A High-Level Synthesis and Verification Tool for Fixed to Floating Point Conversion

Semih Aslan[1], Erdal Oruklu[2], and Jafar Saniie[2]

[1]Ingram School of Engineering, Electrical Engineering
Texas State University
San Marcos, Texas, 78666, USA

[2] Department of Electrical and Computer Engineering
Illinois Institute of Technology
Chicago, Illinois, 60616, USA

Abstract — **A flexible and efficient fixed to floating point conversion tool is presented for digital signal processing and communication systems. Fixed point numbers are heavily used in digital systems because they require less hardware, verification time and design effort compared to floating point number systems. However, floating point numbers offer better precision. Some digital designs may use a hybrid number system wherein fixed and floating point numbers can be used together to improve accuracy. The proposed design tool converts fixed-point numbers to floating-point numbers, including IEEE-754 floating point number standard. This tool generates Verilog RTL code and its testbench that can be implemented in FPGA and VLSI systems. The proposed design tool can increase productivity by reducing the design and verification time. The generated design has been implemented on Xilinx Virtex-5 FPGAs and compared to conventional fixed to floating conversion tools.**

I. INTRODUCTION

Arithmetic operations in signal processing and communication systems are implemented using either fixed-point, floating-point or hybrid number systems wherein fixed [1] and floating point numbers [2] can be used together in the same chip using some conversion tools [3][4][5]. The IEEE754-1985 standard was released for binary floating-point arithmetic [6]. New features were added to this standard and IEEE released the latest standard IEEE754-2008 for binary floating point arithmetic. The current standard includes half precision, single precision, double precision and quadruple precision also known as binary-16, binary-32, binary-64 and binary-128, respectively [6].

The single precision binary-32 floating point number is shown in Figure 1. The most significant bit (index number 32) is the sign bit. The next 8 bits that are indexed as bit numbers 30-23 are biased exponents and the last 23 bits are fractions [7]. The decimal calculation of the single precision floating point number is;

$$x = (-1)^{s}(1.x_{22}x_{21}x_{20}\ldots x_{0})_2\ 2^{(\varepsilon-127)}$$

which is equivalent to

$$X_{Decimal} = (-1)^{S}\left(1+\sum_{i=0}^{22} x_i 2^{i-23}\right)2^{(\varepsilon-127)}$$

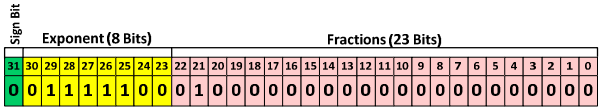

Figure 1. The single precision binary-32 floating point number

The hardware implementation of fixed-point number systems requires less hardware than floating-point number systems. The implementation of addition in a floating point number system can be particularly difficult and will consume more hardware than fixed-point numbers. The fixed-point numbers are limited to the number of bits used. For example, representing the current U.S. National debt, which is 15,450,932,542,123 dollars [8], requires 44-bit fixed points. Representing the same debt in Japanese yen, which is 1,241,173,411,108,740, requires a 51-bit fixed-point number system. Both of these numbers can be represented using single precision 32-bit floating numbers 0x5560D736 and 0x5511A955, respectively. Another advantage of a floating-point number system is in the representation of smaller numbers [7]. For example, the charge and mass of an electron are $1.60217646 \times 10^{-19}$ and $9.10938188 \times 10^{-31}$, respectively. To represent the charge of an electron using a fixed point number system, an 80-bit is required. Error during this representation is 6.5653×10^{-25}. To reduce this error, the number of bits needs to be increased to 88-bit. This error is reduced to 3.4346×10^{-25}.

To improve design accuracy and reduce design time, a custom fixed to floating point conversion tool is proposed. This tool is described in Section II and an example design and its error analysis is discussed in Section III.

II. FIXED TO FLOATING POINT CONVERSION SYSTEM

The GUI of the proposed fixed to floating point conversion system is shown in Figure 2. This conversion tool takes an *n*-bit fixed point input and creates an *m*-bit floating point. This tool creates IEEE 754 and custom *m*-bit based hardware and testbench using Verilog HDL. The testbench is created using MATLAB and functional verification is done using Modelsim. This code is also checked using LEDA [9] for ASIC and FPGA compatibility.

978-1-4673-2527-1/12/$31.00 ©2012 IEEE

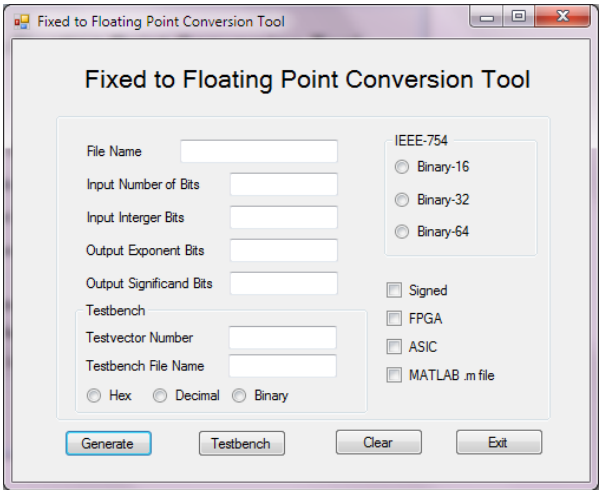

Figure 2. Fixed to floating point conversion GUI

Based on user input parameters, the fixed to floating point conversion system generates synthesizable RTL design and verification files. It also generates MATLAB files for error analysis and Modelsim Tcl scripting files for verification. The main program is designed using Visual Basic, Perl, and Tcl scripting languages. The design generates RTL code and testbench verification files for hardware design.

The proposed fixed to floating point conversion system shown in Figure 2 can accelerate time to market (TTM) by reducing the verification time if IEEE-754 floating point or custom floating point arithmetic is required. The components of this conversion system are:

- File name: An optional component where users can enter floating point Verilog HDL file and folder where all files will be stored. The default file and folder name will be *Fixed_Float_n* if user leaves this textbox empty.
- Input number of bits: This is a required entry where the total number of bits of a fixed point number is entered. This field must be an integer. All other entries will create an error.
- Input integer bits: This is an optional input where the integer number of bits needs to be entered. If this field is left blank, the program will assume the fixed point number is an integer.
- Output exponent bits: This is a required entry where the total number of exponent bits of a floating point number is entered. This field must be an integer. All other entries will create an error.
- Output mantissa bits: This is a required entry where the total number of mantissa bits of a floating point number is entered. This field must be an integer. All other entries will create an error.
- Testvector number: This is an optional entry where the number of testvectors must be entered. The current system accepts testvectors between two and one million. The default number 1000 will be used if this field is left blank.
- Testbench file name: The user can use custom test values that are stored in a text file. This file name and extension must be entered and the file must be located in the same folder where the fixed to float conversion program is running. The current system only accepts one custom test file. In addition, the user needs to identify the testvectors as hexadecimal, decimal or binary numbers.
- IEEE-754: This field needs to be used if fixed to IEEE-754 floating conversion is needed. The current system supports half precision (16-bit), single precision (32-bit) and double precision (64-bit) IEEE-754 standards. If this field is checked, it will overwrite the custom level numbers.
- Signed: This is an optional check entry where the default is unsigned numbers.
- FPGA: If this design is intended for use in FPGA designs, this field needs to be checked. The current tool is realized for Altera- and Xilinx-based FPGAs.
- ASIC: If this design is intended to use for ASIC designs, this field needs to be checked. If both FPGA and ASIC fields are checked, the system will run LEDA [9] scripting for verification. The user computer must be equipped and licensed to run LEDA.
- MATLAB .m file: As an option, a testvector file can be generated in .m file format.
- Generate button: This will execute the program and create a floating point Verilog HDL file and error graphs based on user inputs.
- Testbench button: This program only generates testbench files for Modelsim and MATLAB. It does not generate a floating point Verilog HDL file. It is intended to use if the user already has a design under test (DUT) file.

The idea behind the fixed to floating point conversion system is similar to the High Level Synthesis (HLS) [10] design flow shown in Figure 3.

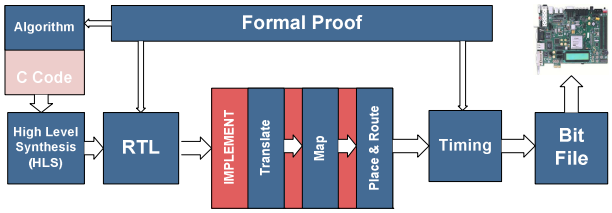

Figure 3. HLS [10] Design flow

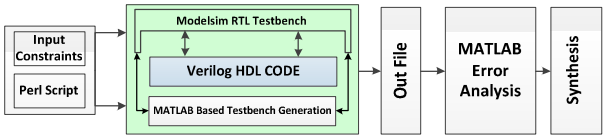

Figure 4. The fixed to floating point design and verification flow

The design and verification flow of the proposed designer system is shown in Figure 4. During generation of the Verilog HDL file, number of the testvectors and testbench file will be also generated and result files from Modelsim will be transferred into MATLAB for error analysis. If the results are correct and error in the acceptable range, the synthesis flow will be the next level of operation.

Based on the user input parameters, the system will compute the possible maximum and minimum values for fixed point numbers. After computing the maximum and minimum values based on the size and integer bit values, the fixed to floating point conversion tool generates a Verilog HDL file and testbench files using the input

constraints. The Modelsim Tcl file generates a project file, compiles the DUT and testbench and creates fixed point and floating point data files that contain simulation results from Modelsim. After completion of these files, the next step is to run generated MATLAB .m files and compare fixed and floating point numbers for error. During the generation of the MATLAB .m file for comparison and verification process, some of the built-in MATLAB functions such as *dec2bin*, *bin2dec*, *hex2dec*, etc. are used. Most of these built-in functions are used for unsigned numbers. If the user input contains unsigned numbers, additional function files will be generated during the verification process. Two of these functions used for unsigned numbers are shown in Figures 5 and 6 respectively.

```
function [D]=bin_dec(A,k)
    [m,n]=size(A);
    D=zeros(1,m)';
    for i=1:m
     if (A(i,1)=='1')
        m=bin2dec(A(i,:))-2^n;
        m=m/2^(n-k);
      else
        m=bin2dec(A(i,:));
         m=m/2^(n-k);
     end
     D(i)=m;
    end
    D;
end
```

Figure 5. MATLAB .m file for signed and unsigned binary to decimal conversion

```
function [N]= floating_to_decimal(R)
   F = dec2bin(hex2dec(R),32)
   S = F(1);
   M = F(10:32);
   E = F(2:9);
   N1 =(2^(bin2dec(E)-127))*(1+bin2dec(M)/2^23);
if (S == '0')
    N = N1
else
    N = -1*N1
end
```

Figure 6. MATLAB .m file for floating point to decimal conversion

III. Case Study

To understand the functionality of the fixed to floating point conversion system, a real industry application, *Received Total Wideband Power (RTWP),* is presented in this section. RTWP is an important factor for uplink load and interference and needs to be calculated and reported effectively [11]. The calculation, calibration and reporting RTWP and power meters are outside the scope of this paper [11][12]. This paper focuses on the custom fixed-to-floating conversion that is used in RTWP. The block diagram and circuit description of RTWP are shown in Figure 7 and Figure 8.

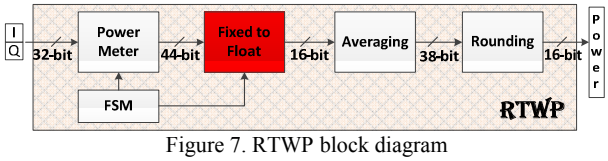

Figure 7. RTWP block diagram

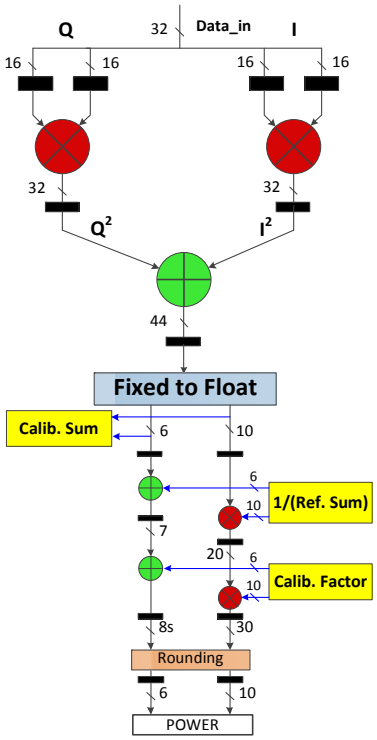

Figure 8 An example of RTWP calculation in LTE systems

As seen in Figure 7 and Figure 8, when using 16-bit I/Q signals, the total power is calculated as the average of I^2+Q^2 over every slot that is every 0.5 ms, and the sum value is later stored as a floating point number. The fixed to floating point conversion tool is used to design hardware for converting a 44-bit fixed point number to a 16-bit floating point for power report. The 44-bit fixed point number has a range of [0,1) and floating point reporting needs to be in the same range with minimal error. One of the most complicated blocks in RTWP is fixed to floating point conversion. The block needs to be validated after intensive testing. This could take a long time and can increase the design time as well as TTM. The proposed design can generate the required Verilog HDL design file and varies it using the testbench that is generated during the generation of the design file. This design flow is similar to the HLS design flow that is given in Figure 3. The tool verifies the design during the design process and this will decrease the design time significantly based on input parameters.

The fixed to floating point conversion tool input parameters are shown in Figure 9. Based on these input parameters, MATLAB generates 50 testvectors during the Verilog HDL design file generation. Next, the testbench file and a Tcl file are generated to test the design using Modelsim for functional verification. The output files are later transferred into MATLAB for error analysis. During the verification flow that is explained in Figure 4, Modelsim waveform is output for verification as well as data files for MATLAB analysis as shown in Figure 10.

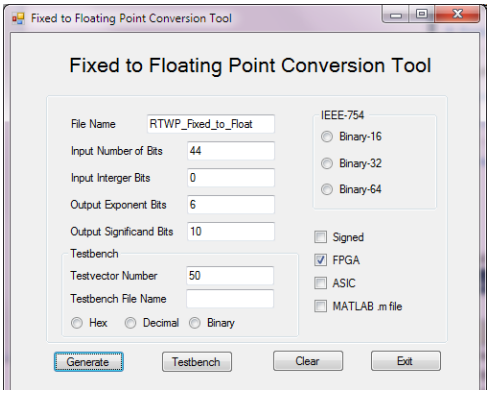

Figure 9 Fixed to Floating point generation for RTWP

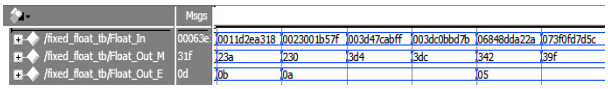

Figure 10. Modelsim verification of RTWP calculation.

After Modelsim simulations, the results are analyzed using MATLAB. The fixed to floating point conversion tool generates necessary MATLAB files and runs them during the verification process. The code that is shown in Figure 11 loads data that is generated by Modelsim into MATLAB and compares it with fixed point values. The error analysis between 44-bit fixed point and 16-bit floating point numbers is shown in Figure 11.

```
load RTWP_Fixed_to_Float_Fixed.dat;
load RTWP_Fixed_to_Float_Exponent.dat;
load RTWP_Fixed_to_Float_Mantissa.dat;

[m,n]=size(RTWP_Fixed_to_Float_Fixed);

A_Fixed=dec2bin(RTWP_Fixed_to_Float_Fixed)/2^44;
A_Float=(dec2bin(RTWP_Fixed_to_Float_Mantissa)/2.^10).* ...
    (2.^(-dec2bin(RTWP_Fixed_to_Float_Exponent)));

error_Conv_percent= abs(A_Fixed-A_Float)/A_Fixed;
plot(1:m,error_Conv_percent)
```

Figure 11. MATLAB .m file for error analysis

The Figure 12 is the MATLAB error analysis output that is shown in Figure 4. It simply compares 44-bit fixed point number and 16-bit floating point number for error analysis. The relative error that is shown in Figure 12 is less than 0.1 %.This design is later transferred into Xilinx ISE design tool for synthesis and implementation using a Xilinx Virtex5 LX110 XUP evaluation board. It only requires 84 LUTs out of 69120 in Virtex 5.

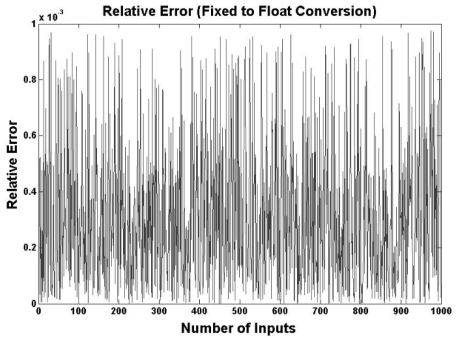

Figure 12. Relative error during conversion

IV. Conclusion

A fixed to floating point conversion tool is presented in this design. This design tool generates fixed to floating point conversion hardware using the IEEE-754 standard floating point and as well as custom floating point numbers. An HLS approach is used to design and verify the HDL code. The HDL code is verified using Modelsim and MATLAB and synthesized using the ISE design tool. The key objective of the proposed tool is to reduce the TTM and increase productivity by verifying the hardware during the design process. Future work will include support for all IEEE-754 2008 standards and also a floating point to fixed point conversion section for design flexibility. In addition, designers can easily design mixed fixed and floating point hardware without sacrificing TTM.

Acknowledgment

The authors would like to thank Xilinx, Inc. [13] for their valuable support.

References

[1] Z.-H. Tan and B. Lindberg, "Fixed-Point Arithmetic," in *Automatic Speech Recognition on Mobile Devices and over Communication Networks*, Springer, 2003, pp. 255 - 275.

[2] M. Clemmesen, "Interval Arithmetic Implementations: Using Floating Point Arithmetic," *ACM Signum Newsletter*, 1984, no. 1.

[3] E. DUMAN, H. CAN, and E. AKIN, "FPGA Modules for Conversions between Fixed and Floating-point in Quartus-II Environment," *International Journal of Electrical & Computer Sciences*, 2010, vol. 10, no. 6, pp. 120-124.

[4] L. Saldanha and R. Lysecky, "Float-to-Fixed and Fixed-to-Float Hardware Converters for Rapid Hardware/Software Partitioning of Floating Point Software Applications to Static and Dynamic Fixed Point Coprocessors," *Design Automation for Embedded Systems*, 2009, vol. 13, no. 3, pp. 139-157.

[5] X. Wang and M. Leeser, "VFloat : A Variable Precision Fixed- and Floating-Point Library for Reconfigurable Hardware," *ACM Transactions on Reconfigurable Technology and Systems*, Sep. 2010, vol. 3, no. 3, pp. 1-34.

[6] M. Standards, *IEEE Std 754™-2008 (Revision of IEEE Std 754-1985), IEEE Standard for Floating-Point Arithmetic*, vol. 2008, no. August. IEEE Computer Society, 2008.

[7] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, and G. Melquiond, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2009, p. 595.

[8] "The US Debt Clock," 2012. [Online]. Available: http://www.usdebtclock.org.

[9] Synopsys, "Synopsys," 2012. [Online]. Available: http://www.synopsys.com/tools/verification/functionalverification/pages/leda.aspx.

[10] C. Desmouliers, S. Aslan, E. Oruklu, J. Saniie, and F. Matrinez Vallina, "HW/SW co-design platform for image and video processing applications on Virtex-5 FPGA using PICO," *Computer Engineering*, 2010, pp. 1-6.

[11] E. Dahlman, S. Parkvall, J. Skold, and P. Beming, *3G Evolution, Second Edition: HSPA and LTE for Mobile Broadband*. Academic Press, 2008, p. 648.

[12] 3gpp.org, "3rd Generation Partnership Project; Technical Specification Group Radio Access Networks; Requirements for Support of Radio Resource Management (FDD) (Release 5)," 2002.

[13] "Xilinx .com." [Online]. Available: http://www.xilinx.com.