*" by pressing the master reset switch SW0. This pin is used as a standard I/O pin to implement a reset in the design.

The data input and output I/0 ports are separated. The mantissa part is read from the A/D-M converter. The exponent part is read from the A/D-E converter. They are all 10-bit sample data at 10MHz frequency. A0 and A1 are output ports that control the PGA gain. When the current gain coincides with the one stored in the *PGAgain* register, the

conversion is correct, and the mantissa and the exponent can be stored in the RAM. When the number of the sample data equals the limitation, for instance, 128, the floating-point A/D conversion stops and the data in RAM can be sent to through the UART to the PC host computer. Then the data can be displayed on PC. Pins "rxdata" and "txdata" are input and output ports of UART.

The following table (Table 4.2) describes the default port list of the FPADC.vhd. In this table, "I" means input and "O" means output.

| Signal | Width | Type | Description |
|---|---|---|---|
| CLKX | 1 | I | Main system clock |
| RESETn | 1 | I | Main system reset (logic "0") |
| rxdata | 1 | I | Received serial data |
| txdata | 1 | O | Transmitted serial data |
| adc_mantissa | 10 | I | A/D converter for mantissa |
| adc_exponent | 10 | I | A/D converter for exponent |
| A0 | 1 | O | To control PGA, gain |
| A1 | 1 | O | To control PGA, gain |

Table 4.2 The default ports list of the FPADC

## 4.3 FP-ADC CONTROL UNIT

FP-ADC is a module of the floating-point analog-to-digital conversion. It has three submodules: UART_DECODER.vhd, GAIN_DECODER.vhd, DUALCLOCK.vhd. Each module is discussed in the following.

### 4.3.1 UART_DECODER UNIT

UART_DECODER.vhd is a module that decodes the FP-ADC control signals (e.g., "start_adc").

When the start_adc command, encoded as i_word_in = "01000001", is received by the UART_DECODER, "start_adc" is set to high and the floating-point A/D conversions begin. The sample data from the A/D-E converter are read to GAIN_DECODER module to be used for the gain calculation. Otherwise, no conversion begins when the signal " start_adc" is "0". The clock frequency is the same as that of the UART clock.

### 4.3.2 DUAL CLOCK UNIT

DUALCLOCK.vhd is a module that transforms the handshake signal "end_adc_f" from the fast clock domain 10MHz to the UART clock domain as the signal "end_adc".

The main technique of transferring signals between two clock domains is shown in Figure 4.3. The waveform is shown in Figure 4.4.

There are two clocks: one is CLK 10 MHz; the other is UART_CLK, which is much slower than CLK. The "end_adc_f" signal is an output from

UART_INTERFACE_ADC. The A/D-E conversion stops when it is high. Because the working clock frequency in GAIN_DECODER is 10 MHz and UART working clock is UART_CLK, the signal "end_adc_f" in the fast clock domain is needed to be transferred to the signal "end_adc" in the low frequency clock domain. When " end_adc" is high, the data stored in RAM can be sent to UART

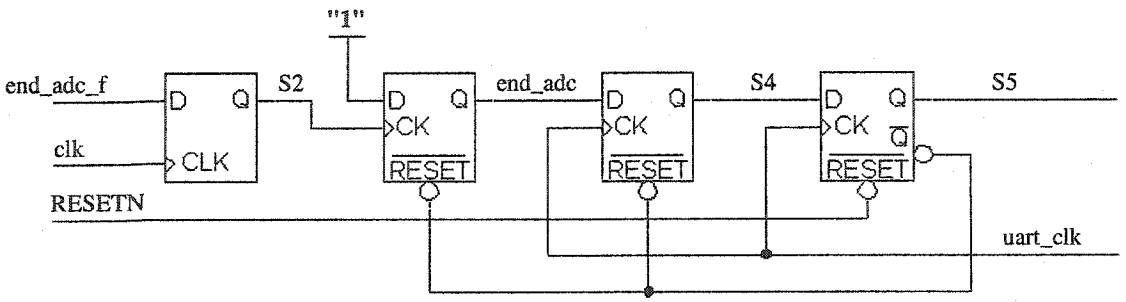


Figure 4.3 Transferring high-speed clock signals to low frequency clock domains- Digital logic circuit

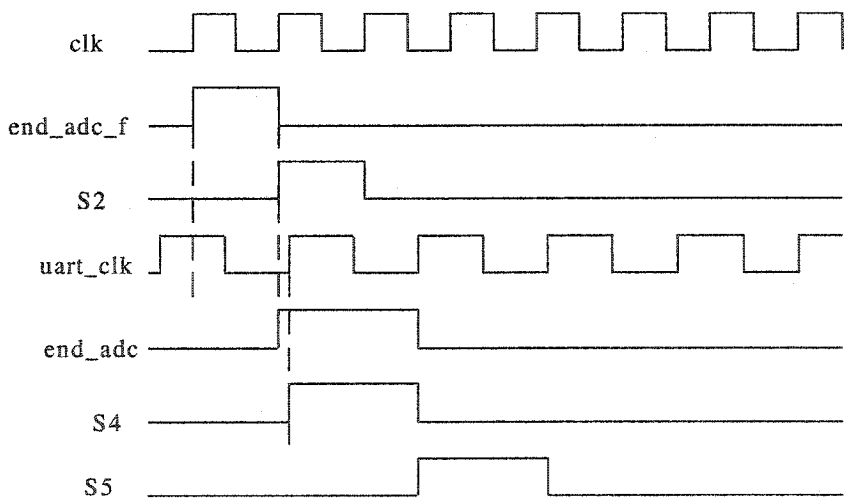


Figure 4.4 Transferring high-speed clock signals to low frequency clock domains - Timing diagram

### 4.3.3 GAIN_DECODER UNIT

GAIN_DECODER unit is a module that implements the logic functions to get the current gain from the A/D-E converter and outputs it to PGA, and generates the exponent from the gain value stored in *PG gain* register.

The architecture of GAIN_DECODER is shown in Figure 4.5.

The state machine is a simple 3-states Mealy type. The state flow chart is shown in Figure 4.6.

Table 4.3 shows the default ports list of the GAIN_DECODER.vhd. "I" means the input and "O" means the output.

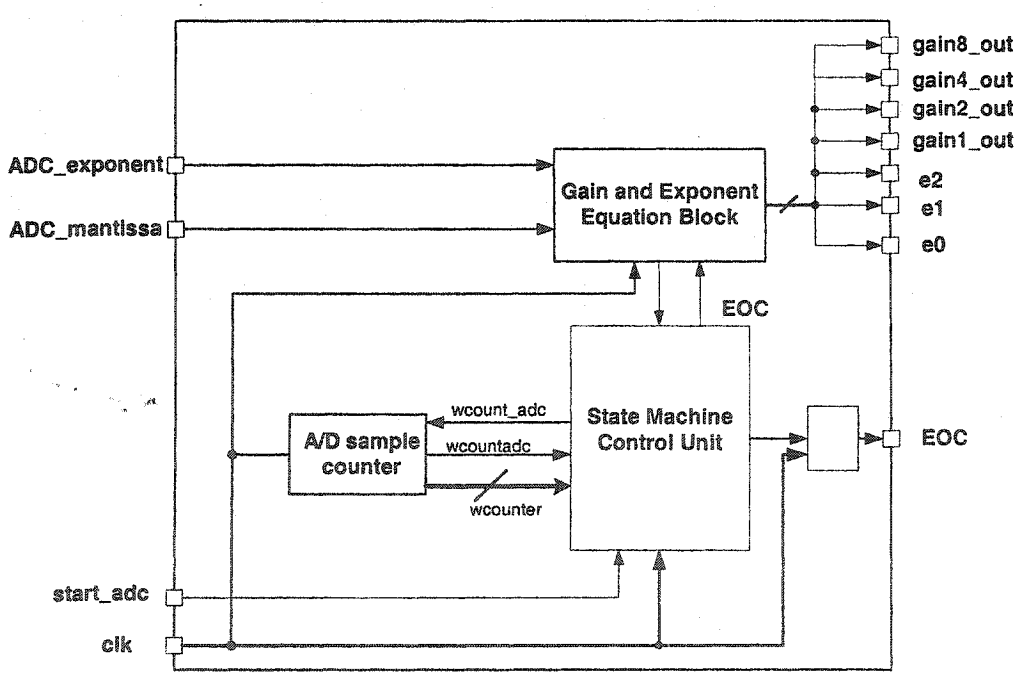


Figure 4.5 Block diagram of GAIN_DECODER unit

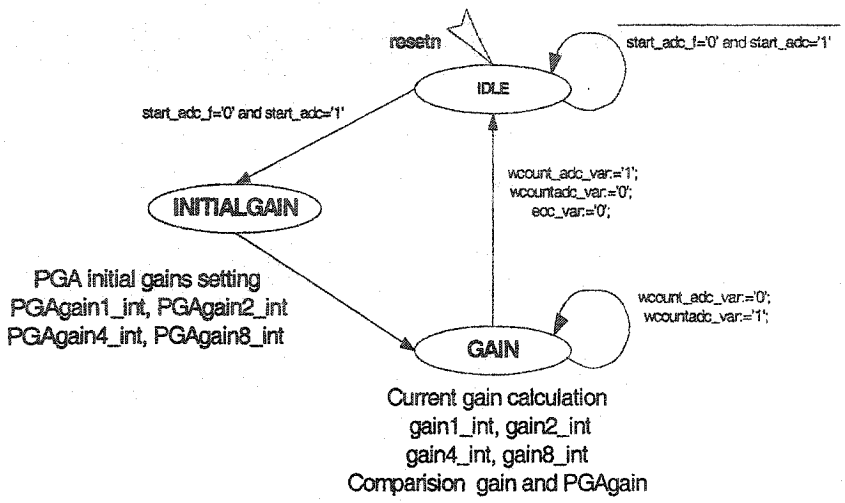


Figure 4.6 The state diagram of GAIN_DECODER control unit

Table 4.3 The ports list of GAIN_DECODER unit.

| Signal | Width | Type | Description |
|---|---|---|---|
| CLK | 1 | I | System clock 10MHZ |
| RESETn | 1 | I | Main system reset (logic "0") |
| ser_data_RX | 1 | I | The serial data input to Receiver |
| start_adc | 1 | I | start_adc command from UART_DECODER |
| adc_exponent | 10 | I | A/D-E converter for exponent |
| Gain1_out | 1 | O | To control PGA, gain1 |
| Gain2_out | 1 | O | To control PGA, gain2 |
| Gain4_out | 1 | O | To control PGA, gain4 |
| Gain8_out | 1 | O | To control PGA, gain8 |

| | | | |
|---|---|---|---|
| E0 | 1 | O | Exponent output |
| E1 | 1 | O | Exponent output |
| E2 | 1 | O | Exponent output |
| EOC | 1 | O | Handshake signal, "1" means conversion successful |

In our design, the exponent $e$ is 3 bits. Therefore, a unique gain is from 1 to $2^7$. For a bipolar input signal $V_{in} \in [-1V,+1V]$ represented in 2's complement, "0" as the most significant bit (sign bit) means positive, while "1" means negative. The quantization step size is $\frac{1}{128} \approx 8mV$. When $|V_{in}| \in (0v,+8mB)$, the exponent is 0. It is represented with the floating-point form, where the mantissa part is in 2's complement.

$$1000,0000,\ldots000 < y_m < \mathbf{0111},1111,\ldots111 \quad * \ 2^0$$

In order to make this value in the upper half of the A/D-M conversion range, the PGA gain should be 128.

Similarly, when $|V_{in}| \in (8mV,16mV)$, the exponent is 1. In order to make this value in the upper half of the A/D-M conversion range, the PGA gain should be 64.

Therefore, we can get the Table 4.4. It shows the relationship among the input signal $V_{in}$, gain, and exponent $e$. The whole table can be referred to Appexdix1. From Table 4.4, it is easily seen how to implement the gain and the exponent.

Table 4.4 The relationships among the input signal $V_{in}$, gain, and exponent $e$

| ye7 | ye6 | ye5 | ye4 | ye3 | ye2 | ye1 | ye0 | mV | gain | e | exponent |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 128 | | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 8 | 64 | | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 16 | 32 | | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 24 | 32 | | 2 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 32 | 16 | | 3 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 40 | 16 | | 3 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 48 | 16 | | 3 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 56 | 16 | | 3 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 64 | 8 | 3 | 4 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 72 | 8 | 3 | 4 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 112 | 8 | 3 | 4 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 120 | 8 | 3 | 4 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 128 | 4 | 2 | 5 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 136 | 4 | 2 | 5 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 232 | 4 | 2 | 5 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 240 | 4 | 2 | 5 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 248 | 4 | 2 | 5 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 256 | 2 | 1 | 6 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 264 | 2 | 1 | 6 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 488 | 2 | 1 | 6 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 496 | 2 | 1 | 6 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 504 | 2 | 1 | 6 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 512 | 1 | 0 | 7 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 520 | 1 | 0 | 7 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 528 | 1 | 0 | 7 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1008 | 1 | 0 | 7 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1016 | 1 | 0 | 7 |

| ye7 | ye6 | ye5 | ye4 | ye3 | ye2 | ye1 | ye0 | mV |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | -8 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | -16 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | -24 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | -32 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | -40 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | -48 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | -56 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | -64 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | -72 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | -80 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | -120 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | -128 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | -136 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | -144 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | -240 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | -248 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | -256 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | -264 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | -272 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | -496 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | -504 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | -512 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | -520 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | -528 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | -536 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | -1016 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1024 |

Upon the system reset, the state machine defaults to IDLE state. In this state, the state machine idles as long as no "start_adc" command is given. When the start_adc command is detected on the rising edge, "start_adc_f" = '0' and "start_adc" = '1', the state machine is set to INITIALGAIN state.

In INITIALGAIN state, the sample data from the A/D-E converter are read and the PGA gains are stored in the register PGAgian1_int, PGAgain2_int, PGAgain4_int and PGAgain8_int. PGA gains are set according to the following logic equations.

```
PGAgain1_int<=((not ad_e_int(9))and ad_e_int(8))or ( ad_e_int(9) and (not ad_e_int(8)));

PGAgain2_int<=((not ad_e_int(9))and (not ad_e_int(8))and ad_e_int(7)) or
                ( ad_e_int(9) and ad_e_int(8)and ( not ad_e_int(7)));

PGAgain4_int<=((not ad_e_int(9))and (not ad_e_int(8))and (not ad_e_int(7))and
                ad_e_int(6)) or ( ad_e_int(9) and ad_e_int(8)and ad_e_int(7)and (not
                ad_e_int(6)));

PGAgain8_int<=((not ad_e_int(9))and (not ad_e_int(8))and (not ad_e_int(7))and (not
                ad_e_int(6))) or ( ad_e_int(9) and ad_e_int(8)and ad_e_int(7)and
                ad_e_int(6));
```

where {ad_e_int($k$+2) = $y_e(k)$, $k$ = 0,1,...,7} are the outputs of A/D-E (Fig. 4.1)

After INITIALGAIN the state machine goes unconditionally to GAIN state.

In GAIN state, the new sample data from the A/D-E converter are read at the next rising clock edge. The current gains are set according to the following equations.

```
gain1_int<=((not ad_e_int(9))and ad_e_int(8))or ( ad_e_int(9) and (not ad_e_int(8)));

gain2_int<=((not ad_e_int(9))and (not ad_e_int(8))and ad_e_int(7)) or (ad_e_int(9) and
        ad_e_int(8)and ( not ad_e_int(7)));

gain4_int<=((not ad_e_int(9))and (not ad_e_int(8))and (not ad_e_int(7))and ad_e_int(6))
        or ( ad_e_int(9) and ad_e_int(8)and ad_e_int(7)and (not ad_e_int(6)));

gain8_int<=((not ad_e_int(9))and (not ad_e_int(8))and (not ad_e_int(7))and (not
        ad_e_int(6))) or ( ad_e_int(9) and ad_e_int(8)and ad_e_int(7)and ad_e_int(6));
```

Notice that the sample data are assigned the new values. If the current gains: gain1_int, gain2_int, gain4_int and gain8_int coincide with those stored in PGAgain registers, that is, gain1_int=PGAgain1_int and gain2_int=PGAgain2_int and gain8_int=PGAgain8_int and gain4_int=PGAgain4_int, then the handshake signal "EOC" is asserted high. It means the best relative precision is obtained and the conversion is successful. Gain1_out, gain2_out, gain4_out and gain8_out are output to PGA amplifier. And the exponents are calculated according to the these equations:

```
e0_int<=gain4_int or gain1_int;

e1_int<=gain1_int or gain2_int;

e2_int<='1';
```

There is a counter in GAIN_DECODER.vhd that is used to keep track of the number of successful conversions. When this count reaches the limit of the number of conversions, like 128, the state-machine stops the floating-point A/D conversion. This

counter has two control inputs: wcountadc and wcount_adc. When the former is active high, the counter advances by 1; when the latter is active high, the counter is cleared to 0.

When the number of successful conversions is less than 128, the two signals are asserted: wcount_adc_var:='0' and wcountadc_var:='1. It means that the counter content is incremented. Otherwise, wcount_adc_var:='1' and wcountadc_var:='1', the counter is reset.

When the conversion is not successful, "EOC' is "0". PGA gains are updated with the current gains and the state machine is set back to GAIN state to do the next conversion. When the number of sample data is up to the limitation, for instance, 128, the state machine goes to IDLE state; otherwise, it goes to GAIN state to continue the next conversion.

## 4.4 PLL

CLKPLL.vhd is a phase-locked loop (PLL) module that is instantiated by using the megafuntion *altclkclock* to increase performance.

APEX20K device has one PLL that features ClockLock and ClockBoost circuitry to increase performance and provide clock-frequency synthesis. The ClockLock feature minimizes clock delay and clock skew within the device, reducing clock-to-output and setup times while maintaining zero hold times. The ClockBoost feature allows designers to run the internal logic of the device at a faster or slower rate than the input clock frequency. The detailed information can be referred to [45]. The board has a 40-Mhz oscillator, which drives the APEX device CLK1P and CLK2P inputs. Figure 4.7 shows

the ClockLock and ClockBoost circuitry block diagrams within the altclkclock megafunction and its ports.

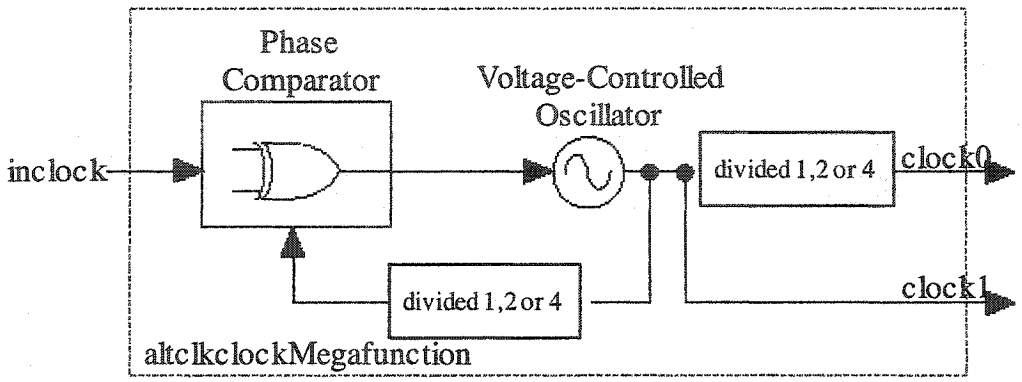


Figure 4.7 ClockLock & ClockBoost Circuitry in APEX20K Devices

The FPGA's dedicated clock pin CLK2 supplies the clock to the PLL and the megafunction. The inclock port must be fed directly by this pin without inversion. Figure 4.8 illustrates the valid clock connections for the PLL and global clock lines in the design. The working clock of the system is 10 MHz.

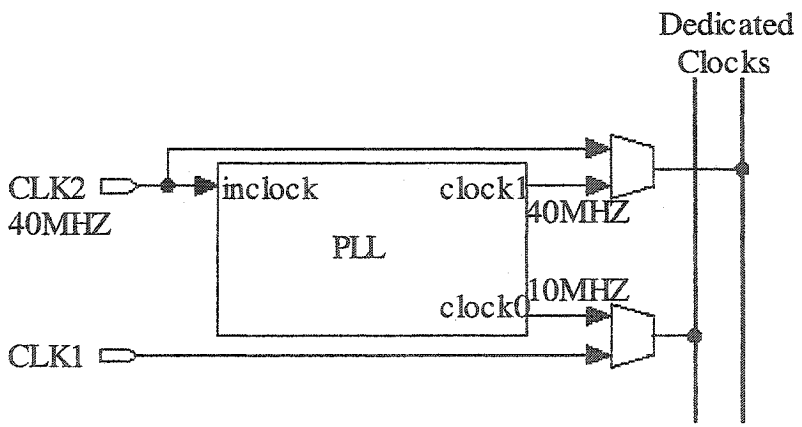

Figure 4.8 APEX 20K Dedicated Global Clock Pin Connections to PLL & Dedicated Clock Lines

## 4.5 UART

The UART is a universal asynchronous receiver and transmitter core that can work fully functionally and synthetically. The receiver and transmitter operate independently.

The UART module comprises of four submodules: UART.vhd, TRANSMITTER.vhd, RECEIVER.vhd and DIV_CLK.vhd. The information about each submodule is elaborated later.

UART.vhd is a top-level hierarchy module. All sub-modules are instantiated here. No logic is present in this module.

TRANSMITTER.vhd is an asynchronous transmitter. A state-machine, serializer and support logic comprises the main bulk of the logic.

RECEIVER.vhd is an asynchronous receiver. A dual-rank synchronizer, state-machine, deserializer and support logic comprises the bulk of the logic.

DIV_CLK.vhd is a baud-rate generator. An internal" baud-clock" which is 16 times the desired baud-rate is generated from this entity.

The UART architecture is shown in Figure 4.9.

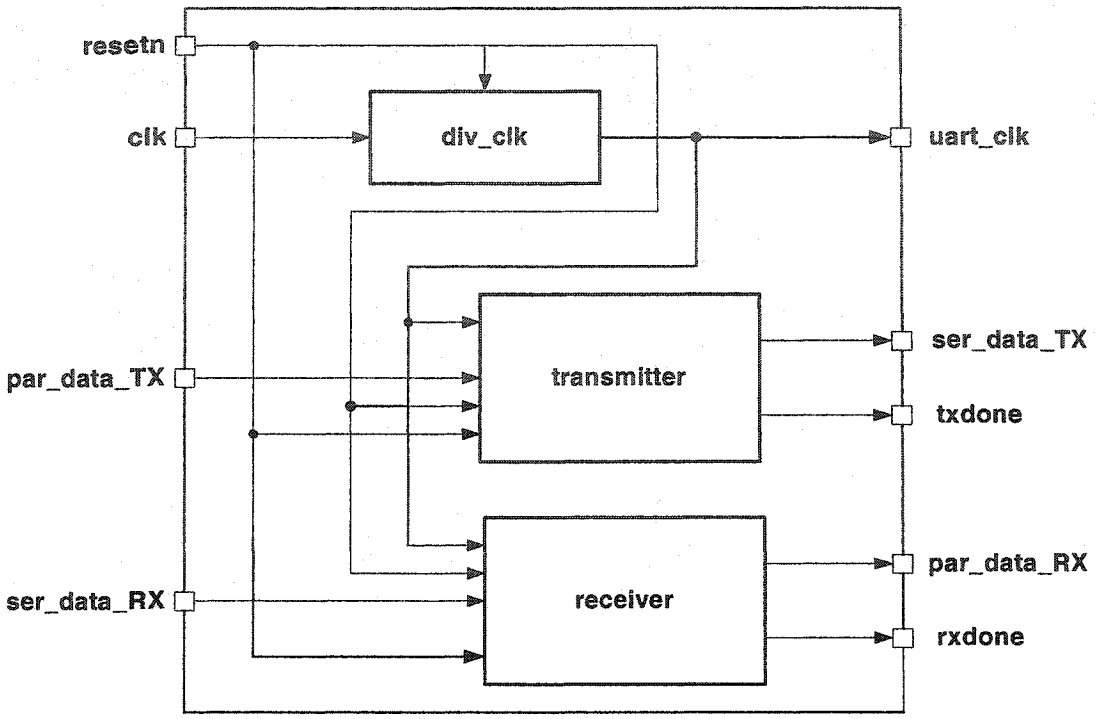


Figure 4.9 Block diagram of the UART architecture

Table 4.5 shows the default ports list of the UART.vhd. "I" means the input and "O" means the output.

| Signal | Width | Type | Description |
|---|---|---|---|
| CLK | 1 | I | system clock 10MHZ |
| UART_CLK | 1 | O | UART clock |
| RESETn | 1 | I | Main system reset (logic "0") |

| | | | |
|---|---|---|---|
| ser_data_RX | 1 | I | The serial data input to Receiver |
| par_data_RX | 8 | O | The parallel data output from Receiver |
| rxdone | 1 | O | Handshake signal: (logic 1), received successfully |
| par_data_TX | 8 | I | The Parallel data input to transmitter |
| Start_TX | 1 | I | Control signal (logic 1), start transmission |
| ser_data_TX | 1 | O | The serial data output from transmitter |
| txdone | 1 | O | Handshake signal: (logic 1), transmitted successfully |

Table 4.5 Port list of the UART

### 4.5.1 UART Operation Theory

Figure 4.10 illustrates the basics of a UART data package. When no data is being transmitted, the line par_data_TX must be logic "1". A data package consists of a start bit, which is always a logic "0", followed by a number of data bits (typically are 8 bits), an optional parity bit, and a number of a stop bits (typically 1). The stop bit must always be logic "1".

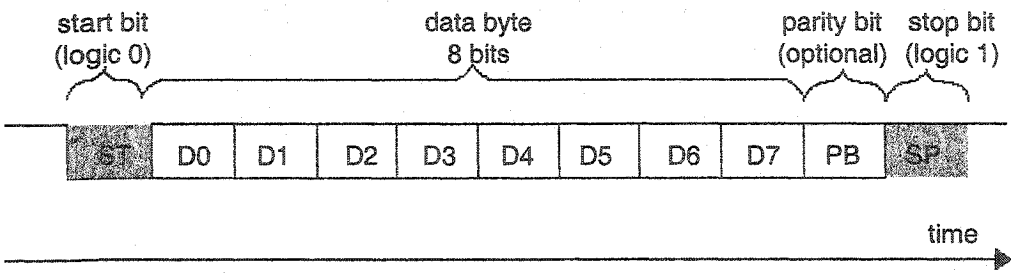


Figure 4.10 Basic UART packet format

In UART, it uses 8 bits for data, no parity bit and 1 stop bit. Thus, it takes 10 bits to transmit a byte of a data.

In the UART protocol, the transmitter and receiver do not share a clock signal. Thus, a clock signal does not emanate from one UART transmitter to the other UART receiver and a baud rate must be set prior to data transmission. That is, the receiving UART needs to know the transmitting UART's baud rate (and verse vice). In almost all cases, the receiving and transmitting baud rates are the same. The transmitter shifts out the data starting with LSB first.

A key concept in UART design is that UART's internal clock runs at much faster rate than the baud rate.

The receiver detects the start bit by detecting the transition from logic "1" to logic "0" (note that while the data line is idle, the logic level is high). Once the start-bit is detected, the next data bit's " center" can be assured to be 24 ticks minus 2. From then on, every next data bit center is 16 clock ticks later. Figure 4.11 illustrates this point.

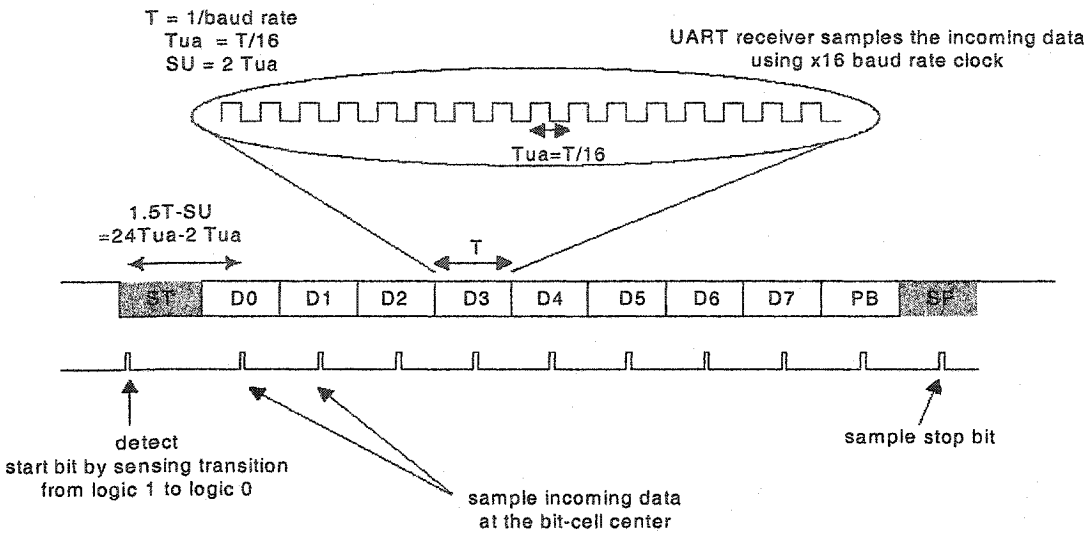


Figure 4.11 Data sampling points by the UART receiver

Once the start-bit is detected, the subsequent data bits assembled in a de-serializer.

### 4.5.2 Functional Description of Receiver

The receiver of UART is composed of a control state-machine, de-serializer, and support logic. The main goal of the receiver is to detect the start-bit, then de-serialize the following bit-stream, detect the stop-bit, and make the data available to the host.

Figure 4.12 illustrates the functional block diagram of the receiver.

The signal UART_CLK is 16 times Baud-Rate generated by the baud rate generator DIV_CLK.vhd. This clock is used to drive the entire clock within the receiver module.

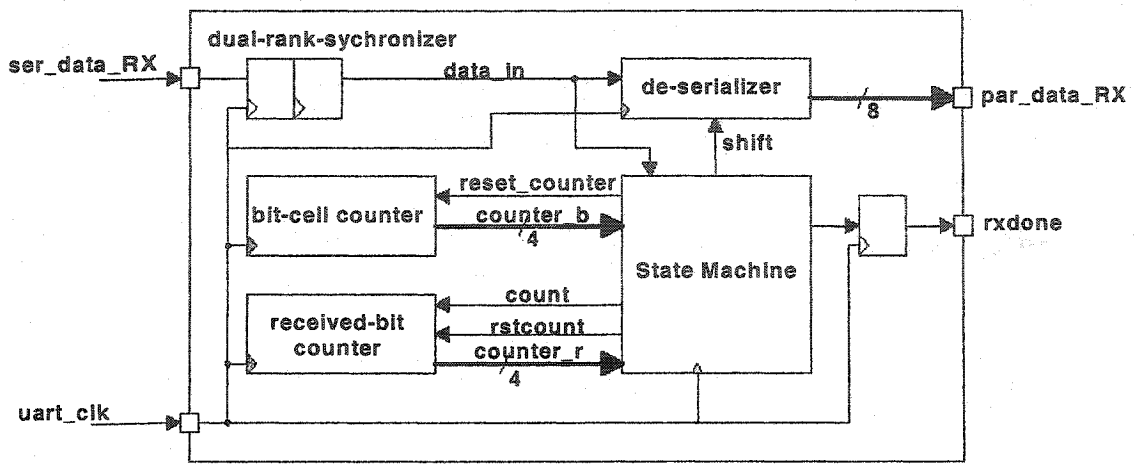


Figure 4.12 Functional block diagram of the receiver

The incoming data ser_data_rx is fed to the dual-rank sychronizer before feeding to the de-serializer. Note that this synchronizer is absolutely essential since the data present on ser_data_rx is synchronous to the transmitter's clock, and not on the receiver's clock.

The de-serializer is a simple serial-to-parallel shift-register. It has a control input "shift" from the state-machine. When this signal is active high, the de-serializer shifts that data by 1 bit. The default shift register width is 8 bits.

The received bit counter is used to keep track of the number of data bits cumulated so far. When this count becomes the limit of the word-length 8, the state-machine stops accepting more data bits. This counter has two control inputs: count and rstcount. When the former is active high, 1 advances the counter; when the latter is active high, the counter is cleared to 0. This counter is 4-bits wide by default.

The state-machine is a simple Mealy type with 5states. Figure 4.13 illustrates the state flow.

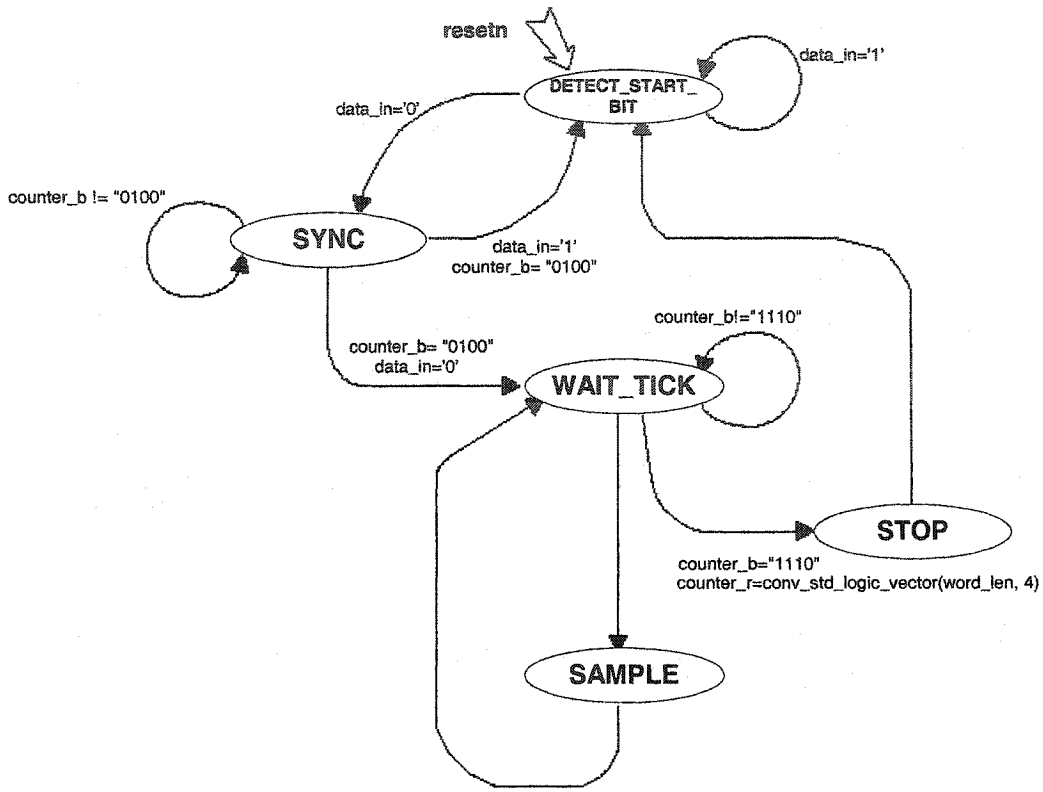


Figure 4.13 UART Receiver State Flow

The state-machine has all of the functional units previously described. Upon the system reset, the state machine defaults to DETECT_START_BIT state. In this case, the state-machine looks for the start-bit. This condition is detected by the transition of the incoming data (which at idle is logic "1") to logic "0". Once the start-bit is detected, the state-machine is transferred to SYNC state.

In SYNC state, the state-machine waits for ½ bit cell in order to find the bit-cell center. Once the bit-cell center is found, if the state of the data_in (synchronized incoming data) is still low, the state machine transits to WAIT_TICK state. If data_in is high, this is not a valid start bit, so the state-machine transits back to SYNC state.

The WAIT_TICK state simply waits for 1-baud tick (16 uart_ticks). Once 1-baud tick is waited, the incoming data can be sampled into the de-serializer. If all 8 bits have been sampled, the state machine is transferred to STOP state; otherwise, it is transferred to SAMPLE state.

In SAMPLE state, the state of data_in is sampled into the de-serializer.

In STOP state, the state of data_in is checked for logic "1". The bit is not sampled into the deserializer. Before transferring to DETECT_START_BIT, a status signal is generated "rxdone_int", to indicate that a valid data is flopped before being made available. This practice reduces critical path timing constraints.

### 4.5.3 Functional Description of Transmitter

The transmitter of UART is composed of a bit cell counter, a transmitted bit counter, a serializer and a sate machine. Like the receiver counter part, the design is minimized and contains no error detecting logic.

Figure 4.14 illustrates the functional block diagram. Notice the similarities with the UART receiver. The bit-cell counter and the transmitted bit counter have the same

function and implementation as those of the receiver; only the signal names have been changed slightly.

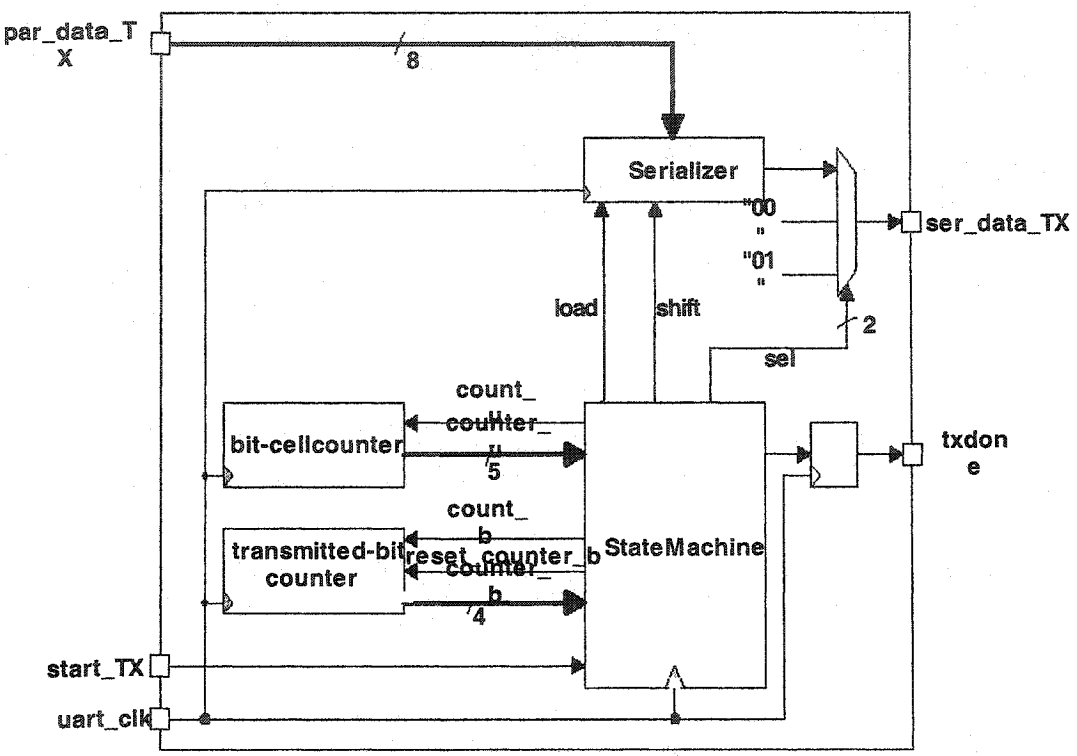


Figure 4.14 Functional block diagram of UART transmitter

The serializer is an 8-bit parallel-in-out shift register. It has two control inputs: "load" and "shift". An active high on the first signal loads the parallel data into the shift register. An active high on the latter signal shifts the loaded data out by 1 bit.

A three-input multiplexer is present on the ser_data_TX signal. This MUX is used to select from the start-bit (logic "0"), user data (from the shift register) and the stop-bit (logic "1").

The status signal txdone_int is registered before being made available. Again, this practice eliminates critical path timing issues.

The state machine is also a simple 5 state mealy type and is shown in Figure 4.15. With the same as the receiver, 1-baud tick is equal to 16 uart_clk ticks

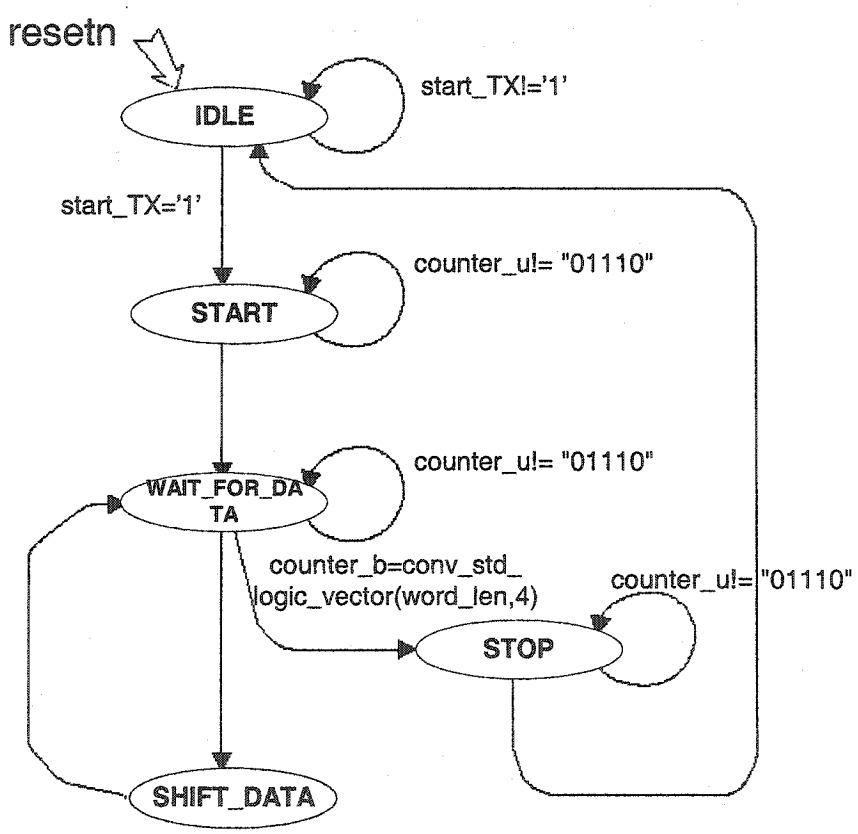


Figure 4.15 UART transmitter State Flow

Upon the system reset, the state machine defaults to IDLE state. In this state, the state machine idles for as long as no transmit command is given. But when the transmit

command "start_TX" becomes active high (for 1 uart_tick), the serializer is loaded and the state machine is set to START state.

In START state, the MUX "sel" is set to "00", ser_data_TX_var := '0'. It means the transmitted start bit is "0". 1-baud tick is waited (16 uart_ticks) before transition to WAIT_FOR_DATA state.

In WAIT_FOR_DATA state, the MUX "sel" is set to "10", pointing to the shift register: ser_data_TX_var := txregister(0). And 1-baud tick is waited. After the wait is complete, if all bits (word-length=8) have been transmitted, the state machine transitions to STOP state, otherwise it goes to SHIFT_DATA state.

In the SHIFT_DATA state, the shift-register is shifted by 1 bit, and transferred to WAIT_FOR_DATA state.

In STOP state, the MUX "sel" is set to "01", ser_data_TX_var := '1'. It means the transmitted stop bit is "1". 1-baud tick is waited and then transitions to IDLE state.

### 4.5.4 Functional Description of Baud Rate Generator

The baud rate generator has a simple function. It takes the externally fed back clock (clk), and generates baud_clk. As previously stated, in order to minimize the gate count, the baud rate and the system clock rate is specified according to this formula.

DIVIDED=integer [CLKX/(BAUD*2*16)].

Constant CPR is the width of the internal counters used to generate the appropriate baud delay, which should be greater than log2 (DIVIDED).

When BAUD is 9600bps, CLKX system clock is 10Mhz, DIVIDED is integer 6.

## 4.6 UART_INTERFACE_ADC Unit

UART_INTERFACE_ADC.vhd is a module to control the data between FP-ADC and UART.

There is a RAM in this module: UART_REC_BUF.vhd. It is instantiated by the parameterized dual-port RAM megafunction: lpm_ram_dp. The data input bus to the memory is 12 bits wide. The address input bus is 8 bits wide. The total memory is 12 bits x 128 words. Two separated read and write clocks are used: write clcok frequency is clk 10MHZ: wrclock=> clk; read clock frequency is uart clock: rdclock=> clk_u. Positive-edge-triggered clock for write and read operation. Write input ports: "data", "waddress", "wren" are registered. Read input ports: "raddress", "rden" are registered. "Wren" and "rden" are writing and reading enable inputs. Disables writing and reading when low (0). The default is 1.

The architecture of UART_INTERFACE_ADC is shown in Figure 4.16.

The output data exponent from GAIN_DECODER unit and the sample data from mantissa A/D conversion written to RAM are controlled by a simple 2-states Mealy type state machine. The state flow chart is shown in Figure 4.17.

Upon the system reset, the state machine defaults to IDLE state. In this state, the state machine idles for as long as no "start_adc" command is given. When the start_adc command " start_adc_f = '0' and start_adc = '1' ", the state machine is set to WRITE state.

In WRITE state, if the handshake signal "EOC" is high, the output data exponent from GAIN_DECODER unit, "EOC" handshake signal and the sample data from mantissa A/D conversion are written to RAM. As long as the number of the sample data is less than the limitation, for instance, 128, the state machine transitions to WRITE state repeating writing to RAM; otherwise it goes to IDLE state, the handshake signal "end_adc_f" is asserted high. This signal goes to DUALCLOCL unit to get the signal "end_adc" in the uart clock domain. If the handshake signal "EOC" is low, "wren" is disabling and data can't be written to RAM.

When the signal "end_adc=1", RAM is ready to be read. The transmitted command start_TX<='1', meaning the transmitter in UART can work. When the signal "txdone =1", RAM output data can be read, "start_tx" and "rden" are assigned to high. When the number of the transmitted data is equal to the limitation, for instance, 128, the transmission stops, "start_tx" and "rden" are assigned to low.

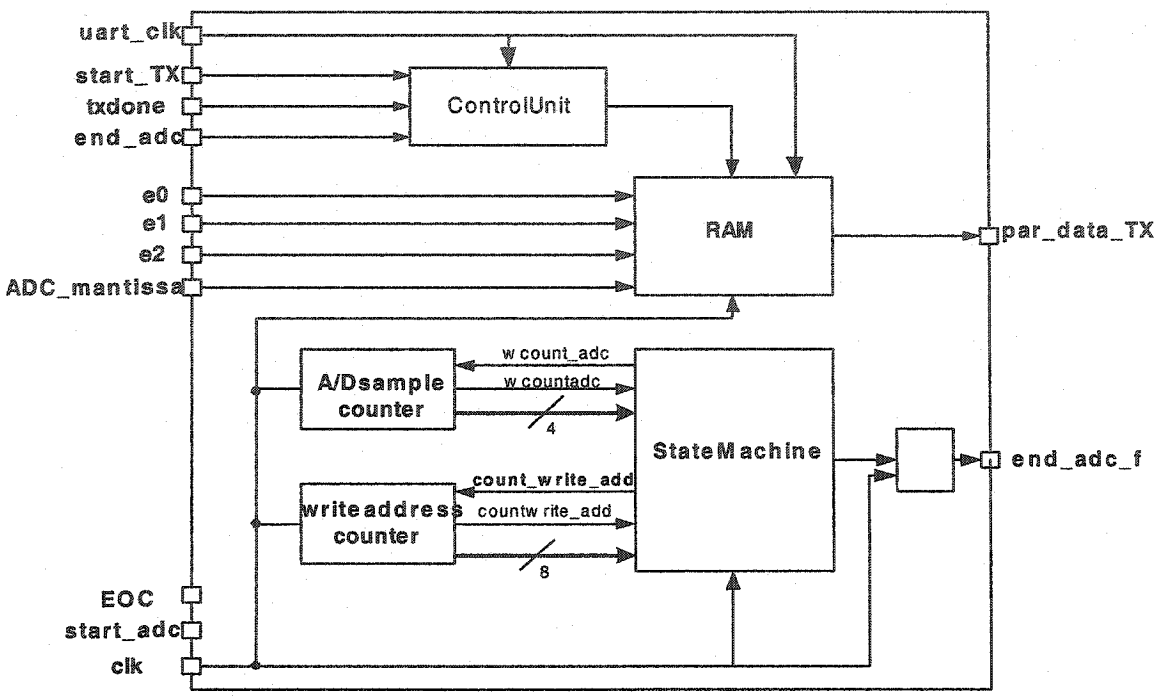


Figure 4.16 Block diagram of UART_INTERFACE_ADC unit

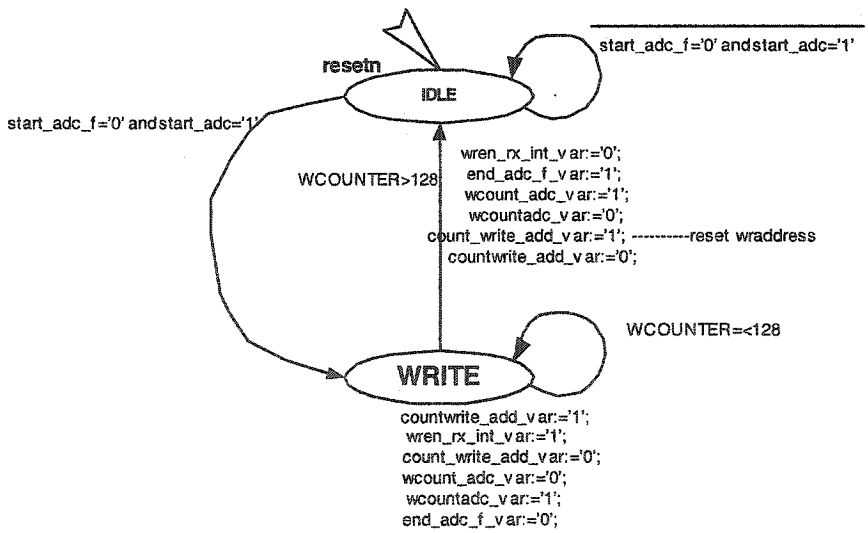


Figure 4.17 The state flow of UART_INTERFACE_ADC unit

## 4.7 Summary

This chapter mainly introduces the whole system of the parallel architecture of the floating-point A/D converter. ALTERA APEX EP20K200EBC652-1X FPGA implementation of the parallel architecture of the floating-point A/D converter is mainly discussed. Each component of the FPGA subsystem is elaborated in details.

# Chapter 5

# Testing

Testability is a quality and feature of modern FPGA system design. This chapter mainly discusses the two test methodologies in this design. One method is the simulation of the system, which is one part of the FPGA design flow. Altera's *Quartus II* is used in this phase. The other method is design verification. It is about testing the floating-point A/D converter after programming FPGA, which is about testing the "real" operation of the parallel architecture of the floating-point A/D converter. MATLAB is used in this testing.

## 5.1 Simulation of the parallel FP-A/D converter

Quartus version 2.0 is used in the whole FPGA subsystem design flow. FPADC "project" consists of the complete set of design files, assignment files, simulation files, system settings, and hierarchy information for a design. The project in this design demonstrates the top-down design methodology. First, top-level Block Design File (FPADC.bdf) that contains blocks representing the lower-level design files is created. Next, the lower-level VHDL Design Files (.vhd) present these blocks. Then, a custom megafunction variation that is instantiated by one of the lower-level design files is created to complete the Design Entry module.

The Quartus II Compiler consists of a series of modules that check the design for errors, synthesize the logic, fit the design into an Altera device, and generate output files for simulation, timing analysis, and device programming. The Compiler first extracts information that defines the hierarchical connections between a project's design files and checks the designs for basic design entry errors. Then, it creates an organizational map of the design and combines all design files into a flattened database that can be processed efficiently.

Verification and simulation is one of the most important phases in FPGA design flow. Normally the average effort in the whole FPGA design is 70%. Simulation is to test a design thoroughly to ensure that it responds correctly in every possible situation before programming or configuring a device. Input vectors must be supplied as the stimuli for the Quartus II Simulator. The Simulator uses these input vectors to simulate the output signals, which a programmed device would produce under the same conditions. In a typical simulation session, the multiple sets of input vectors are created and the resulting outputs are checked.

Functional and timing simulations with the Simulator are performed. Functional simulation tests only the logical operation of a design, while timing simulation tests both the logical operation and the worst-case timing for the design in the target device. We create a Vector Waveform File (***.vwf**), specify Simulator settings, run a timing simulation, and analyze the simulation results.

In the FPGA-based the floating-point analog-to-digital conversion subsystem design, UART is the first functional block needed to test. Because UART is to realize the

serial communication between FPGA and PC via RS232 cable, we need to make sure it works well before we can send the control command from PC to FPGA and get the computing data (exponent) and mantissa from FPGA. Therefore, UART simulation is first presented below. Then, the simulation of FPADC FPGA subsystem is elaborated.

### 5.1.1 UART Simulation

The module under tests are UART_INTERFACE_ADC.vhd and UART.vhd, which includes the three sub-modules: TRANSMITTER.vhd, RECEIVER.vhd, DIV_CLK.vhd.

The main test block diagram is shown in Figure 5.1

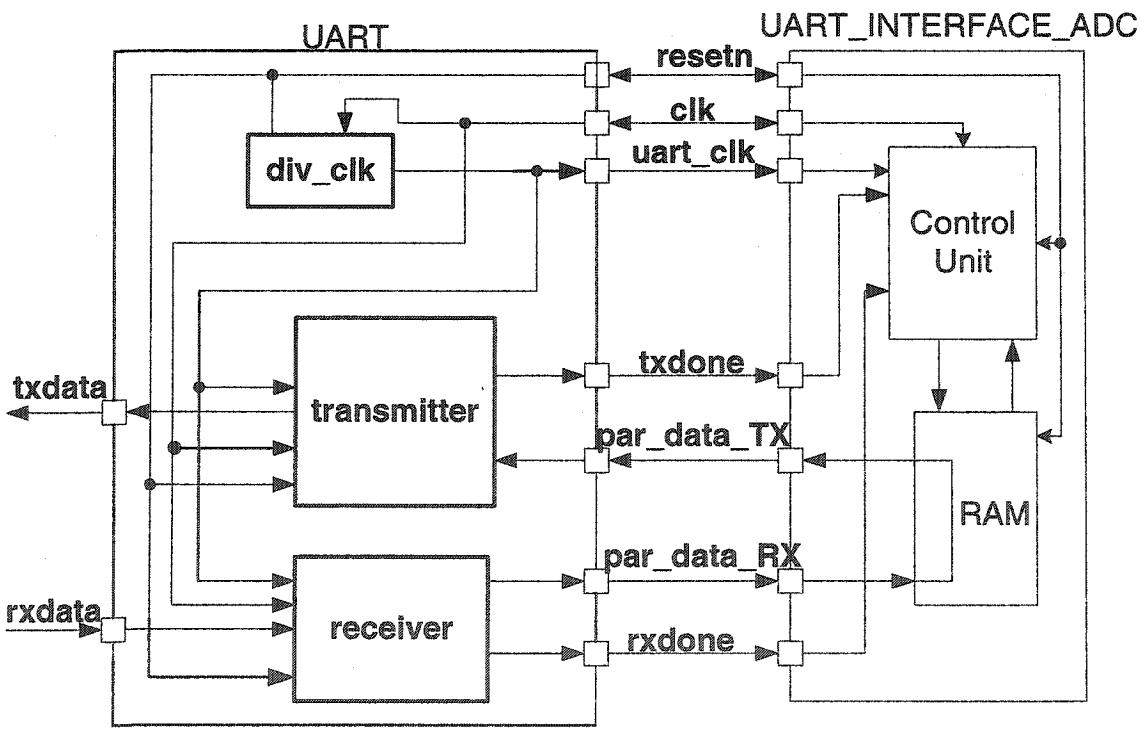


Figure 5.1 Block diagram of UART functionality test

This architecture is used to test the UART function. The data input to UART rxdata are a sequence of 10 bits binary form data (1 start bit, 8 data bits and 1 stop bit). The data output from UART txdata are also a sequence of 10 bits binary form data (1 start bit, 8 data bits and 1 stop bit). In UART_INTERFACE_ADC block, the serial data received from UART receiver is written to RAM. Then the content of the RAM is read to transmitter. UART transmitter then sends the serial data "rxdata" out. If these data "rxdata" are the same as "txdata", then it shows the UART functions correctly.

The simulation waveform can be seen in Figure 5.2.

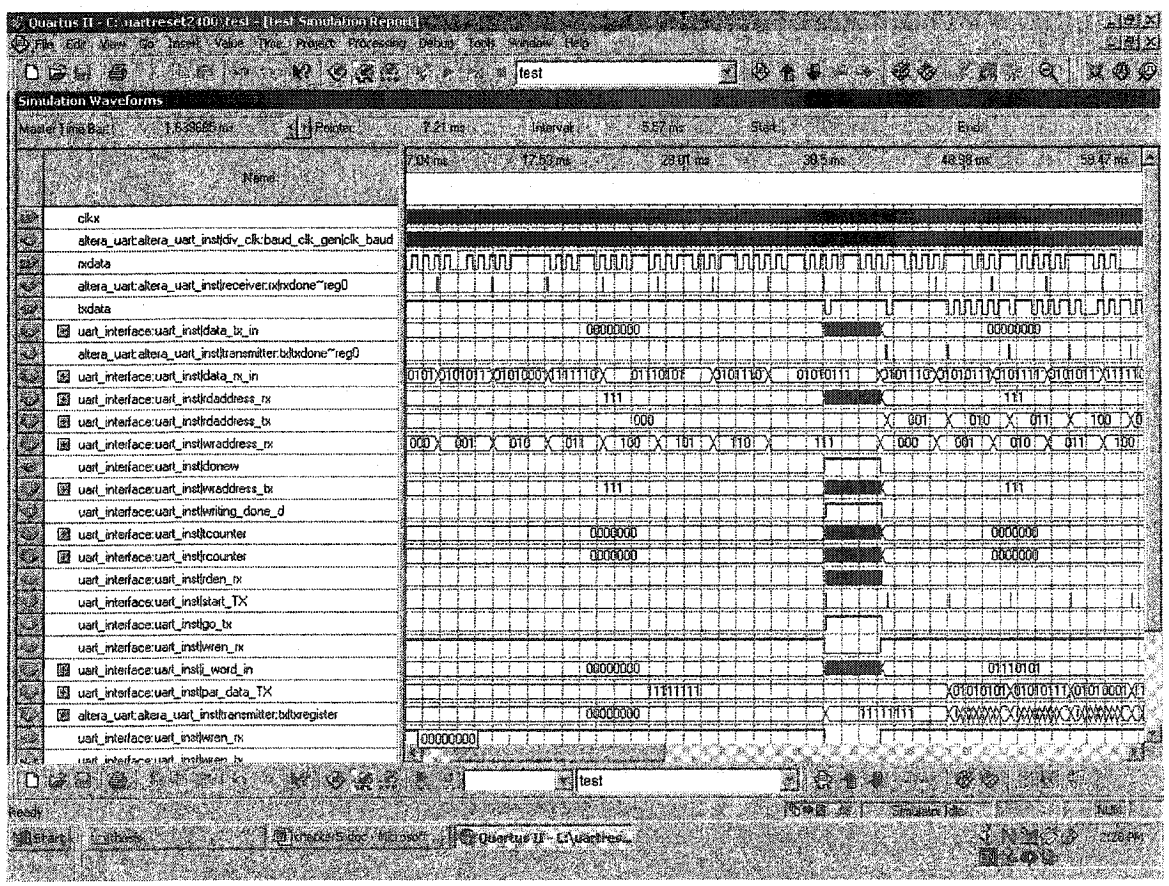


Figure 5.2 UART simulation waveform

In Figure 5.2, CLKX input clock is 40MHz. The UART working system clock is 10MHz, which is fed from PLL. It is easily seen that the input serial data to the RECEIVER " rxdata" are in 8-bit binary data form "0101,0101","0101,0111", "0101,0001", "1111,1101", "0111,0101", ..... plus one start bit "0" and one stop bit "1". The output serial data from the TRANSMITTER " txdata" are also in 8-bit binary data form with "0101,0101","0101,0111", "0101,0001", "1111,1101", "0111,0101", .....plus one start bit "0" and one stop bit "1". It shows that UART components function correctly.

### 5.1.2 FPADC Simulation

The next module under test is the top-level module FPADC.vhd, which includes all sub-modules: FP-ADC.vhd, UART.vhd, CLKPLL.vhd, UART_INTERFACE_ADC.vhd.

The simulation waveform is shown in Figure 5.3

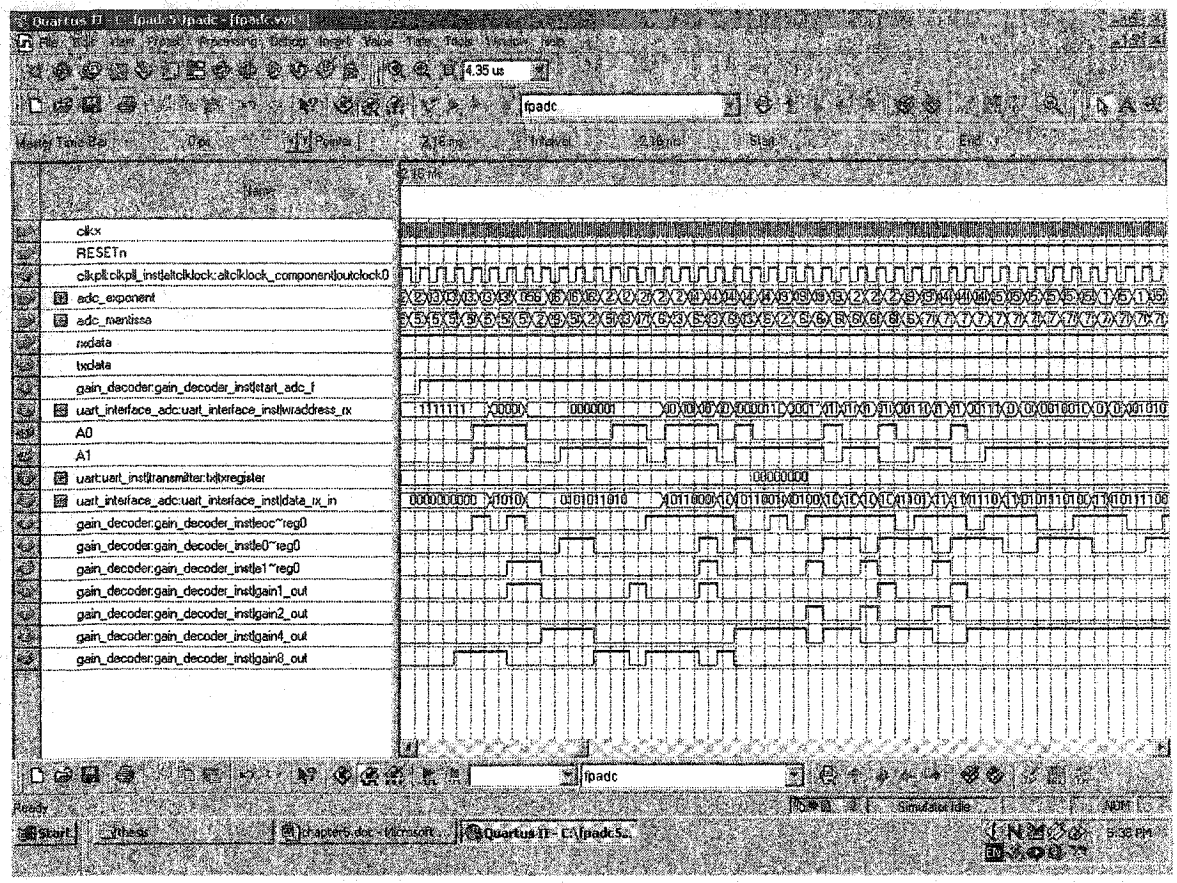


Figure 5.3 FPADC simulation waveform

As we discussed in chapter 4, the PGA gains and the correspondent exponent should have the following relationships shown in Table 5.1.

| | Gain=8 | Gain=4 | Gain=2 | Gain=1 |
|---|---|---|---|---|
| $e_0$ | 0 | 1 | 0 | 1 |
| $e_1$ | 0 | 0 | 1 | 1 |
| $e_2$ | 1 | 1 | 1 | 1 |

Table 5.1 The relationship of the PGA gains and exponent

The exponent has 3 bits: $e_2\ e_1\ e_0$. When PGA gain equals 8, the exponent should be "100". Similarly, when gain is 4, the exponent should be "101". Normally, we just need 2 bits $e_1\ e_0$ to represent the exponent.

In Figure 5.3, when Gain8=1, it means the PGA gain equals 8, the exponent $e_1$ = '0' and $e_0$ = '0'. When Gain4=1, it means the PGA gain equals 4, the exponent $e_1$ = '0' and $e_0$ = '1'. When Gain2=1, the PGA gain equals 2, the exponent $e_1$ = '1' and $e_0$ = '0'. When Gain1=1, the PGA gain equals 1, the exponent $e_1$ = '1' and $e_0$ = '1'. It can be seen that the PGA gains and the correspondent exponent are correct according to Table 5.1.

The input adc_exponent[9..0] from A/D-E are read to FPGA . The coarse conversion begins. The PGA gain is set according to the value of adc_exponent. In our design, we only use 8 bits of adc_exponent. Therefore, adc_exponent[9..2] was chosen. It is in 2's complement form from "01111111" to "10000000"; adc_exponent[9] is the MSB bit. If the current PGA gain is equal to the PGA gain stored in the register PGAgain1, PGAgain2, PGAgain4 and PGAgain8, the handshake signal " EOC" is asserted high; otherwise, it is "0" and A0 and A1 are set according to the gain value and they control the PGA. The input data adc_mantissa[9..0] from A/D-M converter are read to RAM after PGA output is set. In Figure 5.3, when the coarse conversion begins, adc_exponent [9..0]= "123"in hexadecimal, the PGA gain is stored as 1. On the next clock, adc_exponent [9..0] = "030" in hexadecimal, the current gain is 8 and not the same as the stored one, such that the PGA gain is updated from 0 to 8. On the next clock,

adc_exponent [9..] = "031" in hexadecimal, the current gain is still 8. Therefore, A0 and A1 are set to "0" and "0", respectively, and "EOC" is "1".

It is easily seen that the simulation waveforms correspond to the functions of the parallel architecture of the floating-point A/D conversion, which proves that the FPGA design is correct.

After simulation, we use the Programmer and supported programming hardware to easily program or configure a working device in minutes. The Programmer allows you to use files generated by the Compiler to program and configure all Altera® devices and configuration devices supported by the Quartus® II software. After a successful compilation, we download configuration data into a device through the ByteBlasterMV communications cables and configure devices in JTAG mode.

## 5.2 Floating-point analog-to-digital converter testing

The whole system of the parallel architecture of the floating-pointing analog-to-digital converter is shown in Figure 5.4.

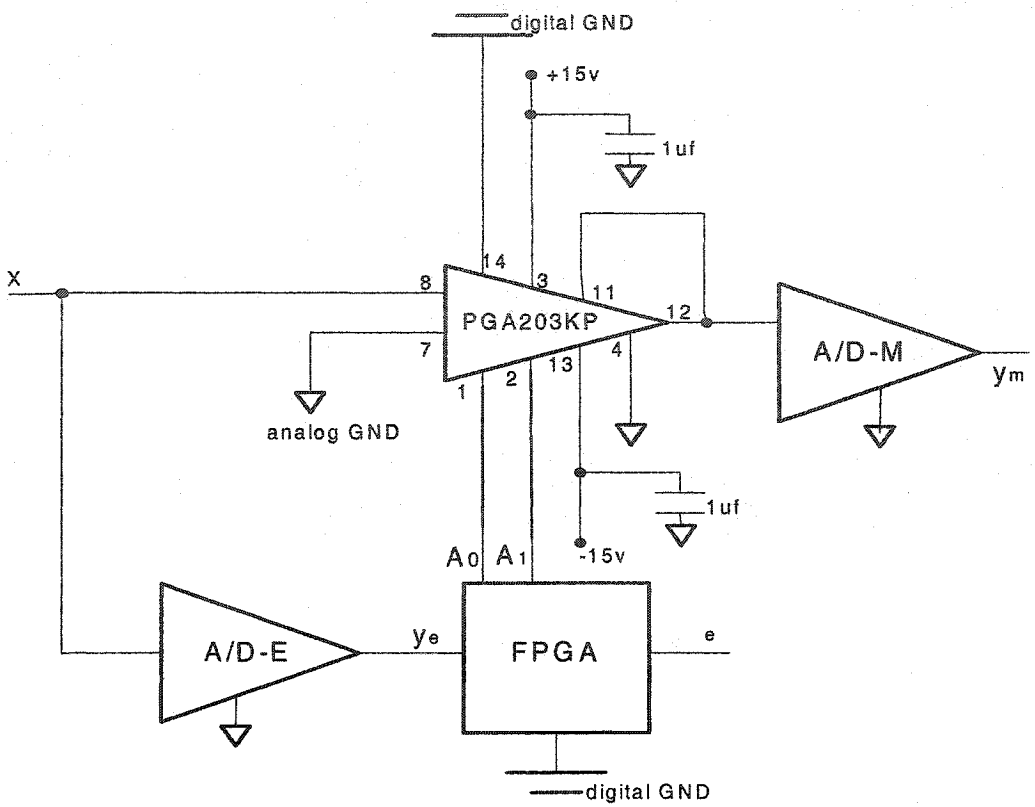


Figure 5.4 Block diagram of testing systems

PGA203KP is selected as Programmable Gain Amplifier (PGA). It is a digitally controlled gain amplifier with binary model. The gains are 1,2,4 and 8. Gain selection is accomplished by the application of a 2-bit digital word to the gain select inputs. The logic inputs are referred to their own separate digital common pin, which can be connected to any voltage between the minus supply and 8v below the positive supply.

The A/D converters on the APEX DSP board produce 10-bit samples at a sample rate of 10 MSPS. The data output format of the A/D converters is two's complement. The MSB AD[9] is the sign-bit. For a resolution of 8 bits, the output of the A/D converter

from AD[9] to AD[2] is chosen. The full scale of the voltage input to A/D converter is 1V.

The A/D converters on the APEX DSP are unipolar, therefore, the 8 bits of their output correspond to the following voltages:

"10000000" correspond to $V_{REF^-} = 1.2V$.

"00000000" correspond to 1.7V = midpoint

"01111111" correspond to $V_{REF^+} = 2.2V$.

The MATLAB serial port interface is used to access directly the EP20K200EBC652-1X FPGA via the PC's serial port. This interface is established through a serial port SW object. The serial port object supports functions and properties that allow to configuring the PC serial communications port, to use serial port control pins, to write and read data, to use events and callbacks and to record information onto disk.

The serial port "COM2" is used to realize the serial communication with FPGA. The baud rate is 9600bps, no parity bit and with one stop bit. The total number of bytes that can be stored in the input buffer during a read operation is 128. The maximum time (in seconds) to wait to complete a read or write operation is 100s. The terminator is a carriage return followed by a line feed. The serial port object continuously queries the device to determine if data is available to be read. If data is available, it is automatically

read and stored in the input buffer. So, the main properties of the serial port object are following.

```
s=serial('com2');
set(s,'Baudrate',9600,'Parity','none','StopBits',1);
set(s,'InputBufferSize',1024,'Timeout', 100);
set (s, 'Terminator','CR/LF');
set(s,'ReadAsyncMode','continuous');
```

When starting to A/D conversion, the binary form of the control command to FPGA is correspondence to " 01000011". So, write the binary data to the serial port object s.

```
fwrite(s, 65,'int8').
```

When the FPGA accepts this control command, it can start to read the output data of the A/D converter. When the data are available in the serial port object, the binary data can be read from the FPGA and be returned to Out.

```
Out = fread (s, 2^n , 'int8')
```

Where 2^n is the total number of the data read, $n \geq 0$.

'int8' means the 8-bit integer.

The waveforms can be obtained from MATLAB and shown as following.

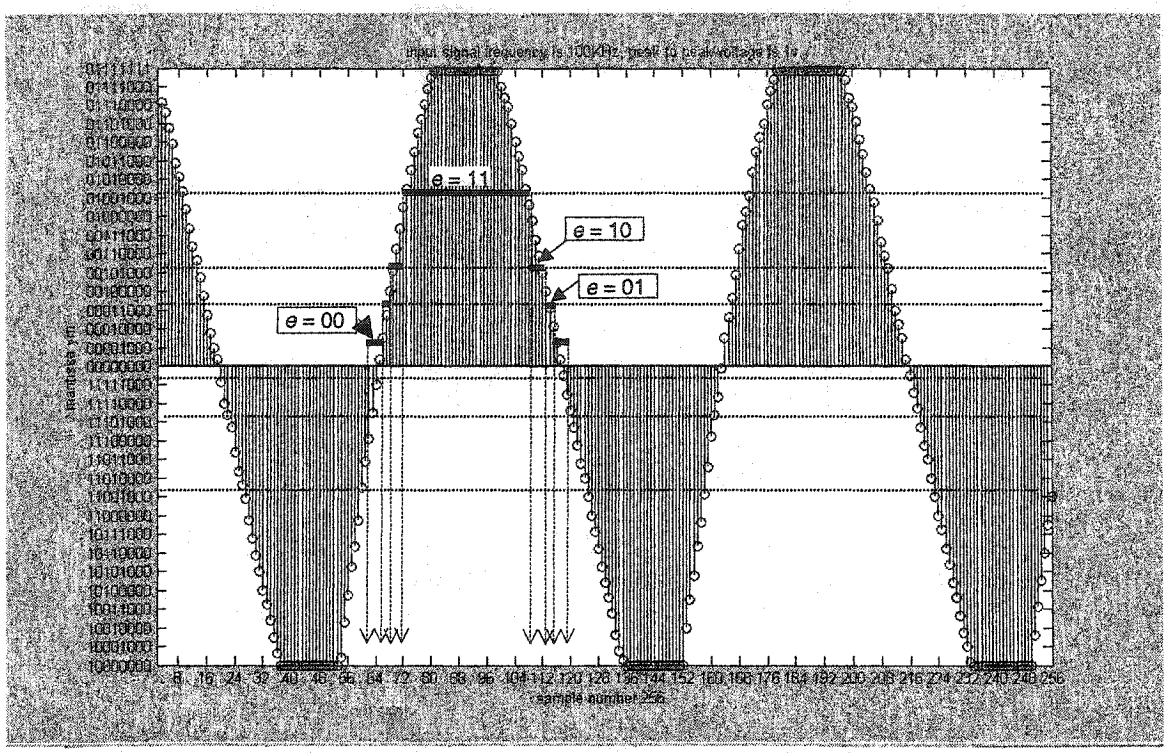


Figure 5.5 mantissa $y_m$ when the peak-to-peak voltage 1.0v

Figure 5.5 shows the waveforms of the mantissa $y_m$ when the input signal frequency is 100KHz and the peak-to-peak voltage is 1v. The number of samples is 256. For an convenient verification of the way the exponent is derived by FPGA, the PGA gain was set to 1 by interrupting the lines $A_1A_0$ between PGA and FPGA, and enforcing $A_1A_0$=0.

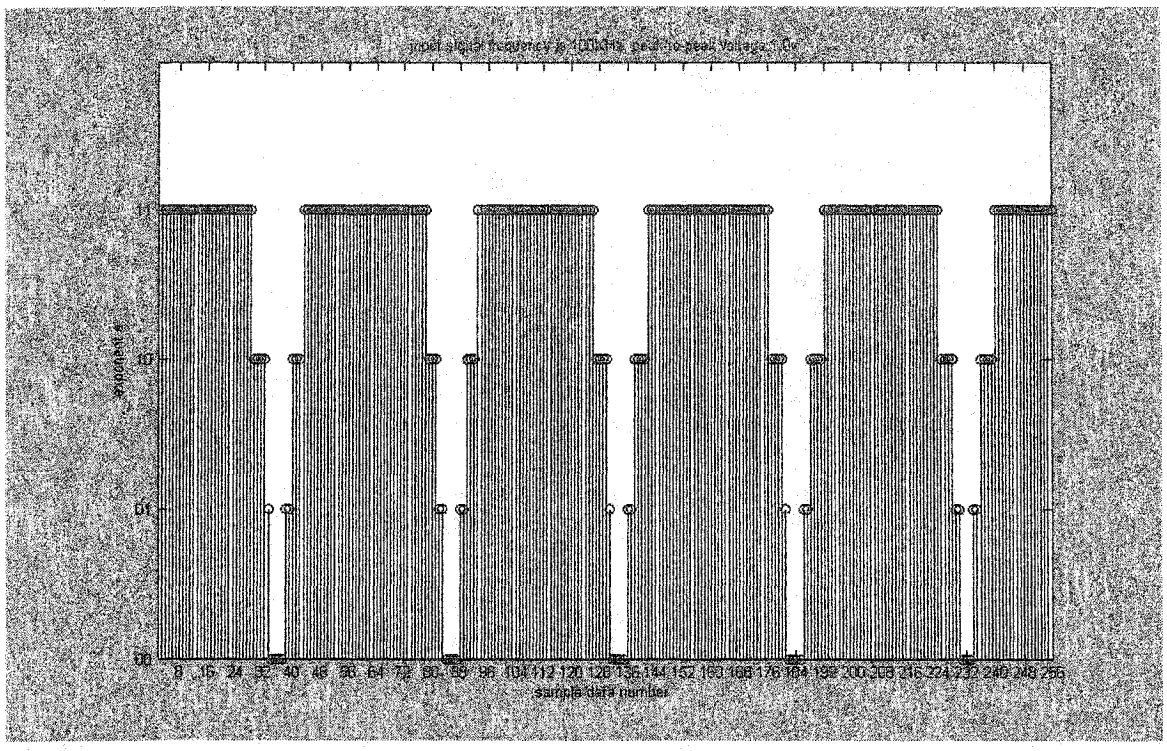


Figure 5.6 exponent $e$ when the peak-to-peak voltage 1.0v

Figure 5.6 shows the waveforms of the exponent $e$ when the input signal frequency is 100KHz and the peak-to-peak voltage is 1v. The number of samples is 256.

It is clearly observed that the regions marked in Fig. 5.5. with "$e$ = 00", "$e$ = 01", "$e$ = 10", and "$e$ = 11", relate correctly to the corresponding zones of Fig. 5.6.

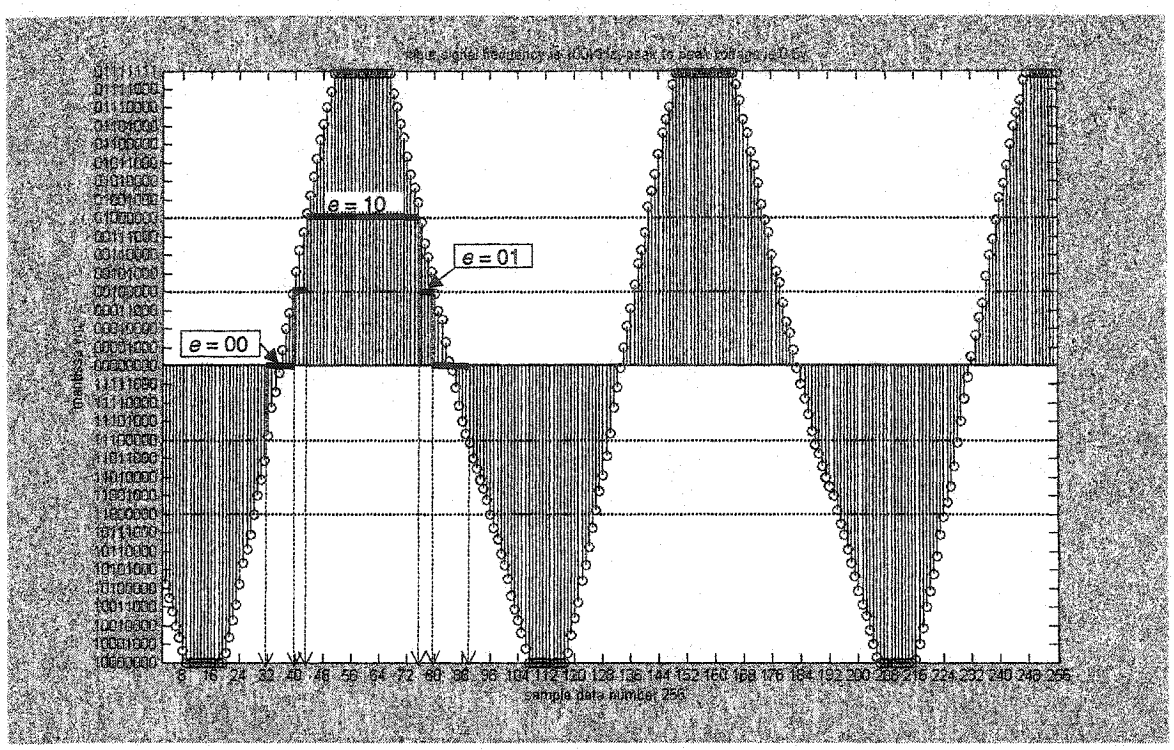


Figure 5.7 mantissa $y_m$ when the peak-to-peak voltage 0.5v

Figure 5.7 shows the waveforms of the mantissa $y_m$ when the input signal frequency is 100KHz and the peak-to-peak voltage is 0.5*V*. The number of sample is 256.

To stretch the quantized signal to cover the span the quantization range of AD-M and for an easier verification of the way the exponent is derived by FPGA, the PGA gain was set now to 2 by enforcing $A_1A_0$=01.

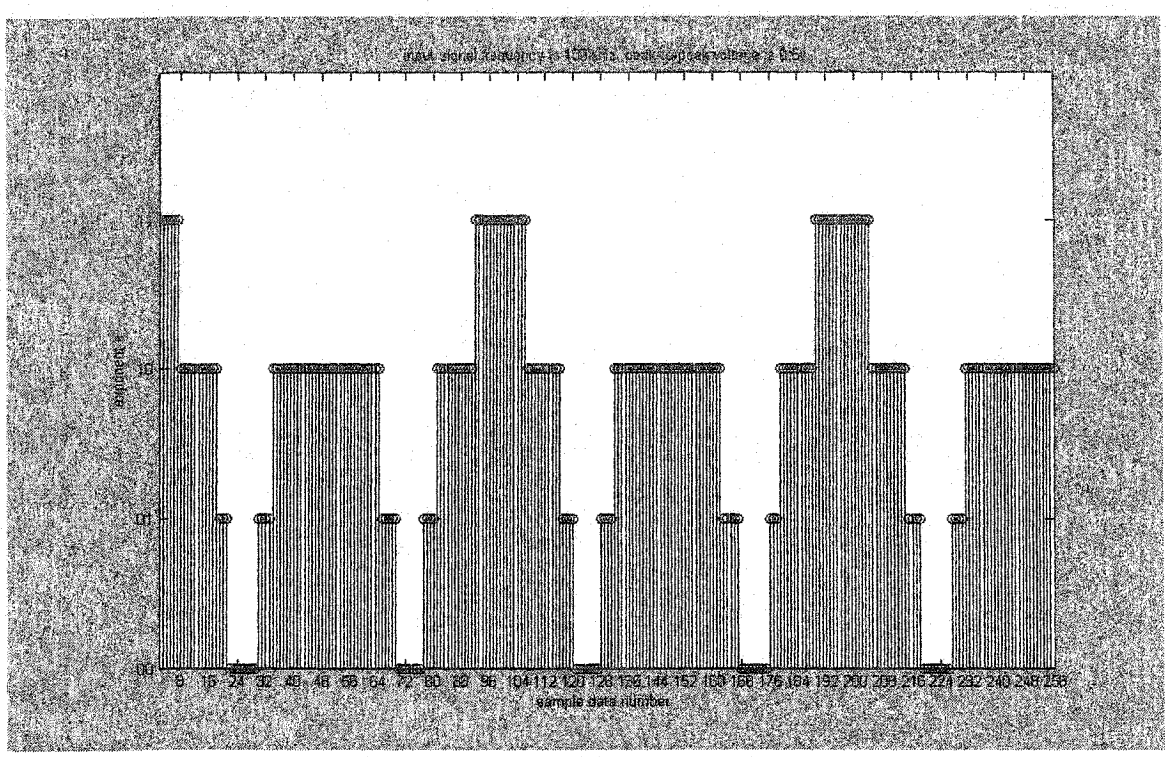


Figure 5.8 exponent $e$ when the peak-to-peak voltage 0.5v

Figure 5.8 shows the waveforms of the exponent $e$ when the input signal frequency is 100KHz and the peak-to-peak voltage is 0.5$V$.

It is easily observed that the regions marked in Fig. 5.7. with "$e$ = 00", "$e$ = 01", "$e$ = 10", generally relate correctly with the corresponding zones of Fig. 5.8. Since in this experiment the input signal's amplitude has exceeded some times 0.5$V$, the exponent in Fig. 5.8 increased up to "$e$ = 11" for those periods of time.

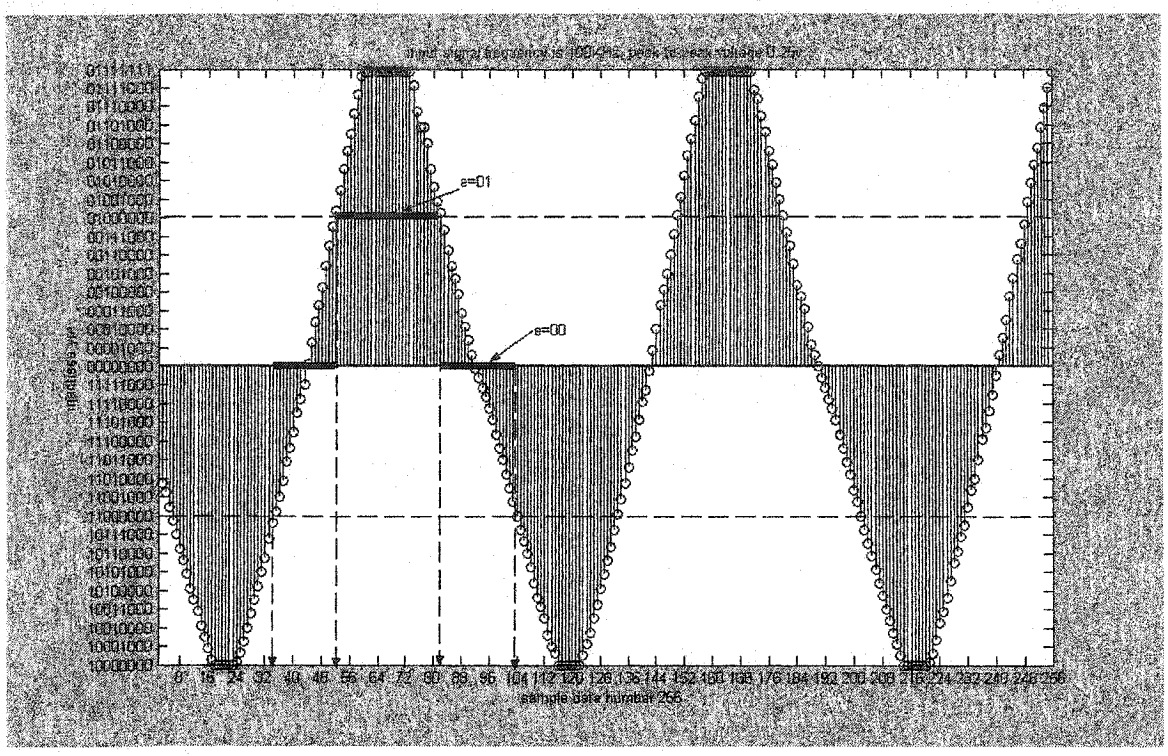


Figure 5.9 mantissa $y_m$ when the peak-to-peak voltage 0.25v

Figure 5.9 shows the waveforms of the mantissa $y_m$ when the input signal frequency is 100KHz and the peak-to-peak voltage is 0.25v. The number of sample is 256.

In order to make the quantized signal to cover the span of the quantization range of AD-M and for an easier verification of the way the exponent is derived by FPGA, the PGA gain was set now to 4 by enforcing $A_1A_0$=10.

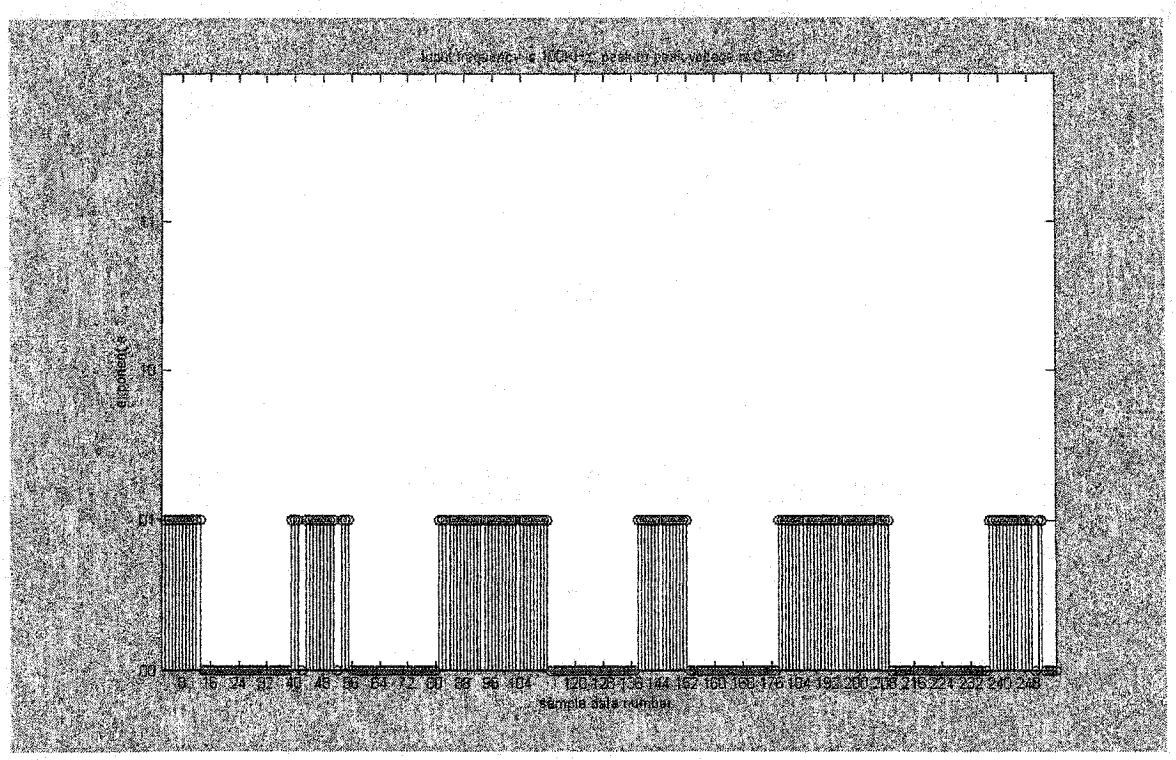


Figure 5.10 exponent $e$ when the peak-to-peak voltage 0.25v

Figure 5.10 shows the waveforms of the exponent $e$ when the input signal frequency is 100KHz and the peak-to-peak voltage is 0.25v. The sample number is 256.

It is easily seen that the regions marked in Fig. 5.9. with "$e = 00$" and "$e = 01$" , generally relate correctly with the corresponding zones of Fig. 5.10

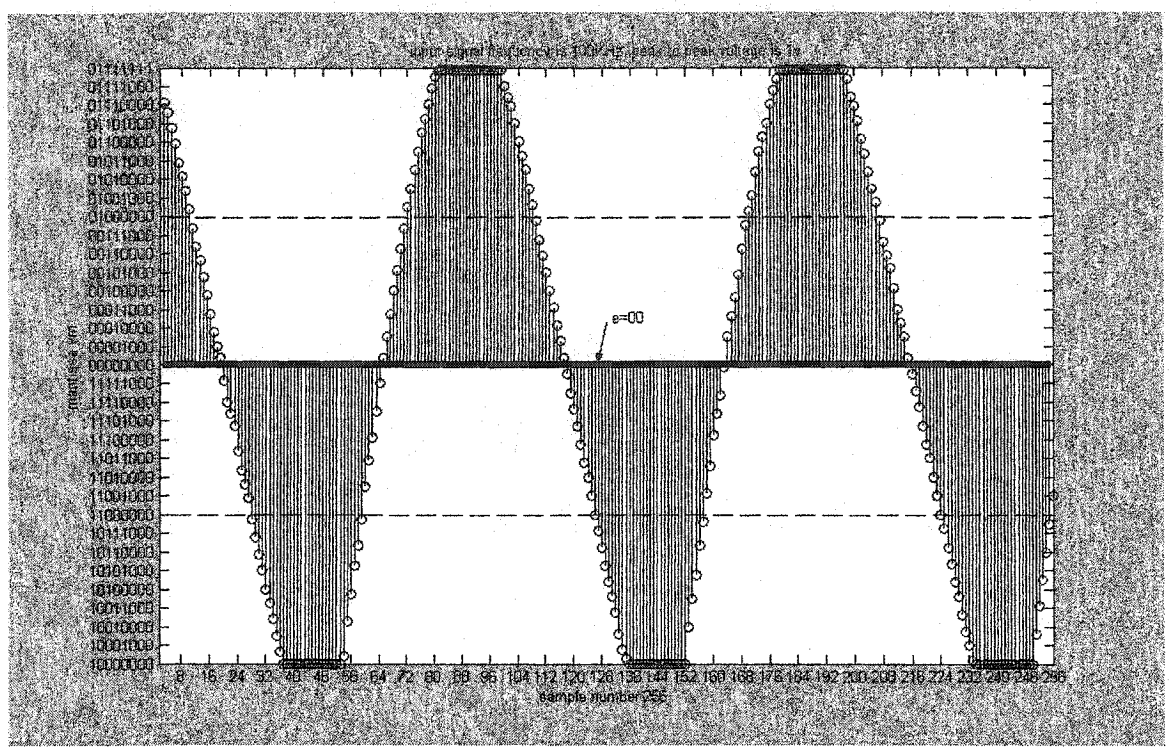


Figure 5.11 mantissa $y_m$ when the peak-to-peak voltage 0.125v

Figure 5.11 shows the waveforms of the mantissa $y_m$ when the input signal frequency is 100KHz and the peak-to-peak voltage is 0.125v. The sample number is 256.

For an easier verification of the way the exponent is derived by FPGA, the quantized signal is spanned over the quantization range of AD-M by setting the PGA gain now to 8 by enforcing $A_1A_0$=11.

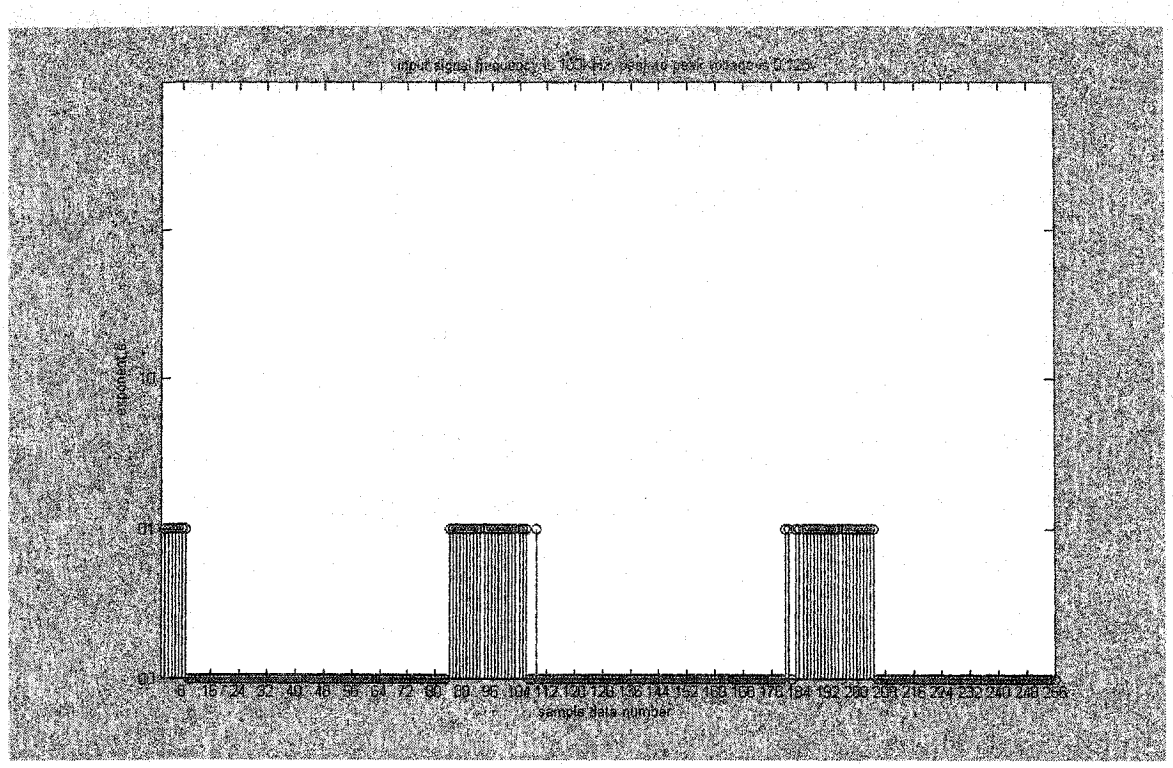


Figure 5.12 exponent $e$ when the peak-to-peak voltage 0.125v

Figure 5.12 shows the waveforms of the exponent $e$ when the input signal frequency is 100KHZ and the peak-to-peak voltage is 0.125$V$. The sample number is 256.

It is easily observed that the regions marked in Fig. 5.11. with "$e$ = 00", generally relate correctly with the corresponding zones of Fig. 5.12. Since in this experiment the input signal's amplitude has exceeded some times 0.125$V$, the exponent in Fig. 5.12 increased up to "$e$ = 01" for those periods of time.

It should be pointed that there is saturation in Figure 5.5, 5.7, 5.9 and 5.11. It is because the voltage that is applied to the ADC input exceeds in those portions the maximum input voltage of 1V. When the input is amplified by the PGA, it is limited to 1V. Increasing the sample data number can get the best waveforms.

## 5.3 Summary

The two test methodologies in this design were mainly introduced in this chapter. One is component simulation of the FPGA system, which is one part of the FPGA design flow; Quartus II is used in this phase. The other is testing the floating-point A/D converter after programming FPGA, which is about testing the "real" operation of the parallel architecture of the floating-point A/D converter, MATLAB is used in this testing.

# Chapter 6

# Conclusion and Future Work

This research work has evaluated a new parallel architecture of floating-point analog-to-digital converters implemented by Field Programmable Gate Array (FPGA), which employs uniform quantizers to keep the precision of the sequential floating-point A/D converter and minimizes the conversion time close to the characteristics of the non-uniform one-cycle quantizer. The definitions of the class of floating-point A/D converters are extended; the characteristics of floating-point A/D converters versus fix-point A/D converter are evaluated. The different architectures of floating-point A/D converters, based on a study of the state of the art of the domain are reviewed.

The whole FPGA design flow is carried out on this FPGA based on the parallel architecture of the floating-point A/D converter (Ref: Chapter 4). An experimental PC based test-bed, which is able to test the functionality of FP-ADCs and to evaluate their performance, is developed. This test-bed was used to evaluate the FP-ADC characteristics, facilitating thus the comparison of the simulation of the parallel architecture of the floating-point A/D converter with its FPGA hardware implementation.

Over all, the parallel architecture of the floating-point analog-to-digital converter has the characteristics of keeping the minimum conversion time as well as the remarkable precision of the uniform quantizers.

The future work for this research is that the scope of the analysis and comparison in this thesis can be broadened to address many studies in floating-point A/D converters.

First is to evaluate the new parallel architecture of the floating-point analog-to-digital converter more thoroughly under more scenarios. The second item is to reduce the acquisition time by a more accurate prediction measurement domain while minimizing the quantization noise.

# References


1. Arild Lacroix, "Floating-point signal processing-arithmetic, roundoff-noise, and limit cycles", IEEE Proceedings Int. Symposium On Circuits and Systems, Espoo, Finland, pp2023-2030, June 7-9, 1988.

2. J. Kontro, K. Kalliojarvi, and Y. Neuvo," Floating-point arithmetic in signal processing", Proc. 1992 IEEE Int. Symposium On Circuits and Systems, San Diego, CA, USA, pp1784-1791, May 10-13, 1992.

3. The Institute of Electrical and Electronics Engineers, "IEEE standard for binary floating-point arithmetic, ANSI/IEEE Std 754-1985, New York, NJ, USA, Aug. 1985.

4. The Institute of Electrical and Electronics Engineers, "IEEE standard for radix-independent floating-point arithmetic, ANSI/IEEE Std 854-1987, New York, NJ, USA, Oct. 1987.

5. K. Kalliojarvi, J. Kontro, and Y. Neuvo, " Novel Floating-point A/D- and D/A-conversion methods", Proceeding 1994 IEEE Int. Symposium On Circuits and Systems, London, UK, pp1-4 vol.2, May30-June 2.

6. K. Kalliojarvi, J. Kontro, and Y. Neuvo, " Use of short floating-point formats in audio applications", IEEE Transaction On Consumer Electronics, Rosemont, IL, USA AUG. 1992 Volume: 38 Issue: 3, pp200-207.

7. K. Tsukamoto, T. Watahiki, and T. Miyata, "A floating-point A/D converter using switched-capacitor array", Electronic and communications in Japan, 1986, Pt.2, 69, pp68-75.

8. S. K. Sharma, G. O. Tomo, K. Tsukamoto, and T. Miyata, "A floating-point A/D converter uses low resolution DAC to get wide dynamic range", International Journal of Electronics, 1988, Vol. 64, No.5, pp787-794.

9. C. Q. Zhang, K. Tsukamoto, and T. Miyata, "Charge-balancing floating-point analog-to-digital converter using a cyclic conversion", International Journal of Electronics, 1993, Vol. 74, No.5, pp705-716.

10. A. V. Oppenheim and R. W. Schsfter, "Discrete-Time Signal Processing", Prentice-Hall, Englewood Cliffs, 1989.

11. F. Francescon, and F. Maloberti, "A low power Logarithmic A/D converter", Proceeding s of IEEE International Symposium on Circuits and Systems, vol.1, Atlanta, USA, pp 473-476, 1996.

12. B. A. Hafeth and M. A. H. Abdul-Karim, "Logarithmic analogue-to-digital converter based on a non-linear analogue-to-digital converter", International Journal of Electronics, 1985, 58, pp503-508.

13. G. Haller and D. R. Freytag, "Analog floating-point BiCMOS sampling chip and architecture of the BaBar CsI calorimeter frond-end electronics system at the SLAC B-factory", IEEE Trans. Nuclear. Science, pt.2, vol.43, pp1610-1614, June 1996.

14. J. Sevenhans and Z. Y. Chang, "A/D and D/A converters for telecommunication", IEEE Circuits and Devices, pp32-42, 1998.

15. D. U. Thompson and B. A. Wooley, "A 15-b pipelined CMOS floating-point converter", IEEE journal of Solid-state circuits, Vol.36, Issue 2, Feb. 2001, pp299-303.

16. J. Yuan and J. Piper, "Floating-point analog-to-digital converter", Proceedings of the 6th IEEE International Conference On Circuits and Systems, Vol.3, 1999, pp1385-1388.

17. J. Piper and J. Yuan, IACAS 2001, "Realization of a floating-point A/D converter", IEEE International Symposium on Circuits and Systems, Vol.1, pp404-407, May 6-9, 2001, Sydney, Australia.

18. V. Z. Groza, "High resolution Floating-point Analog-to-digital Converter", IMTC1999, Proceedings of the 16th IEEE, vol.3, Instrumentation & Measurement Technology Conference, pp1663-1666.

19. V. Z. Groza, "High resolution Floating-point Analog-to-Digital Converter", IEEE Transactions on instrumentation and measurement, Vol.5, No.6, pp1822-1829, Dec.2001.

20. V. Z. Groza and B. Dzerdz, " FPGA Based Implementation of a Floating-Point Analog-to-Digital Converter", 4-th IEE International Conference on Advanced A/D and D/A Conversion Techniques and their Applications & 7-th European

Workshop on ADC Modelling and Testing ADDA&EWADC 2002, pp.143-146, Prague, Czech Republic, June 26-28, 2002.

21. V. Z. Groza, "Floating-Point ADC Optimized for Acquisition of Deterministic Signals," IEEE Instrumentation and Measurement Technology Conference IMTC 2002, pp707 – 712, Anchorage, Alaska, 21-23 May, 2002.

22. L. Grisoni, A. Heubi, P. Balsiger and F. Pellandini, "Implementation of a micro power 15-bit 'floating-point' A/D converter", International Symposium on Lower Power Electronics and Design, 1996, pp247-252.

23. J. Yuan, "Floating-point analog-to-digital converter", US patent 6,317,070, Nov.13, 2001.

24. Carpenter, J. Robert, Yee, and W. Kenneth, " High speed, wide dynamic range analog-to-digital conversion", US patent 4,069,479, Jan. 17, 1978.

25. B. Fowler, A. E. Gamal, and D. Yang, "Method and apparatus for converting a low dynamic range analog signal to a large rang floating-point digital representation", US patent 6,369,737, April 2002.

26. Beauducel Claude and Fouquet Pierre, "Device for amplifying and sampling multiplexed analog signals", US patent 4,774,474, Sept. 1998.

27. Rialan Joseph and Cretin Jacques, "Method for amplifying multiplexed signals through different gain amplification units for seleting an optimim gain signal and device therefore", US patent 4,449,120, May 1984.

28. Rudy van de Plassche, "Integrated Analog-to-Digital and Digital-to-Analog Converters", ISBN: 0-7923-9436-4, Kluwer Academic Publishers.

29. D. F. Hoeschele, "Analog-to-digital and Digital-to-analog Conversion Techniques.", ISBN: 0-471-57147-4, John Wiley & Sons.

30. B. Razavi, "Principles of Data Conversion System Design", ISBN: 0-7803-1093-4, IEEE Press.

31. B. Leibowitz, "Design of a single cycle floating point A/D converter", Berkley.

32. G. Knittel, "A fast logarithm converter", Proceedings of 7th IEEE ASIC Conference, Rochester, NY, Sep. 19, 1994, pp450-453.

33. A. Gersho, "Principles of quantization", IEEE Transactions On Circuits and Systems, vol. CAS-25, No. 7, pp427-436, July 1978.

34. R. M. Gray, " Quantization noise spectra ", EEE Transactions On Information theory, vol. 36, No.6, pp1220-1244, Nov. 1990.

35. B. Widow, I. Kollar, and M.C. Liu," Statistical theory of quantization", IEEE Transactions On instrumentation and Measurement, vol.45, No. 2, pp353-360, April 1996.

36. R. H. Walden, "Analog-to-digital converter survey and analysis", IEEE Transaction On selected areas in Communications, vol. 17, No. 4, pp539-549, April, 1999.

37. S. K. Tewksbury, F. C. Meyer, and H. K. Schoenwetter," Terminology related to the performance of S/H, A/D and D/A Circuits", IEEE Transaction On Circuits and Systems, vol. CAS-25, No.7, pp419-426, July 1978.

38. T. R. Viswanathan, "A level-crossing sampling scheme for A/D conversion", IEEE Transactions On Circuits and Systems-II: analog-to-digital signal processing, vol. 43, No. 4, pp335-339, April 1996.

39. A. Wurz and R. Manner, "Flexible high-speed fastbus master for data read-out and performance", IEEE Transactions On Nuclear Science, vol. 37, No. 2, pp256-261, April 1990.

40. Samir Palnitkar, "Verilog HDL –A guide to digital design and Synthesis", ISBN: 0-13-451675-3, SunSoft Press.

41. Motorola MCore: MMC2001 Reference Manual, Motorola, 1998

42. http://www.hal-pc.org/~clyndes/computer-arithmetic/floats.html

43. Altera Application Note 115: Using the ClockLock & ClockBoost PLL Features in APEX Devices.

# Appedix1

The relationships among the input signal $V_{in}$, gain, and exponent $e$

| ye7 | ye6 | ye5 | ye4 | ye3 | ye2 | ye1 | ye0 | mV | gain | e | exp | | ye7 | ye6 | ye5 | ye4 | ye3 | ye2 | ye1 | ye0 | mV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 128 | | 0 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | -8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 8 | 64 | | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | -16 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 16 | 32 | | 2 | | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | -24 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 24 | 32 | | 2 | | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | -32 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 32 | 16 | | 3 | | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | -40 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 40 | 16 | | 3 | | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | -48 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 48 | 16 | | 3 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | -56 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 56 | 16 | | 3 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | -64 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 64 | 8 | 3 | 4 | | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | -72 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 72 | 8 | 3 | 4 | | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | -80 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 80 | 8 | 3 | 4 | | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | -88 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 88 | 8 | 3 | 4 | | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | -96 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 96 | 8 | 3 | 4 | | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | -104 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 104 | 8 | 3 | 4 | | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | -112 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 112 | 8 | 3 | 4 | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | -120 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 120 | 8 | 3 | 4 | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | -128 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 128 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | -136 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 136 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | -144 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 144 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | -152 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 152 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | -160 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 160 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | -168 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 168 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | -176 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 176 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | -184 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 184 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | -192 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 192 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | -200 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 200 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | -208 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 208 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | -216 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 216 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | -224 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 224 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | -232 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 232 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | -240 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 240 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | -248 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 248 | 4 | 2 | 5 | | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | -256 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 256 | 2 | 1 | 6 | | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | -264 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 264 | 2 | 1 | 6 | | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | -272 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 272 | 2 | 1 | 6 | | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | -280 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 280 | 2 | 1 | 6 | | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | -288 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 288 | 2 | 1 | 6 | | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | -296 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 296 | 2 | 1 | 6 | | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | -304 |

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 304 | 2 | 1 | 6 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | -312 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 312 | 2 | 1 | 6 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | -320 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 320 | 2 | 1 | 6 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | -328 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 328 | 2 | 1 | 6 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | -336 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 336 | 2 | 1 | 6 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | -344 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 344 | 2 | 1 | 6 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | -352 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 352 | 2 | 1 | 6 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | -360 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 360 | 2 | 1 | 6 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | -368 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 368 | 2 | 1 | 6 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | -376 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 376 | 2 | 1 | 6 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | -384 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 384 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | -392 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 392 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | -400 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 400 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | -408 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 408 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | -416 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 416 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | -424 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 424 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | -432 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 432 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | -440 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 440 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | -448 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 448 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | -456 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 456 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | -464 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 464 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | -472 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 472 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | -480 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 480 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | -488 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 488 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | -496 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 496 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | -504 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 504 | 2 | 1 | 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | -512 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 512 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | -520 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 520 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | -528 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 528 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | -536 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 536 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | -544 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 544 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | -552 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 552 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | -560 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 560 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | -568 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 568 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | -576 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 576 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | -584 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 584 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | -592 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 592 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | -600 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 600 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | -608 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 608 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | -616 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 616 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | -624 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 624 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | -632 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 632 | 1 | 0 | 7 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | -640 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 640 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | -648 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 648 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | -656 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 656 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | -664 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 664 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | -672 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 672 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | -680 |

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 680 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | -688 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 688 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | -696 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 696 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | -704 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 704 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | -712 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 712 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | -720 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 720 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | -728 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 728 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | -736 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 736 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | -744 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 744 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | -752 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 752 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | -760 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 760 | 1 | 0 | 7 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | -768 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 768 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | -776 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 776 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | -784 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 784 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | -792 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 792 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | -800 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 800 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | -808 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 808 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | -816 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 816 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | -824 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 824 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | -832 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 832 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | -840 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 840 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | -848 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 848 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | -856 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 856 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | -864 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 864 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | -872 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 872 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | -880 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 880 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | -888 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 888 | 1 | 0 | 7 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | -896 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 896 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | -904 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 904 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | -912 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 912 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | -920 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 920 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | -928 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 928 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | -936 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 936 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | -944 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 944 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | -952 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 952 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | -960 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 960 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | -968 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 968 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | -976 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 976 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | -984 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 984 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | -992 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 992 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | -1000 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1000 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | -1008 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1008 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | -1016 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1016 | 1 | 0 | 7 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1024 |

# Appedix2

## *VHDL Coding*

```
-------------------------------------------------------------------------------------------

-- FPADC.vhd / top Entity Declaration
-- date: Aug., 7, 2003, version 1
-- Author: Shumin Shen
-- modification :version 3
-- date: sept.22, 2003 for  floating-point ADC
-------------------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity FPADC is
  PORT(          clkx        : in std_logic; ----------system clock 40MHZ
                 rxdata      : in std_logic; ----------UART
                 txdata      : out std_logic;
               RESETn        : in STD_LOGIC; ------reset alll system
            adc_mantissa     : in std_logic_vector(9 downto 0);
                                              ---A/D converter for mantissa
             adc_exponent: in std_logic_vector(9 downto 0);
                                              ---A/D converter for exponent
                 A0          : out std_logic; ----- to PGA
                 A1          : out std_logic
        );
end FPADC;

architecture top_level of FPADC is

component clkpll
  PORT(inclock : IN STD_LOGIC;
       locked : OUT STD_LOGIC;
       clock0 : OUT STD_LOGIC;
       clock1 : OUT STD_LOGIC );
end component;

component uart
  PORT(clk          : in std_logic;
    resetn          : in std_logic;
    uart_clk        : out std_logic;
    ser_data_TX     : out std_logic;
    start_TX        : in std_logic;
    par_data_TX     : in std_logic_vector(7 downto 0);
    txdone          : out std_logic;
    ser_data_RX     : in std_logic;
```

```
    par_data_RX        : out std_logic_vector(7 downto 0);
    rxdone             : out std_logic);
end component;

component  uart_decoder is
  PORT(              clk       : in std_logic;
                     rxdone    : in std_logic;
                 i_word_in     : in std_logic_vector( 7 downto 0);
                     resetn    : in std_logic;
                  start_adc    : out std_logic);
end component;

component dualclock is
  port       (end_adc_f, RESETN, CLK1, CLK2 : in std_logic;
                                  S3: inout std_logic);
end component;

component gain_decoder is
    port (     clk   : in std_logic;
           resetn    : in std_logic;
        adc_exponent : in std_logic_vector( 9 downto 0);
             A0      : out std_logic;
             A1      : out std_logic;
             e0      : out std_logic;
             e1      : out std_logic;
             e2      : out std_logic;
           start_adc: in std_logic;
             eoc     : OUT STD_LOGIC);
end component;

component uart_interface_adc
  PORT(              clk              : in std_logic;
                      clk_u           : in std_logic;
                    rxdone   : in std_logic;
                    txdone   : in std_logic;
                    start_TX: out std_logic;
                 start_adc   :in std_logic;
               EOC   : in std_logic;----------one conversion
           end_adc_f   : out std_logic;
           end_adc     : in std_logic;
              e0       : in std_logic;
              e1       : in std_logic;
              e2       : in std_logic;
                adc_mantissa  : in std_logic_vector(9 downto 0);
         resetn        : IN STD_LOGIC;
                    par_data_TX       : out std_logic_vector(7 downto 0) );
end component;

signal      clk ,clk_2:  STD_LOGIC;
signal      start_tx, start_tx_f:  STD_LOGIC;
signal      uart_clk :  STD_LOGIC;
signal data_tx :  STD_LOGIC_VECTOR(7 downto 0);
signal      rx_done :  STD_LOGIC;
signal      tx_done :  STD_LOGIC;
signal      data_rx:  STD_LOGIC_VECTOR(7 downto 0);
```

```
signal dataout: std_logic_vector(7 downto 0);
signal  end_adc_f, end_adc:  std_logic;
signal eoc, e0,e1,e2: std_logic;
signal start_adc: std_logic;

BEGIN

uart_inst : uart
PORT MAP(clk =>clk,
             resetn => resetn,
             ser_data_RX => rxdata,
             start_TX => start_tx,
             par_data_TX => data_tx,
             rxdone => rx_done,
             ser_data_TX => txdata,
             txdone => tx_done,
             uart_clk => uart_clk,
             par_data_RX => data_rx);

uart_decoder_inst: uart_decoder
PORT Map (         clk=>uart_clk,
                                rxdone=>rx_done,
             i_word_in=>data_rx,
                  resetn=>resetn,
                start_adc=>start_adc);

clkpll_inst : clkpll
PORT MAP(inclock => clkx,  ----------40Mhz
             clock0 => clk,    ----------10MHZ
             clock1 => clk_2);

uart_interface_inst : uart_interface_adc
PORT MAP(clk => clk,
             clk_u => uart_clk,
      ADc_mantissa=>ADc_mantissa,
      resetn => resetn,
             rxdone => rx_done,
             txdone => tx_done,
             par_data_TX => data_tx,
               eoc=>eoc,
                e0=>e0,
                e1=>e1,
                e2=>e2,
             start_TX => start_tx,
             start_adc=>start_adc,
             end_adc_f=>end_adc_f,
             end_adc=>end_adc);

gain_decoder_inst:  gain_decoder
port  map(    clk=>clk,
          esetn=>resetn,
          adc_exponent=>adc_exponent,
          start_adc=>start_adc,
            A0=>A0,
            A1=>A1,
```

```
            e0 =>e0,
            e1 =>e1,
            e2 =>e2,
            eoc=>eoc);

 dualclock_inst: dualclock
port map (end_adc_f=> end_adc_f,
        RESETN=>resetn,
        CLK1=>clk,
        CLK2=>uart_clk,
                S3=>end_adc );

end ARCHITECTURE top_level ;
```

```
----------------------------------------------------------------------------------------------------
-- uart.vhd
-- BASIC UART
-- sub-entities   : transmitter.vhd,  receiver.vhd,  div_clk.vhd
-- date : Feb. 23 , 2003
-- Author: Shumin Shen
----------------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity uart is
  port (
    clk             : in std_logic;
    resetn          : in std_logic;
    uart_clk        : out std_logic;
    ser_data_TX     : out std_logic;
    start_TX        : in std_logic;
    par_data_TX     : in std_logic_vector(7 downto 0);
    txdone          : out std_logic;
    ser_data_RX     : in std_logic;
    par_data_RX     : out std_logic_vector(7 downto 0);
    rxdone          : out std_logic
    );
end uart;

architecture syn of uart is

-- UART Transmitter
component transmitter
  port (
    clk             : in std_logic;
    resetn          : in std_logic;
    ser_data_TX     : out std_logic;
    start_TX        : in std_logic;
    par_data_TX     : in std_logic_vector(7 downto 0);
    txdone          : out std_logic
    );
end component;

-- UART Receiver
component receiver
   port (
    resetn          : in std_logic;
    clk             : in std_logic;
    ser_data_RX     : in std_logic;
    par_data_RX     : out std_logic_vector(7 downto 0);
    rxdone          : out std_logic
    );
end component;

-- UART baud clock generator
component div_clk
  generic (
    CLKX            : integer;
```

```
    BAUD                : integer   );
  port (
    clk             : in std_logic;
    resetn          : in std_logic;
    baud_clk        : out std_logic
    );
end component;

  signal ser_data_TX_int   : std_logic;
  signal txdone_int        : std_logic;
  signal par_dta_RX_int    : std_logic_vector (7 downto 0);
  signal rxdone_int        : std_logic;
  signal clkuart           : std_logic;

begin

  ser_data_TX <= ser_data_TX_int;
  txdone <= txdone_int;
  par_data_RX <= par_dta_RX_int;
  rxdone <= rxdone_int;
  uart_clk<=clkuart;

tx : transmitter
    port map (
      clk => clkuart,
      resetn => resetn,
      ser_data_TX => ser_data_TX_int,
      start_TX => start_TX,
      par_data_TX => par_data_TX,
      txdone => txdone_int);

rx : receiver
    port map (
      resetn => resetn,
      clk => clkuart,
      ser_data_RX => ser_data_RX,
      par_data_RX => par_dta_RX_int,
      rxdone => rxdone_int);

baud_clk_gen : div_clk
    generic map(
                    CLKX=>10000000,
                    BAUD=>9600 )
    port map (
      clk => clk,
      resetn => resetn,
      baud_clk => clkuart);

end architecture syn;
```

```
--------------------------------------------------------------------------
-- UART receiver module: receiver.vhd

--date: march 12,2003; version1.0
---update: April , 2, 2003. Version2.0
---author: shumin shen
--------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity receiver is
  port (
    resetn          : in std_logic;
    clk             : in std_logic;
    ser_data_RX     : in std_logic;
    par_data_RX     : out std_logic_vector(7 downto 0);
    rxdone          : out std_logic); --  -received data ready
end entity receiver;

architecture syn of receiver is

  constant word_len : integer :=8;
  type rx_states is (DETECT_START_BIT,SYNC,WAIT_TICK,SAMPLE,STOP);
  signal state                : rx_states;
  signal next_state           : rx_states;
  signal data_in              : std_logic;
  signal data_in_d            : std_logic;
  signal counter_b            : std_logic_vector(3 downto 0);
  signal reset_counter        : std_logic;
  signal par_datas            : std_logic_vector(7 downto 0);
  signal par_datah            : std_logic_vector(7 downto 0);
  signal shift                : std_logic;
  signal counter_r            : std_logic_vector(3 downto 0);
  signal count                : std_logic;
  signal rstcount             : std_logic;
  signal rxdone_int           : std_logic;

begin

 par_data_RX <= par_datah ;

--- synchronize the asynchrnous input
-- to the system clock domain
  process (clk, resetn)     --posedge clock, negege resetn
  begin
    if (resetn='0') then
      data_in_d <= '1';
      data_in <= '1';
    elsif (clk'event and clk = '1') then
      data_in_d <= ser_data_RX;
      data_in <= data_in_d;
    end if;
```

```
  end process;

-- Bit-cell counter
  process (clk, resetn)
  begin
    if (resetn='0') then ------------------------counter bytes
      counter_b <= "0000";
    elsif (clk'event and clk = '1') then
      if (reset_counter = '0') then      -- modification(2)
        counter_b <= "0000";
      else
        counter_b <= counter_b + "0001";
      end if;
    end if;
  end process;

--Shifte Register to hold the incoming serial data,LSB is shifted in first

  process (clk, resetn)
  begin
    if (resetn='0') then
      par_datah <= "00000000";
      par_datas <= "00000000";
    elsif (clk'event and clk = '1') then

      if (shift = '1') then
      par_datas(6 downto 0) <= par_datas(7 downto 1);
       par_datas(7) <= data_in;
      end if;
     if(  rxdone_int ='1')then
       par_datah<=par_datas;
      end if;
    end if;
  end process;

---RECEIVED BIT Counter
----This coutner keeps track of the number of bits received

  process (clk, resetn)              ---resetn=high level reset:=0
  begin
   if (resetn='0') then
     counter_r <= "0000";            --counter-r is
   elsif (clk'event and clk = '1') then

   if (rstcount = '1') then         ------------------(1)
       counter_r <= "0000";
    else
      if (count = '1') then
      counter_r <= counter_r + "0001";
       end if;
      end if;
    end if;
  end process;
```

```
-- State Machine - Next State Assignment

  process (clk, resetn)
  begin
    if (resetn='0') then
      state <= DETECT_START_BIT;
    elsif (clk'event and clk = '1') then
      state <= next_state;
    end if;
  end process;

-- State Machine - Next State and Output Decode, mealy type statemachine.
  process (state, data_in, counter_b, counter_r)

    variable reset_counter_var  : std_logic;
    variable shift_var  : std_logic;
    variable count_var  : std_logic;
    variable reset_count_var  : std_logic;
    variable rxdone_int_var  : std_logic;
  begin

    reset_counter_var := '0';   ----------------------------(2)
    shift_var := '0';
    count_var := '0';
    reset_count_var := '1';   -----------------------------(1)
    rxdone_int_var := '0';
    case state is
      when DETECT_START_BIT =>
            if (not data_in = '1') then
              next_state<= SYNC;     ----------------------
            else
              next_state<= DETECT_START_BIT;
              reset_count_var := '1'; -- place the RECEIVED bit counter in rst state
              rxdone_int_var := '0';   ------?-
            end if;
---the stae machine wait for 1/2 bitcell in order to find the bit-cell center,
---a bit-cell is 1 baud tick and corresponds to 16 uart_clk ticks.
---1/2 bitcell corresponds to 8 uart_ticks. here is 4 not 8 because synchronizer uncertainity
---adds 2 uart_ticks,overhead also add 2 ticks, so it is 4.

      when SYNC =>
            if (counter_b = "0100") then        -----waiting for 4 uart_ticks-------
              if (not data_in = '1') then       -----if the data is low, then go to nextstate
                next_state<= WAIT_TICK;
              else
                next_state<= DETECT_START_BIT; --not start-bit, go detect-start bit.
              end if;
            else
              next_state<= SYNC;
              reset_counter_var := '1';      --- allow counter to tick
            end if;
-- Wait a bit-cell time before sampling the       state of the data in
      when WAIT_TICK =>
            if (counter_b = "1110") then         ----- (2) if bitcell is 16 uart_ticks,
              if (counter_r =  conv_std_logic_vector(word_len, 4)) then
```

```
                next_state<= STOP;
              reset_counter_var := '0';---------- (2)
            else
              next_state<= SAMPLE;
              count_var := '1';          ---- one more bit received
            end if;
          else
            next_state<= WAIT_TICK;
            reset_counter_var := '1';  --- allow counter to tick
          end if;

--Sample the state of the RECEIVE data pn
      when SAMPLE =>
          shift_var := '1';  ---shift in the serial data
          next_state<= WAIT_TICK;

-- make sure that we've seen the stop bit
     when STOP =>
          next_state<= DETECT_START_BIT;
          rxdone_int_var := '1';

    when others  =>
          next_state<= DETECT_START_BIT;
          reset_counter_var := '0';  ----------(2)
                shift_var := '0';
                count_var := '0';
                reset_count_var := '0';
                rxdone_int_var := '0';

   end case;

   reset_counter <= reset_counter_var;
   shift <= shift_var;
   count <= count_var;
   rstcount <= reset_count_var;
   rxdone_int <= rxdone_int_var;
  end process;

-- register the state machine outputs to eliminate ciritical-path/glithces

  process (clk, resetn)
  begin
    if (resetn='0') then
      rxdone  <= '0';
    elsif (clk'event and clk = '1') then
      rxdone <= rxdone_int;
   end if;
  end process;

end architecture syn;
```

```
-------------------------------------------------------------------------------------------------------------
-- UART Transmitter module: transmitter.vhd
--Author :Shumin Shen
--Date: March 9, 2003: updated
--VHDL UART Transmitter Entity: transmits serial data from on-chip receive buffer to the pc.
--This is the asynchronous transmitter portion of the UART.
-------------------------------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity transmitter is
  port (
    clk             : in std_logic;
    resetn          : in std_logic;
    ser_data_TX     : out std_logic;   ---output: initial is 1.
    start_TX        : in std_logic;
    par_data_TX     : in std_logic_vector(7 downto 0);
    txdone          : out std_logic);   --finish is 1.
end  transmitter;

architecture syn of transmitter is

  signal load             : std_logic;
  signal shift            : std_logic;
  signal counter_u        : std_logic_vector(4 downto 0);   ---bitcell counter
  signal count_u          : std_logic;
  signal txregister       : std_logic_vector(7 downto 0);
  signal counter_b        : std_logic_vector(3 downto 0); ---transmitted bit counter
  signal reset_counter_b  : std_logic;
  signal count_b          : std_logic;
  signal sel              : std_logic_vector(1 downto 0);

  type tx_states is (IDLE,START,WAIT_FOR_DATA,SHIFT_DATA,STOP);
  signal state            : tx_states;
  signal next_state       : tx_states;
  constant  word_len      : integer := 8;
  signal ser_data_TX_int  : std_logic;
  signal txdone_int       : std_logic;

begin
  --register the outputs to eliminate ciritical-path/glithces
  process (clk)
  begin

   if (clk'event and clk = '1') then
      ser_data_TX <= ser_data_TX_int;
   end if;
  end process;

  process (txregister, sel)
    variable ser_data_TX_var  : std_logic;
  begin
```

```
    case sel is
      when "00" =>
            ser_data_TX_var := '0';   ---transmitt start bit is 0.
      when "01" =>
            ser_data_TX_var := '1';   ---transmitt stop bit is 1.
      when "10" =>
            ser_data_TX_var := txregister(0);   ----shift
      when others  =>
            ser_data_TX_var := '1';
    end case;
    ser_data_TX_int <= ser_data_TX_var;
  end process;

 -- Bit Cell time Counter

  process (clk, resetn)
    begin
     if (resetn = '0') then
      counter_u <= "00000";
    elsif (clk'event and clk = '1') then
     if (count_u = '1') then
     counter_u <= counter_u + "00001";
      else
     counter_u <= "00000";
      end if;
    end if;
   end process;

--Shift Register, The LSB must be shifted out first

  process (clk, resetn)
  begin
    if (resetn = '0') then
      txregister <= "00000000";
    elsif (clk'event and clk = '1') then
      if (load = '1') then
        txregister <= par_data_TX;
      else
        if (shift = '1') then
          txregister(6 downto 0) <= txregister(7 downto 1);
          txregister(7) <= '1';
        else
          txregister <= txregister;
        end if;
      end if;
    end if;
  end process;


-- Transmitted bit counter

  process (clk, resetn)
  begin
    if (resetn = '0') then
      counter_b <= "0000";
```

```
    elsif (clk'event and clk = '1') then
      if (reset_counter_b = '1') then        ---------- idle
        counter_b <= "0000";
      else
        if (count_b = '1') then
          counter_b <= counter_b + "0001";
        end if;
      end if;
    end if;
  end process;

-- STATE MACHINE,State Variable
  process (clk, resetn)
  begin
    if (resetn = '0') then
      state <= IDLE;
    elsif (clk'event and clk = '1') then
      state <= next_state;
    end if;
  end process;

--- Next State, Output Decode

  process (state, start_TX, counter_u, counter_b)

    variable load_var  : std_logic;
    variable count_u_var  : std_logic;
    variable shift_var  : std_logic;
    variable reset_counter_b_var  : std_logic;
    variable count_b_var  : std_logic;
    variable sel_var  : std_logic_vector(1 downto 0);
    variable txdone_int_var  : std_logic;
  begin

    load_var := '0';
    count_u_var := '0';
    shift_var := '0';
    reset_counter_b_var := '0';
    count_b_var := '0';
    sel_var := "01";
    txdone_int_var := '0';
    case state is

---- wait for the start command
      when IDLE =>
            sel_var := "01";          -----(2)
          if (start_TX = '1') then
            next_state<= START;
            load_var := '1';
            txdone_int_var := '0';
          else
            next_state<= IDLE;
            reset_counter_b_var := '1';
            txdone_int_var := '0';
          end if;
```

```
---send start bit, 1 baud tick i swaited (16-uart_ticks)

      when START =>
            sel_var := "00";
           if (counter_u = "01110") then          ----------14 uart_ticks
             next_state<= WAIT_FOR_DATA;
           else
             next_state<= START;
             count_u_var := '1';          --allow to count up
           end if;

-- wait 1 bit-cell time before sending data on the xmit pin
      when WAIT_FOR_DATA =>
            sel_var := "10";   ---------shift register,1 bit-cell time wait completed
           if (counter_u = "01110") then
             if (counter_b = conv_std_logic_vector(word_len, 4)) then
               -------all bits have been transmitted.
               next_state<= STOP;
             else
               next_state<= SHIFT_DATA;
               count_b_var := '1';   ----------------1 more bit sent
             end if;
           else
             next_state<= WAIT_FOR_DATA; -- bit-cell wait not complete
             count_u_var := '1';
           end if;
      when SHIFT_DATA =>
            sel_var := "10";
           next_state<= WAIT_FOR_DATA;
           shift_var := '1';  --------  shift out next bit
      when STOP =>
           ----send stop bit
           sel_var := "01";
           if (counter_u = "01110") then
             next_state<= IDLE;
             txdone_int_var := '1';
           else
             txdone_int_var := '0';
             next_state<= STOP;
             count_u_var := '1';  ---  allow bit cell cntr
           end if;
      when others  =>
          next_state<=IDLE;
             load_var := '0';
                        count_u_var := '0';
                        shift_var := '0';
                        reset_counter_b_var := '0';
                        count_b_var := '0';
                        sel_var := "01";
                        txdone_int__var := '0';

    end case;

    load <= load_var;
```

```
    count_u <= count_u_var;
    shift <= shift_var;
    reset_counter_b <= reset_counter_b_var;
    count_b <= count_b_var;
    sel <= sel_var;
    txdone_int <= txdone_int_var;
  end process;

--register the state machine outputs
--to eliminate ciritical-path/glithces
  process (clk, resetn)
  begin
    if (resetn = '0') then
      txdone <= '0';
    elsif (clk'event and clk = '1') then
      txdone <= txdone_int;
    end if;
  end process;

end architecture syn;
```

```
---------------------------------------------------------------------------------------------------
--Simple Baud-clock generator for VHDL UART,  div_clk.vhd

--Author: Shumin Shen
---------------------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity div_clk is
  generic (
    -- clock frequency
    CLKX                : integer;
    BAUD                : integer
    );
  port (
    clk             : in std_logic;
    resetn          : in std_logic;
    baud_clk        : out std_logic
    );
end entity div_clk;

architecture divide of div_clk is

 constant DIVIDED              : integer:=CLKX/(BAUD*2*16);
constant CPR                   : integer:= 6; ---(9600)

--CPR>=LOG2(DIVIDED)=6.0245       --------generate appropriate baud delay
  signal clk_div        : std_logic_vector(CPR-1 downto 0);
  signal clk_baud       : std_logic;

begin
baud_clk <= clk_baud;

process (clk, resetn)

  begin
    if (resetn = '0') then
      clk_div <= (others => '0');
      clk_baud <= '0';

    elsif (clk'event and clk = '1') then
      if (clk_div = conv_std_logic_vector(CONV_UNSIGNED(DIVIDED, CPR), CPR)) then
        clk_div <= (others => '0');
        clk_baud<= not clk_baud;
      else
        clk_div<= clk_div + conv_std_logic_vector(CONV_UNSIGNED(1, CPR ), CPR);
        clk_baud <= clk_baud;
      end if;
    end if;
  end process;

end architecture divide;
```

```
-----------------------------------------------------------------------------------------------------------------
-----Dualclock.vhd:  transfer signal between two clock domains:
---end_adc_f: fast clock (10MHZ) to end_adc: uart_clock.
---Data: Aug. 13, 2003
-----Author: Shumin Shen
-----------------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity dualclock is
  port      (end_adc_f, RESETN, CLK1, CLK2 : in std_logic;
             S3 : inout std_logic);
end entity dualclock;

architecture BEHAV of dualclock is
  signal S5N : std_logic;      -- internal signals
   signal s2, s4, s5 : std_logic;
begin
  STATE_CHANGE1 : process (CLK1) is
  begin
            if CLK1'event and CLK1 = '1' then
                      S2 <= end_adc_f;
            end if;
  end process STATE_CHANGE1;

  STATE_CHANGE2 : process (S2, S5N) is
  begin
            if (S5N = '0') then
                      S3 <= '0';
            elsif S2'event and S2 = '1' then
                      S3 <= '1';
            end if;
  end process STATE_CHANGE2;

  STATE_CHANGE3 : process (CLK2, S5N) is
  begin
            if (S5N = '0') then
                      S4        <= '0';
            elsif CLK2'event and CLK2 = '1' then
                      S4 <= S3;
            end if;
  end process STATE_CHANGE3;

  STATE_CHANGE4 : process (CLK2, RESETN) is
  begin
            if (RESETN = '0') then
                      S5 <= '0';
                      S5N <= '1';
            elsif CLK2'event and CLK2 = '1' then
                      S5 <= S4;
                      S5N <= not(S4);
            end if;
  end process STATE_CHANGE4;

end architecture BEHAV;
```

```
-- megafunction wizard: %ALTCLKLOCK%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altclklock

-- ============================================================
-- File Name: clkpll.vhd
-- Megafunction Name(s):  altclklock
-- ============================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY clkpll IS
  PORT
  (
        inclock         : IN STD_LOGIC ;
        locked          : OUT STD_LOGIC ;
        clock0          : OUT STD_LOGIC ;
        clock1          : OUT STD_LOGIC
  );
END clkpll;

ARCHITECTURE SYN OF clkpll IS

  SIGNAL sub_wire0      : STD_LOGIC ;
  SIGNAL sub_wire1      : STD_LOGIC ;
  SIGNAL sub_wire2      : STD_LOGIC ;


  COMPONENT altclklock
  GENERIC (
        inclock_period          : NATURAL;
        clock0_boost            : NATURAL;
        clock1_boost            : NATURAL;
        operation_mode          : STRING;
        intended_device_family          : STRING;
        valid_lock_cycles               : NATURAL;
        invalid_lock_cycles             : NATURAL;
        valid_lock_multiplier           : NATURAL;
        invalid_lock_multiplier         : NATURAL;
        clock0_divide           : NATURAL;
        clock1_divide           : NATURAL;
        outclock_phase_shift            : NATURAL
  );
  PORT (
                inclock : IN STD_LOGIC ;
                clock0  : OUT STD_LOGIC ;
                clock1  : OUT STD_LOGIC ;
                locked  : OUT STD_LOGIC
  );
```

```
  END COMPONENT;

BEGIN
  clock0    <= sub_wire0;
  clock1    <= sub_wire1;
  locked    <= sub_wire2;

  altclklock_component : altclklock
  GENERIC MAP (
            inclock_period => 25000,
            clock0_boost => 1,
            clock1_boost => 1,
            operation_mode => "NORMAL",
            intended_device_family => "APEX20KE",
            valid_lock_cycles => 5,
            invalid_lock_cycles => 5,
            valid_lock_multiplier => 5,
            invalid_lock_multiplier => 5,
            clock0_divide => 4,
            clock1_divide => 1,
            outclock_phase_shift => 0
  )
  PORT MAP (
            inclock => inclock,
            clock0 => sub_wire0,
            clock1 => sub_wire1,
            locked => sub_wire2
  );


END SYN;
```

```
-- ================================================================
-- File Name: uart_rec_buf.vhd
-- Megafunction Name(s):
--                          lpm_ram_dp
-- ================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.lpm_components.all;

ENTITY uart_rec_buf IS
  PORT
  (
        data            : IN STD_LOGIC_VECTOR (11 DOWNTO 0);
        wraddress               : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
        rdaddress               : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
        wren            : IN STD_LOGIC  := '1';
        rden            : IN STD_LOGIC  := '1';
        wrclock         : IN STD_LOGIC ;
        rdclock         : IN STD_LOGIC ;
        q               : OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
  );
END uart_rec_buf;


ARCHITECTURE SYN OF uart_rec_buf IS

  SIGNAL sub_wire0      : STD_LOGIC_VECTOR (11 DOWNTO 0);

  COMPONENT lpm_ram_dp
  GENERIC (
        lpm_width               : NATURAL;
        lpm_widthad             : NATURAL;
        rden_used               : STRING;
        intended_device_family          : STRING;
        lpm_indata              : STRING;
        lpm_wraddress_control           : STRING;
        lpm_rdaddress_control           : STRING;
        lpm_outdata             : STRING;
        use_eab         : STRING;
        lpm_type                : STRING
  );
  PORT (
                rdclock : IN STD_LOGIC ;
                wren    : IN STD_LOGIC ;
                wrclock : IN STD_LOGIC ;
                q       : OUT STD_LOGIC_VECTOR (11 DOWNTO 0);
                rden    : IN STD_LOGIC ;
                data    : IN STD_LOGIC_VECTOR (11 DOWNTO 0);
                rdaddress       : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
                wraddress       : IN STD_LOGIC_VECTOR (6 DOWNTO 0)
  );
  END COMPONENT;
```

```
BEGIN
  q   <= sub_wire0(11 DOWNTO 0);

  lpm_ram_dp_component : lpm_ram_dp
  GENERIC MAP (
            lpm_width => 12,
            lpm_widthad => 7,
            rden_used => "TRUE",
            intended_device_family => "UNUSED",
            lpm_indata => "REGISTERED",
            lpm_wraddress_control => "REGISTERED",
            lpm_rdaddress_control => "REGISTERED",
            lpm_outdata => "UNREGISTERED",
            use_eab => "ON",
            lpm_type => "LPM_RAM_DP"
  )
  PORT MAP (
            rdclock => rdclock,
            wren => wren,
            wrclock => wrclock,
            rden => rden,
            data => data,
            rdaddress => rdaddress,
            wraddress => wraddress,
            q => sub_wire0
  );


END SYN;
```

```
-------------------------------------------------------------------------------------------------------------------
-- uart_decoder.vhd:  Entity Declaration
--date: Aug. 9, 2003
--Author: Shumin Shen
--modification : date: Aug 13, 2003 for  adc
--decorder start_adc signal as well as trasferring this signal between two clock domains
-- from uart_clock domain to clk( 10MHZ) domain.
-------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity uart_decoder is
  PORT(              clk         : in std_logic;
                     rxdone    : in std_logic;
                      i_word_in: in std_logic_vector( 7 downto 0);
                      resetn: in std_logic;
                   start_adc: out std_logic );

end uart_decoder;

architecture buffers of uart_decoder is
begin

process( clk, resetn)
begin
if(resetn='0') then
  start_adc<='0';
  elsif(rising_edge(clk)) then
              if (i_word_in="01000001") then
      start_adc<='1';
    else start_adc<='0';
     end if;

 end if;
end process;

end buffers;
```

```
-------------------------------------------------------------------------------------------------------------------
-- gain_decoder.vhd:  Entity Declaration
--date: sep.13, 2003
--Author: Shumin Shen
---update: nOV.13, 2003
-------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity  gain_decoder is
   port  ( clk   : in std_logic;  ------------10Mhz
           resetn : in std_logic;
          adc_exponent  : in std_logic_vector( 9 downto 0);
           start_adc: in std_logic;-----------------uartclcok
             e0  : out std_logic;
             e1  : out std_logic;
             e2  : out std_logic;
             A1  : out std_logic;
             A0  : out std_logic;
             eoc : OUT STD_LOGIC
       );
end gain_decoder;

architecture buffers of gain_decoder is

type tx_states is (  IDLE, gain );
signal state     :  tx_states;
signal next_state :  tx_states;

--Buffer signals
signal ad_e, ad_e_int          : std_logic_vector(9 downto 0);
signal wcounter: integer  range 0 to 256;
signal wcount_adc, wcountadc: std_logic;
SIGNAL gain1_out, gain2_out , gain4_out, gain8_out  : std_logic;
signal gain1_out_int, PGAgain1_int, gain1_int            : std_logic;
signal gain2_out_int, PGAgain2_int, gain2_int            : std_logic;
 signal gain4_out_int, PGAgain4_int, gain4_int : std_logic;
signal gain8_out_int, PGAgain8_int, gain8_int : std_logic;
signal EOC_int      : std_logic;
signal start_adc_f: std_logic;
 signal e1_int, e0_int, e2_int: std_logic;

begin

process( clk, resetn)
begin

  if(resetn='0') then
            gain1_out<='0';
            gain2_out<='0';
            gain8_out<='0';
            gain4_out<='0';
     ad_e<=(others=>'0');
```

```
            e0<='0';
            e1<='0';
            e2<='0';
    start_adc_f<='0';
    EOC<='0';
  else
            if(rising_edge(clk)) then
    ad_e<=adc_exponent;
     start_adc_f<=start_adc;
    gain1_out<=gain1_out_int;
    gain2_out<=gain2_out_int;
    gain4_out<=gain4_out_int;
    gain8_out<=gain8_out_int;
    e0<=e0_int;
    e1<=e1_int;
    e2<=e2_int;
    EOC<=EOC_int;
    end if;
  end if;
end process;

-- STATE MACHINE,State Variable
  process (clk, resetn)
  begin
    if (resetn = '0') then
      state <= IDLE;
    elsif (clk'event and clk = '1') then
      state <= next_state;
    end if;
  end process;

process (clk, resetn)
    begin
     if (resetn = '0') then
      wcounter<=0;
    elsif (clk'event and clk = '1') then
        if (wcount_adc = '1') then
           wcounter<= 0;
        else
          if (wcountadc = '1') then
             wcounter<= wcounter+1;
          end if;
        end if;
    end if;
end process;

--- Next State, Output Decode

  process (state)
variable EOC_INT_var, wcount_adc_var, wcountadc_var: std_logic;
  begin

                    EOC_INT_var:='0';
                        wcountadc_var:='0';
                    wcount_adc_var:='0';
```

```
case state is

---- wait for the start command
      when IDLE =>
   if ((start_adc_f='0' and start_adc='1')) then

PGAgain1_int<='0';
PGAgain2_int<='0';
PGAgain4_int<='0';
PGAgain8_int<='0';
     next_state<=gain;
   else
      next_state<= IDLE;
     end if;
 ----------the current gain is

when gain=>
 if (wcounter<256)then
       ad_e_int<=ad_e; ----------current gain

       wcount_adc_var:='0';
            wcountadc_var:='1';

   gain1_int<=((not ad_e_int(9))and ad_e_int(8))or ( ad_e_int(9) and (not ad_e_int(8)));
   gain2_int<=((not ad_e_int(9))and (not ad_e_int(8))and ad_e_int(7))
          or (ad_e_int(9) and ad_e_int(8)and ( not ad_e_int(7)));
   gain4_int<=((not ad_e_int(9))and (not ad_e_int(8))and (not ad_e_int(7))and ad_e_int(6)) or
                  ( ad_e_int(9) and ad_e_int(8)and ad_e_int(7)and (not ad_e_int(6)));
   gain8_int<=((not ad_e_int(9))and (not ad_e_int(8))and (not ad_e_int(7))and (not ad_e_int(6)) and
ad_e_int(5))  or   ( ad_e_int(9) and ad_e_int(8)and ad_e_int(7)and ad_e_int(6)and ( not ad_e_int(5)));

       if ( gain1_int=PGAgain1_int and gain2_int=PGAgain2_int and gain4_int=PGAgain4_int  and
gain8_int=PGAgain8_int) then

                  EOC_INT_var:='1';
              gain1_out_int<=gain1_int;
              gain2_out_int<=gain2_int;
              gain4_out_int<=gain4_int;
              gain8_out_int<=gain8_int;
              e0_int<=gain4_int or gain1_int;
              e1_int<=gain1_int or gain2_int;
              e2_int<='1';

         else

          PGAgain1_int<=gain1_int;
           PGAgain2_int<=gain2_int;
           PGAgain4_int<=gain4_int;
           PGAgain8_int<=gain8_int;
           EOC_INT_var:='0';
           e0_int<='0';
           e1_int<='0';
           e2_int<='0';
```

```
            end if;

             next_state<=gain ;

else

          wcount_adc_var:='1';
         wcountadc_var:='0';
                    next_state<=idle;

             gain1_out_int<='0';
            gain2_out_int<='0';
            gain4_out_int<='0';
            gain8_out_int<='0';
            e0_int<='0';
             e1_int<='0';
             e2_int<='0';
        EOC_INT_var:='0';

end if;

 when others=>
             next_state<=idle;
             gain1_out_int<='0';
             gain2_out_int<='0';
             gain4_out_int<='0';
             gain8_out_int<='0';
                    e0_int<='0';
                    e1_int<='0';
                    e2_int<='0';
             EOC_INT_var:='0';
             wcountadc_var:='0';
            wcount_adc_var:='0';
    end case;
          wcountadc<=wcountadc_var;
          wcount_adc<=wcount_adc_var;
          EOC_int<=EOC_INT_var;

  end process;

process( clk, resetn)
begin
if(resetn='0') then
  A0<='0';
  A1<='0';
elsif(rising_edge(clk)) then
            if (gain1_out='1') then
                A0<='0';
                A1<='0';
             elsif (gain2_out='1') then
                A0<='1';
                A1<='0';
             elsif (gain4_out='1') then
                A0<='0';
```

```
                A1<='1';
            elsif (gain8_out='1') then
                A0<='1';
                A1<='1';
        else
            A0<='0';
            A1<='0';

        end if;


end if;

end process;


end buffers;
```

```
-------------------------------------------------------------------------------------------------------------------
-- uart_interface_adc.vhd: Entity Declaration
--date: Aug., 7, 2003, version 1
--Author: Shumin Shen
--modification :version 3
-- date: sept.22, 2003 for floating-point ADC
-------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity uart_interface_adc is
  PORT(          clk              : in std_logic;
                 clk_u            : in std_logic;
                rxdone   : in std_logic;
                txdone   : in std_logic;
                start_TX : out std_logic;
               start_adc  : in std_logic;
                 EOC      : in   std_logic;
               end_adc_f: out std_logic;
                 end_adc: in std_logic;
                      e0: in std_logic;
                     e1: in std_logic;
                      e2: in std_logic;
      ADc_mantissa: IN STD_LOGIC_VECTOR(9 DOWNTO 0);
                   resetn: in std_logic;
     par_data_TX   : out std_logic_vector(7 downto 0)
          );
end uart_interface_adc ;

architecture buffers of uart_interface_adc is

component uart_rec_buf
  PORT
  (
          data            : in std_logic_vector (7 downto 0);
          wraddress               : in std_logic_vector ( 8 downto 0);
          rdaddress               : in std_logic_vector (8 downto 0);
          wren            : in std_logic;
          wrclock         : in std_logic ;
          rdclock         : in std_logic ;
          rden            : in std_logic ;
          q               : out std_logic_vector ( 7 downto 0)
  );
end component;

constant value:integer:=1;
constant width:integer:= 8;
constant wide:integer:= 7;

  type tx_states is (IDLE,WRITE );
  signal state     :  tx_states;
  signal next_state :  tx_states;
```

```
--Buffer signals
signal data_rx_in, data_rx_in_int      : std_logic_vector(wide downto 0);
signal wraddress_rx            : std_logic_vector(8 downto 0);
signal wren_rx, wren_rx_int            : std_logic;
signal data_tx_out  : std_logic_vector(wide downto 0);
signal rdaddress_tx : std_logic_vector(8 downto 0);
signal rden_tx                 : std_logic;
signal endadc_f                : std_logic;
signal start_adc_n: std_logic;
signal   wcount_adc , wcountadc: std_logic;
signal wcounter, twcounter: integer  range 0 to 256;
signal scounter, icounter: integer range 0 to 256;
signal     countwrite_add,  count_write_add : std_logic;

begin
uart_rec_buf_inst : uart_rec_buf
PORT MAP(
            data            => data_rx_in,
            wraddress       => wraddress_rx,
            rdaddress       => rdaddress_tx,
            wren            => wren_rx,
            wrclock         => clk,
            rdclock         => clk_u,
            rden            => rden_tx,
            q               => data_tx_out
  );

process (clk, resetn)
    begin
      if (resetn = '0') then
       wcounter<=0;
    elsif (clk'event and clk = '1') then
         if (wcount_adc = '1') then
            wcounter<= 0;
        else
          if (wcountadc = '1') then
             wcounter<= wcounter+1;
          end if;
        end if;
    end if;
end process;

process (clk, resetn)
    begin
      if (resetn = '0') then
            wraddress_rx <= ( others=>'1');
    elsif (clk'event and clk = '1') then
      if (count_write_add = '1') then
            wraddress_rx<= ( others=>'1');
      else
        if (countwrite_add = '1') then
           wraddress_rx<=wraddress_rx+conv_std_logic_vector(1,width);
         end if;
```

```
            end if;
          end if;
        end process;

        -- STATE MACHINE,State Variable
          process (clk, resetn)
          begin
            if (resetn = '0') then
              state <= IDLE;
            elsif (clk'event and clk = '1') then
              state <= next_state;
            end if;
          end process;

        --- Next State, Output Decode
        process (state,  wcounter, twcounter)
        variable wren_rx_int_var, wcount_adc_var,wcountadc_var, end_adc_f_var,  countwrite_add_var,
count_write_add_var: std_logic;

        begin
                    count_write_add_var:='0';
                    countwrite_add_var:='0';
                     wren_rx_int_var:='0';
                      wcount_adc_var:='0';
                        end_adc_f_var:='0';
                        wcountadc_var:='0';

         case state is
        ---- wait for the start command
              when IDLE =>
             if (start_adc_n='0' and start_adc='1') then
              next_state<= write;
               else
                      next_state<= IDLE;
              end if;
             when write =>
        if (EOC='1') then
                 case  wcounter is
                 when 254=>
                  wren_rx_int_var:='0'; ----------------------
                          end_adc_f_var:='1';
                          wcount_adc_var:='1';  ---reset counter
                          wcountadc_var:='0';
                  count_write_add_var:='1'; ----------reset wraddress
                  countwrite_add_var:='0';
                   data_rx_in_int<=(others=>'0');
                  next_state<=idle;
                        when  0 to 253 =>
                          end_adc_f_var:='0';
                      data_rx_in_int<=( ADc_mantissa( 9 downto 2) &  e1 & e0 ); ----------from mantissa
                  countwrite_add_var:='1';  ---address+1
                            wren_rx_int_var:='1';
                            count_write_add_var:='0';
                            wcount_adc_var:='0';
                            wcountadc_var:='1';
```

```
                    next_state<=write ;
          when others=>
            countwrite_add_var:='0';
            wren_rx_int_var:='0';
                      end_adc_f_var:='0';
                   wcount_adc_var:='0';
                wcountadc_var:='0';
            count_write_add_var:='0';
            data_rx_in_int<=(others=>'0');
            next_state<=idle;----------------
          end case;
else

       countwrite_add_var:='0';
           wren_rx_int_var:='0';
                      end_adc_f_var:='0';
                   wcount_adc_var:='0';
                wcountadc_var:='0';
            count_write_add_var:='0';
            next_state<=write;
end if;
when others=>
              next_state<=idle;
              count_write_add_var:='0';
              data_rx_in_int<=(others=>'0');
             wren_rx_int_var:='0';
              wcount_adc_var:='0';
             wcountadc_var:='0';
                 end_adc_f_var:='0';
              countwrite_add_var:='0';
    end case;
      wren_rx_int<=wren_rx_int_var;
      wcount_adc<=wcount_adc_var;
            wcountadc<= wcountadc_var;
      count_write_add<= count_write_add_var;
      countwrite_add<= countwrite_add_var ;
      end_adc_f<=end_adc_f_var;
  end process;

  write_received : process(clk,resetn,wcounter)
begin
  if(resetn='0') then

       data_rx_in<=(others=>'0');
       wren_rx<='0';
   start_adc_n<='0';

  else
             if(rising_edge(clk)) then
               wren_rx<=wren_rx_int;
             data_rx_in<=data_rx_in_int;
             start_adc_n<=start_adc;

      end if;
  end if;
```

```
end process write_received;

read_tx_data : process(clk_u,resetn,txdone,data_tx_out,rdaddress_tx,end_adc)
begin
  if(resetn='0') then
            rdaddress_tx<=(others=>'1');
            par_data_TX<=(others=>'1');
            start_TX<='0';
            rden_tx<='0';
            twcounter<=0;
            scounter<=0;
  else
            if(rising_edge(clk_u)) then
                 if( end_adc ='1' ) then
                                start_TX<='1'; rdaddress_tx<=( others=>'1');
           else IF (  twcounter=256)    THEN
                  rden_tx<='0';
                         start_TX<='0';
                         else
                            if(txdone='1' ) then
                         if  ((scounter mod 2)=0) then

                         rden_tx<='1';
                         rdaddress_tx<=rdaddress_tx;
                         else
                         par_data_TX<=("0000"& data_tx_out( 1 downto 0) );
                         scounter<=scounter+1; start_TX<='1'; rden_tx<='1';
                           rdaddress_tx<=rdaddress_tx+conv_std_logic_vector(1,width);
                           twcounter<=twcounter+1;
                       end if;
                    else
                         rden_tx<='0';
            start_TX<='0';
                end if;
             end if;
          end if;
     end if;
  end if;
end process read_tx_data;
end buffers;
```