I.J. Information Engineering and Electronic Business, 2010, 1, 1-8
Published Online November 2010 in MECS (http://www.mecs-press.org/)

A Novel Fixed-Point Simulation Library for the Design of FPGAs

Bingyang Liu, Xiaojun Zhang, Zhenchao Chang, Jing Liu
National Digital Switch System Engineering & Technological R & D Center
450002 ZhengZhou, HeNan, China
E-Mail: liubingyang1985@126.com

***Abstract*—Fixed-point simulation is extremely important in the design of fixed-point FPGAs. Float-point simulation is used to verify the arithmetic, while the fixed-point simulation is adopted to evaluate the performance and verify the implementation. The design of high-precision system based on FPGAs must focus on the fixed-point simulation to reduce the error acceptable even to zero. Basing on the analysis of fixed-point simulation approaches, especially the approach with MATLAB Fixed-Point Toolbox, we propose a novel fixed-point simulation library consisting of four modules. The library works under VC environment and its basic definition module imitates operating principle of MATLAB Fixed-Point Toolbox. The aim of library is to assit setting up fixed-point simulation conveniently, easily and quickly. Finally, simulation shows effect of the library.**

Index Terms -fixed-point simulation, library, zero error, float-point to fixed-point conversion

I. INTRODUCTION

Concerned with the cost of area and speed, fixed-point arithmetic is generally adopted during the design of FPGAs (Field Programmable Gate Array). The process of FPGA fixed-point design is approximately divided into three proportions as shown in Fig.1, while establishing the fixed-point model is extremely important and highly difficult. In recent years, some software programs have appeared which can automatically generate Verilog/VHDL(Hardware Description Language) codes. All of these software programs can generate target codes according to fixed-point simulation codes, which shows the importance of fixed-point simulation. Reference [1] shows that in a recent survey conducted by AccelChip Inc., 53 percent of the respondents identified float-point to fixed-point conversion as the most difficult aspect of implementing an algorithm on an FPGA. The main two problems of fixed-point simulation are errors and cycles, each of which would be analyzed below.

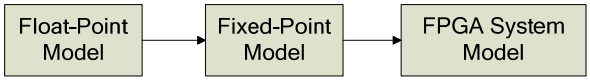


Figure 1. The process of FPGA fixed-point design

A. *Error problem*

Reference [2~5] shows that in recent years, there are many methods to research compiling fixed-point simulation codes, most of which just analyze and research the error between float-point output and fixed-point output, while missing the error between fixed-point output and FPGA output. At present, most systems today consist of certain components, thus if an error appears between any component of a fixed-point simulation model and a FPGA model, accumulated errors of entire system, especially high-precision demand system (such as area of radar, navigation, speech recognization) can not be ignored, because great errors make the simulation nonsensical to simulate [6].

So, it is important to establish fixed-point simulation model with acceptable errors, even zero error.

Causes for errors between fixed-point output and FPGA output list below:

1) Different scaling or bit-width. Different bit-width/scaling may process different accuracy.
2) Different operation rules. In addition and subtraction operations, whether type cast input values to the result type before operating on the values or not should be concerned. So as the multiplication operations.
3) Different implement approach. For example, the division implementation uses CORDIC in FPGA code, on the contrary, the fixed-point simulation just magnifies dividend. Thus, there must be some errors.

B. *Cycle problem*

As is known， IT(Information Technology) industry always pursues speed and short development cycle. Fixed-Point simulation often takes long time, sometimes even longer than compiling Verilog code. To ease FPGA development, manufacturers provide library which includes abundant IP(intellectual property) cores. However, no IP core function library is provided to help set up fixed-point simulation model. Programmers have to compile the code according to IP core, which usually takes long time and directly leads to low productivity.

This paper is organized as follows. Section Ⅱ mainly introduces background of fixed-point simulation, describes existent fixed-point simulation methods and analyzes their merits and demerits. Section Ⅲ mainly describes fixed-point simulation approaches with MATLAB Fixed-Point Toolbox which is the same as the basic rules of the library proposed in this paper. Section Ⅳ mainly describes the library's specification and section Ⅴ details the components of the library. Section Ⅵ provides simulation between library, MATLAB and VC, then an analysis of the simulation results is list. Finally, section Ⅶ draws conclusions and offers hints for further studies.

Ⅱ. BACKGROUND

There are many fixed-point simulation methods. Based on the generation form, it can be separated into two categories: manual compiling and code generation tools.

A. Manual Compiling

As one of the popular programming languages, the C language is widely applied in fixed-point simulation. However, it has many disadvantages:

1) Integer data type bit-width is 8,16,32,64, while data bit-width of FPGA is variable. Therefore, C language can't reflect overflow or precision loss of FPGA code.

2) C language is incapable of expressing data more than 64bits. If the data exceeds the range, the only solution is to define structural body. Therefore, it is greatly inconvenient.

3) There is no genuine fixed-point data type in C language. All data expressed by C language is magnified, and the scaling must be annotated. Therefore, it is difficult to comprehend the code.

4) Lack of fixed-point operation rules. It is inconvenient that different scaling operands must be shifted to the same scaling before addition and subtraction. Similarly, both operands must be forced to change data type before multiplication.

Recently, there are several fixed-point simulation methods such as Catapult C of Mentor and Impulse C of Impulse which define extended set of C to compile fixed-point simulation, but because of function shortage, they are not widely applied.

B. Code generation tools

In reference [7], AccelDSP Synthesis Tool includes automated floating-point to fixed-point conversion, where the floating-point MATLAB model is analyzed during simulation to determine the dynamic range of the input data and constants.

However, the disadvantages are obvious. Firstly, it lacks in transplantation because fixed-point simulation codes generated must work based on the function library of AccelDSP; Secondly, the code generated is difficult to comprehend, and it is impossible to amend and fix. Thirdly, only a few arithmetic conversions are supported. So do other similar tools.

Ⅲ. MATLAB FIXED-POINT TOOLBOX

MATLAB Fixed-Point Toolbox offers plenty of fixed-point functions and operation rules and so on. It makes fixed-point simulation more easy and free.

Firstly, we can define bit-width freely as FPGA data definition with Fi function.

Secondly, operation rules can be easily defined with Fimath attribution. By setting a series of parameters, the code shows the same mechanism as FPGA.

Thirdly, it is convenient to change data type such as the conversation between binary, octonary, decimal and hexadecimal. Therefore, data observation, record and analysis will be easy.

Last, MATLAB itself is a powerful comprehensive tool to process and analyze errors.

A. MATLAB Fixed-Point Toolbox introduction

Numerictype and Fimath parameters which are known as setting of Fi function are usually used in MATLAB fixed-point simulation process.

Fi function is used to construct a fixed-point numeric object. You can use it as the way of a = $fi(v,s,w,f)$, it returns a fixed-point data with value v, signed property value s, word length w, and fraction length f. For example, $fi(pi,1,32,28)$ constructs a fixed-point data scaling S3.28 of π. Repeating setting is avoided because of the transmissibility of Numerictype.

Fimath is used to define operation rules and rounding rules. Option setting comes as follows [8]:

- CastBeforeSum: to enable or disable type casting of input values for addition and subtraction operation. "1" for enable and "0" for disable.
- OverflowMode：an overflow-handing mode. The default is "saturate". "Saturate" is used for saturatation to the maximum or minimum value of the fixed-point range on overflow, while "wrap" means wrap on overflow. This mode is also known as overflow of two's complement.
- RoundMode: Rounding option. The default is "nearest", which means round toward nearest. Ties round toward positive infinity. "Ceil" means round toward positive infinity, while "fix" means Round toward zero. "Floor" means round toward negative infinity.

B. General steps of MATLAB fixed-point simulation and method

General steps of MATLAB fixed-point simulation:

1) Establish a float-point simulation model;

2) Range analysis of each step and variable;

3) Compile fixed-point simulation codes and establish a fixed-point simulation model;

4) Compare errors between fixed-point and float-point output, then adjust and confirm the word-width of variable.

References [3~5,9,10] concentrate on step 2 and 4 to research on how to analyze range and precision faster and better. However, Step 3 is also important because errors will be generated when mistakes appear in the fixed-point

conversion process, even though the scaling is totally right. The key is operation rules and overflow handling. The details show as following:

1) In the option of CastBeforeSum setting, "0" behaves as ASIC or FPGA implementations, while "1" imitates DSP chips.

2) RoundMode option should be set to "floor" to model FPGA or ASIC(Application Specific Integrated Circuit).

3) According to FPGA code, set ProducMode/SumMode option to "SpecifyPrecision" to specify scaling of output toensure precision.

4) Multiplication-accumulator unit is widely used in DSP area and others. Overflow maybe occurs during accumulation, and the precision is difficult to handle, making the fixed-point simulation difficult to realize. To solve the problem, it is necessary to analyze the range and then confirm the scaling of output. The result will be the same as FPGA output when setting OverflowMode to "wrap".

5) Avoid complex arithmetic implementation such as division and other operations by predigestion and conversation.

With options and methods above, the absolute error between fixed-point simulation and FPGA will be zero.

Ⅳ. A NOVEL FIXED-POINT SIMULATION LIBRARY FOR THE DESIGN OF FPGAS

However, MATLAB Fixed-Point program runs very slow in speed and low in efficiency. For example, ten frames of MATLAB fixed-point MFCC computation time is amazingly around one hour while VC fixed-point time is only thirteen seconds. So, most researchers tend to use C language to realize the fixed-point simulation. What is more, there is no integrated IP core function to be frequently used in MATLAB environment, which increases the difficulty of fixed-point simulation and makes the precision uncertain. As a result, the design period of FPGA would be extended.

The initial idea of building up this library is to combine speediness of C/C++ and convenience of MATLAB. The main target of the library proposed is to simplify the fixed-point modeling flow via integrating the definition of fixed-point, definition of algorithm, determination of dynamic range, optimized methods of bit-width and function of common IP core. It aims to shorten modeling period and increase modeling precision, which is of great significance in IC industry which has a highly requirement of timeliness. More frankly speaking, the idea of the library is to solve the time-consuming and error-prone problem in fixed-point simulation during hardware design by combining merits of C/C++ and MATLAB, plus optimal analysis methods and a common IP core module. It can provide some options according to different requirements on implement methods, precision and bit-width.

The library can be divided into four functional modules, as shown in Fig.2. The basic definition library module provides basic definition of fixed-point and operation rules. The dynamic range analysis module is mainly used to determine

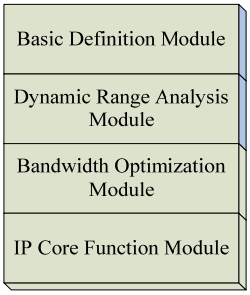


Figure 2. Parts of Library

the dynamic range of variables. Basing on the dynamic range analysis module, we can define the initial bit-width of variables. The initial fixed-point model will be established with combining the IP core functions. During the process of design, bit-width needs to be optimized according to the error and precision demand. Because the library can be applied to different phases of FPGA design, instruction of its application during different phases can be specified as follows:

A. Prime Phase of FPGA Design

This phase covers the whole design flow based on application arithmetic of FPGA implementation. Work can be done as follows: Firstly, setting up floating-point model and verifying the arithmetic. Secondly, dynamic range of variables is analyzed by using the dynamic range analysis library, and the initial scaling is defined. Thirdly, the corresponding function library in the IP core is transferred to replace the model's IP core with functions in the library, and an initial fixed-point simulation model is established. Fourthly, error between floating-point and fixed-point model is calculated. While error satisfies the user's requirements, fixed-point model is established completely. Then, the corresponding IP core need to be chosen according to the established fixed-point model, and so does the scaling. While the error dissatisfies the user's requirements, scaling has to be adjusted in the optimizing bit-width model. At the same time, other arithmetic for implementation in the IP core function library is selected. And then, the error between the adjusted fixed-point model and the original floating-point model need to be calculated. These steps need be repeated until the error meets user's requirements. The flow chart of this process is shown in Fig.3.

B. Verification Phase of FPGA Design

During this phase, FPGA program is already compiled and fixed-point model needs verifying. Firstly, we should scale the variable according to the FPGA program. Secondly, the implementation of the arithmetic needs to be fulfilled and the fixed-point model needs to be established by replacing the IP core used in the program with corresponding functions in the relevant IP core library. Then, the results of the new model should be compared with that of the original program to see whether they are the same. If not, the underlying overflow and timing problems should be found by

returning to the before-mentioned steps. Verification is completed by numbers of iterations. The flow chart of this process is shown in Fig.4.

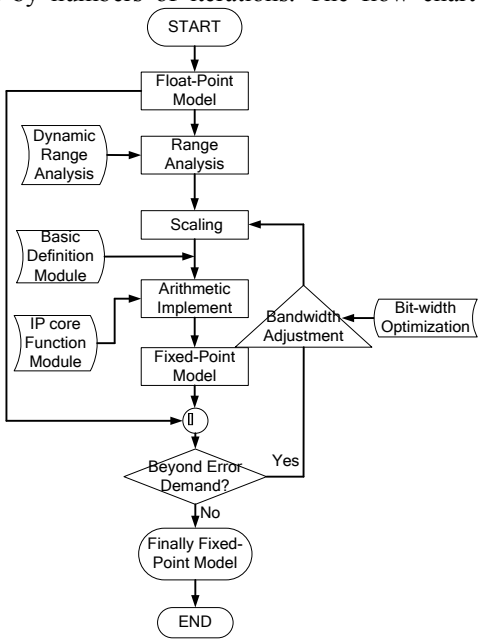


Figure 3. Prime Phase of FPGA Design

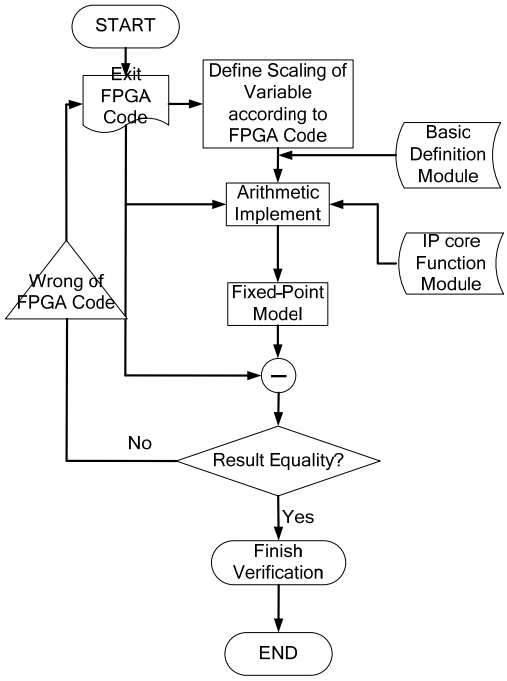


Figure 4. Verification Phase of FPGA Design

Ⅴ. COMPONENT OF LIBRARY

The library can be available when putting header file into the project.

A. Basic Definition Module

During this phase, according to the corresponding floating-point value and the definition of scaling and sign bit, a fixed-point number is confirmed, which is stored in the computer in the form of a series of sequential binary digits as shown in (1).

$$FP < 8,4,1 > a = 3.1415926 \quad (1)$$

It represents that fixed-point variable "a" is signed, its scaling is Q8.4，and the value is 3.1415926. In fact, it's a struct body filled with some parameters.

The algorithm in this module imitates MATLAB Tool Box and is defined by fully combining the FPGA algorithm. Now, it is only available for FPGA type, because its truncation is floor and its overflow handling is wrap. For example, operands would not be converted into the output type before manipulation. In the inner operation, their bits are shifted to achieve the same scaling, an intermediate variable is needed and then truncated to output. But from outside, only a simple "+" operation is needed to be executed.

As is shown in Fig.5(a), an addition between two operands "a" and "b", the corresponding scaling of signed numbers "a" and "b" are S3.4 and S2.2. The steps of addition can be partitioned as follows.

Firstly, the scaling of "b" needs to be changed into S2.4 by shifting its significant bits. Secondly, "a" is added with "b" to get an signed number with a scaling of S4.4,Thirdly，the result of the second step is given to "c". Considering the scaling of "c" is Q7.2, the trail two bits of "c" is truncated.

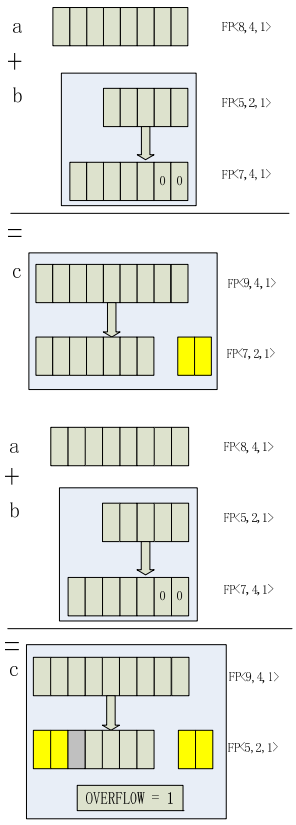
During the process of operation, the operating principle of FPGA is fully considering about overflow mechanism. When the result is overflowed, the final output is attained by intercepting corresponding bits of the result directly. As shown in Fig.5(b), the scaling of "c" is S2.2, but the result is overflowed. According to the principle, the middle five bits are directly intercepted as the final output. The first bit of "c" is recognized as the sign bit, and the overflow marker is set to 1.

B. Dynamic Range Analysis Module

Dynamic range analysis module plays an important role in the scaling of variables. Analyzing the dynamic range precisely can avoid overflow, which is of great help to the latter verification work. This model is used to analyze the dynamic range of variables. Three approaches are provided as follows:

1) Simulation approach

This method can be applied under the condition that the dynamic range of the input variables is already known.



(a) No overflow occurs (b) Overflow occurs

Figure 5. Addition of two operators

Initial value is regarded as the minimum value input, and step is set to the incremental change according to the practical condition. Increasing operation will not stop until the maximal value of input is reached, then the test sequence is formed. The dynamic range of the variable and output can be found from the input of the test sequence. In this process, while the step is made shorter, the precision of dynamic range will be higher, but the test speed will be much slower.

2) Analytical approach

It defines scaling of input variables according to their range, and then defines scaling of other variables based on the scaling of input variables and operation rules of analytical approach. This approach is suitable for plenty of area and precision sensitivity. Operation rules of analytical approach are shown in Fig.6.

The fixed-point number is consist of two parts, IB(Interger Bit-width) and FB(Fraction Bit-width). The basic rules of analytical approach are divided into two parts. When addition between two inputs with QIB1+FB1.FB1 and QIB2+FB2.FB2 respectively occurs, the scaling of output is $Q\{\max\{IB1, IB2\}+1+\max\{FB1, FB2\}\}.\max\{FB1, FB2\}$, while the scaling of multiplication output is $Q\{IB1+IB2+FB1+FB2\}.\{FB1+FB2\}$.

3) Extreme value theory approach

The approach results from extreme value theory introduced in reference [11]. Extreme value theory is used to define the range of a signal. The theory extract M groups with N samples in each group. And then the max N and min N samples were extracted to calculate the mean and standard deviation. According to extreme value theory, maximum and minimum are converged to Gumbel distribution. Maximum is got by (2).

$$x_{\max} = \mu - \sigma \ln(\ln(\frac{1}{P_r})) \quad (2)$$

Where $x_{\max}$ is maximum, P_r is percentage against overflow which is defined by the user. Generally, it is 99.9999999% while the possibility of overflow is 0.0000001%. μ and σ represent mean and variance of Gumbel distribution. We can get them with (3):

$$\sigma = \frac{s\sqrt{6}}{\pi}, \mu = \overline{X} - \sigma\lambda \quad (3)$$

Where s and $\overline{X}$ represents the standard deviation and sample mean respectively. λ is the Euler's constant (0.5772). Analogically, the minimum is shown in (4).

$$x_{\min} = \mu - \sigma \ln(\ln(\frac{1}{1-P_r})) \quad (4)$$

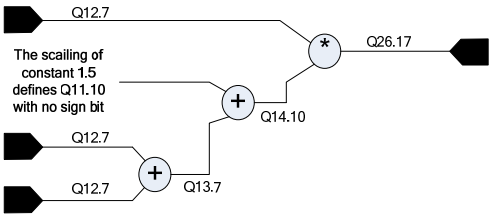


Figure 6. Operation rules of analytical approach

Then the range of signal is between $[x_{\min}, x_{\max}]$, the IB of it is generated accordingly. It shows as (5).

$$IB_{signal} = \left[\log_2(\max\{|x_{\max}|, |x_{\min}|\})\right] + 1 \quad (5)$$

Extreme value theory approach is suitable for area sensitivity and short development cycle user. It is default approach option.

C. Width optimum module

The module mainly resolves optimum of FB. Error is generally decided by FB while no overflow occurs. Theoretically, the more FB, the higher precision with less error. However, area of resource and computation is limited, while the relationship of error and FB is nonlinear. The aim of module is to decrease FB and minish the area of resource and computation while regarding demand of precision.

At present, only a coarse width optimum function *widthopti* is available. If the output is beyond the demand of error, it just increases intermediate variable with one bit once. And it adjusts intermediate variable to decrease bits according to result of range analysis module until the output is below the demand.

D. IP core function module

The most difficult step of fixed-point simulation is how to describe the scheme to correspond with hardware implementing. In the design of FPGA, manufacturer provides users with abundant IP core to shorten design cycle and decrease difficulty of development. However, there is no IP core function library during the design of fixed-point simulation. The user needs to write the code themselves, and usually it is the most time-consuming part. This part increases the difficulty of simulation and affects the schedule of FPGA design. Furthermore, if it is not in accordance with IP core, great inconvenience will occur during test and validation. For example, cordic scheme IP core is chosen to calculate division in the design of FPGAs while the user just expands the width of dividend to deal with division. Obviously, error will certainly occur and break the essence of fixed-point simulation.

However, with the module the paper proposed in this paper, users just need to choose the function according to design of FPGAs. It is very convenient to decrease the possibility of error between fixed-point simulation and FPGA implement.

At present, a few simple common IP core functions such as trigonometric function, logarithm and exponent are available. Later on, more complex IP core functions will be added into the module.

VI. Example of Application

This section introduces two examples to compare with different methods including C, MATLAB and our library. Then, this section analyzes the result and summarizes the conclusion.

A. A look-up table method on logarithm

1) Arithmetic description

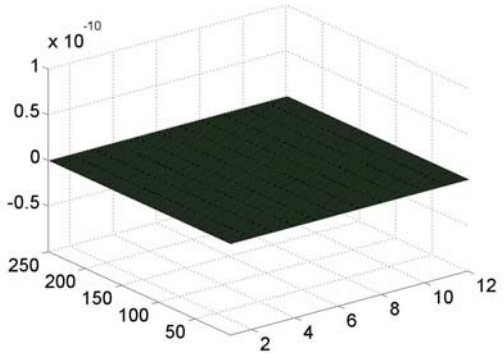
FPGA code has been compiled. The linear fitting method is introduced in reference [12] to implement the Logarithmic 2 module. It divides (1,2] span into 128 segments, while each segment corresponds to a slope and a bias. Computation begins with shifting input to (1,2] span, then the section is chosen correspondingly to get the result. The implementation method consists of multiplication, shift operations, and accumulation, which is highly representative.

Scaling of input is S13.19, while scaling of output is S5.26 by range analysis. Slopes and bias are generated by function *polyfit* of MATLAB and stored into LUT(Look Up Table) with the scaling of S0.31 and S1.31. Comparison among MATLAB, VC and our library are introduced below.

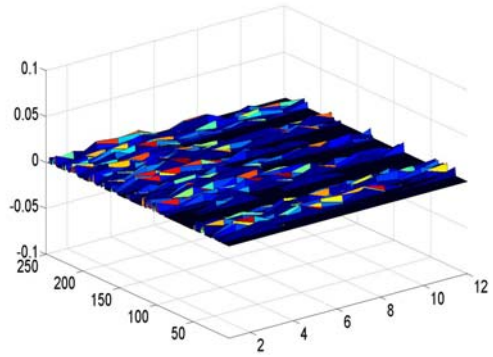
2) Result analysis

With a 12 by 250 matrix of FPGA output as input code above, the result is generated. Analysis of the error result with FPGA result is shown in Fig.7.

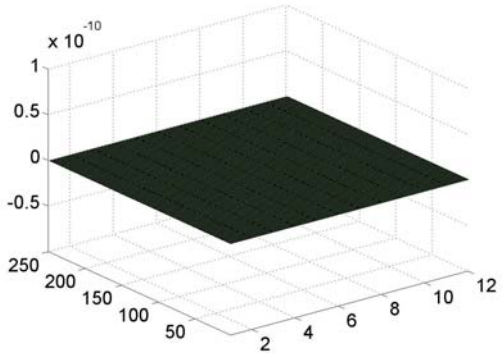
As shown in Fig.7 (a), the absolute error between MATLAB fixed-point simulation and FPGA is zero. On the contrary, errors of VC fixed-point simulation with the same word-width and implement approach can not be ignored as fig.7 (b). Causes of errors generated are operation rules and rounding rules, and they are not the same as FPGA. Though it is faster in C language, errors are not affordable. Therefore, the approach proposed above with MATLAB Fixed-Point Toolbox is deserved and is greatly helpful to the design of FPGAs.



(a) MATLAB fixed-point simulation error with FPGA



(b) VC fixed-point simulation error with FPGA



(c) Error between Library simulation and FPGA

Figure 7. Error of LOG2 module

TABLE I.
SUM OF VARIABLE IN A SINGLE LOGARITHM COMPUTATION

Approach	Sum of Variable
Library	364bits
MATLAB	374bits

TABLE II. RUNNING TIME OF EACH APPROACH

Method	Running time of 250*24
MATLAB	73 seconds
VC	less than 2 seconds
Library	16 seconds

The error demand between FPGA and float-point is 10^{-3}.With the error demand, error of library simulation result is also zero error, as shown in Fig.7 (c). But the sum of variable width of the library and MATLAB is shown in Table 1.

Cause of difference between two methods is the scaling of the second variable when extending the bit-width from 32 to 43. With the range analysis module of library, the bit-width of variable is set to 41 bits. With the analysis above, FPGA code can be modified by changing bit-width of the register. At last, area is saved with assistance of the library. What's more, running time is also considered and listed in Table.2.

From Table 2, we know that VC runs fastest. However, it introduces error with FPGA, while MATLAB takes a long time to simulate. Our library is a better choice in that it takes a shorter simulation time than MATLAB and realizes zero error with FPGA.

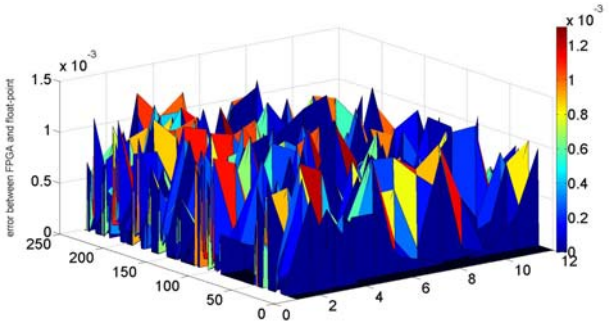


Figure 8. Error of LOG2 module

TABLE III.
SUM OF VARIABLE IN A SINGLE LOGARITHM COMPUTATION OF LIBRARY

Error demand	Sum of Variable
10^{-4}	396bits
10^{-3}	364bits

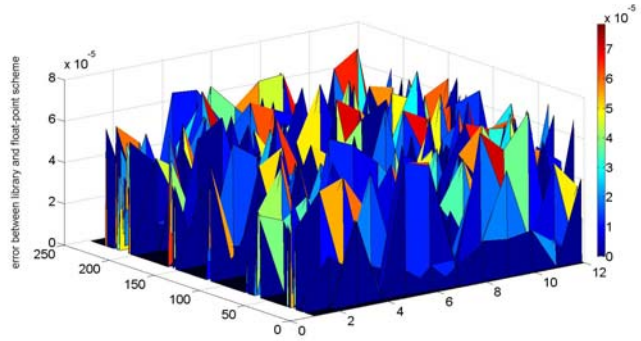


Figure 9. Error of LOG2 module

Fig.8. shows error between fixed-point and float-point simulation, it's at 10^{-3} level. While error requirement is 10^{-4}, library approach adjusts bit-width of intermediate variable and output. The sum of variable in a single logarithm computation is shown in Table 3.The error is shown as Fig.9.

B. *IP core of square root*

This example adopts square root based on IP core of cordic and shows the process and performance of library and MATLAB.

Reference [13] introduces the principle of square root. Square root implement adopts IP core of cordic which works on vector rotation model. Vector rotation takes a vector (x, y) and rotates it over an angle ρ to a new position (x', y'), while maintaining the magnitude, as shown in Fig.10. Equation (6) shows how to calculate square root by cordic:

$$\begin{aligned} x_n &= A_n\sqrt{x_0^2 - y_0^2} \\ y_n &= 0 \\ z_n &= z_0 + \tanh^{-1}\frac{y_0}{x_0} \\ A_n &= \prod_n \sqrt{1-2^{-2i}} \approx .8282 \end{aligned} \tag{4}$$

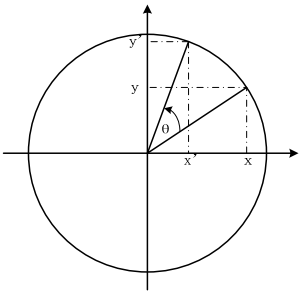


Figure 10. Vector rotation

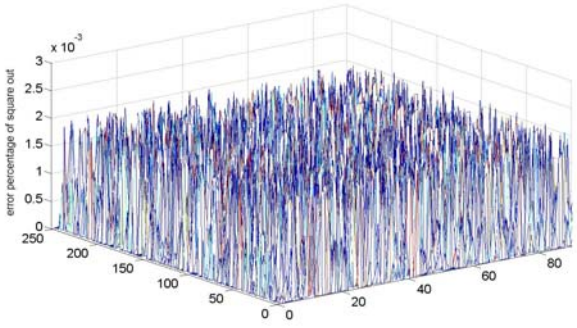


Figure 11. Error percentage of square out module

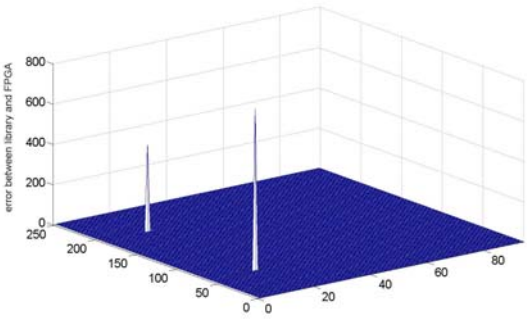


Figure 12. Error between FPGA and our library

A 94 by 250 matrix of FPGA output is the input matrix. Scaling of input is S64.32, while output is S15.17. With the library, we just call function of *cordic_squareroot* with the parameter of input. However, we need to compile it in MATLAB according to IP core specification. Because our input has a large dynamic range, we introduce error percentage between cordic and float-point shown in Fig.11.

Theoretically, the error of our library and FPGA IP core should be zero error. According to our simulation, the error of our library and FPGA IP core is shown in Fig.12.

From the Fig.12, there are two errors in 94 by 250 output while others are zero error. With analyzing, we find that if the input is larger than 2^31(2147483648), it's been recognized as a sign number and lead to mistake. Therefore, our library needs to improve.

Ⅶ. Conclusion and Future Work

Resulting in zero error with FPGA output, fixed-point simulation with powerful MATLAB Toolbox is convenient. However, its demerits are obviously. Therefore, this paper proposes a novel fixed-point simulation library for the design of FPGAs. Four modules of library provide plenty of tools to assist fixed-point simulation easily and quickly. Simulation shows comparison between MATLAB and library.

However, there are a lot of elements to add into the library. Firstly, IP core function library needs more function; Secondly, width optimum module is still coarse and the function to adjust bit-width is less efficient and time-consuming; Thirdly, there should be some DSP options to add into the basic definition module.

References

[1] Tom Hill, Floating-Point to Fixed-Point MATLAB Algorithm Conversion for FPGAs, DSP Journal, 2006:5.

[2] Kyungtae Han, Alex G.Olson, Brian L.Evans, "Automatic Floating-Point to Fixed-Point Transformations," Fortieth Asilomar Conference on Signals, Systems and Computers, 2006. ACSSC '06. pp.79-83.

[3] Linsheng Zhang, Yan Zhang, Wenbiao Zhou, "Floating-point to Fixed-point Transformation using Extreme Value Theory," 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science, 2009, pp 271-276.

[4] W. Sung and K. Kum, Simulation-based word-length optimization method for fixed-point digital signal processing systems", IEEE Trans. Signal Process, vol.43, no.12, pp.3087-3090, Dec. 1995.

[5] K. I. Kum and W. Sung, Combined word-length optimization and high-level synthesis of digital signal processing systems, IEEE Trans. Computers, vol. 20, no. 8, pp. 921–930, Aug.2001.

[6] Chungen Liu, "Floating computation-programming principle, implementation and application," 2008, in press.

[7] Thomas Hill, "AccelDSP Synthesis Tool Floating-Point to Fixed-Point Conversion of MATLAB Algorithms Targeting FPGAs," XILINX WP239(v1.0) April 19,2006.

[8] The Mathworks Company, Fixed-Point Toolbox™ 3 User's Guide, 2010.3

[9] W. G. Obsborne, R. C. C. Cheung, J. G. F. Coutinho, W. Luk, "Automatic accuracy-guaranteed bit-width optimization for fixed and floating-point systems," Proc. FPL2007, pp. 617–620, Aug. 2007.

[10] D.U. Lee, A. A. Gaffar, R. C. C. Cheung, O. Mencer, "Accuracy-guaranteed bit-width optimization", IEEE Trans. Comput. -Aided Design Integr. Circuits Syst., vol.25,no.10,pp. 1990–2000, Oct.2006.

[11] Laurens de Haan, Ana Ferreira, "Extreme Value Theory: An Introduction". Springer. 2010

[12] Qian Gao, "Design and Implementation of Feature Extraction Unit for Speaker Recognition System," Zhengzhou : National Digital Switch System Engineering & Technological R&D Center, 2008.

[13] The Synphony Company, *Synphony HLS User Guide*. 2009.

Bingyang Liu was born in Weihui, China, in 1985. He received the B.S. degree of automation in 2004. He is currently studying for a M.S degree in communication and information system from National Digital Switch System Engineering &Technological R & D Center. He has worked on the design and implement of FPGAs on speech recognization at NDSC, Zhengzhou. From 2008 to 2010, he participated in the project of Henan Province Science and Technology Bureau on speech recongnization and EV-DO data transmission. His research interests are in the general areas of high performance computer, digital signal processing and communication.

Xiaojun Zhang was born in Henan, China, on July 23, 1969. He received the M.S. of communcation in 2000. From 2000 to 2005, he participated in the high-performance digital program-controlled switches project at NDSC. He worked on circuit-swithced network subsystem design, FPGAs design, and high-speed communications problems. Presently he is engaged in research on high-speed dedicated cryptographic chip, multimedia processing ASIC design and many-core processing techniques at NDSC. He is coauthor of the book " Modern SoC Design Technology (Publishing House of Electronic Industry, 2009). He is member of National Digital Switching System Engineering &Technological R&D Center.

Zhenchao Chang was born in Handan, China, in 1987. He received the B.S. degree in electrical engineering and computer science from the University of Yanshan, Qinhuangdao. He is currently a graduate student in National Digital Switch System Engineering & Technological R & D Center. His research interests are in the areas of communication, electronic design and hardware design.