ISSN (e): 2250 – 3005 || Volume, 06 || Issue, 04||April – 2016 ||
International Journal of Computational Engineering Research (IJCER)

Matlab Based High Level Synthesis Engine for Area And Power Efficient Arithmetic Operations

Semih Aslan
Ingram School of Engineering Texas State University San Marcos, Texas, 78666, USA

Abstract
Embedded systems used in real-time applications require low power, less area and a high computation speed. For digital signal processing (DSP), image processing and communication applications, data are often received at a continuously high rate. Embedded processors have to cope with this high data rate and process the incoming data based on specific application requirements. Even though there are many different application domains, they all require arithmetic operations that quickly compute the desired values using a larger range of operation, reconfigurable behavior, low power and high precision. The type of necessary arithmetic operations may vary greatly among different applications. The RTL-based design and verification of one or more of these functions may be time-consuming. Some High Level Synthesis tools reduce this design and verification time but may not be optimal or suitable for low power applications. The developed MATLAB-based Arithmetic Engine improves design time and reduces the verification process, but the key point is to use a unified design that combines some of the basic operations with more complex operations to reduce area and power consumption. The results indicate that using the Arithmetic Engine from a simple design to more complex systems can improve design time by reducing the verification time by up to 62%. The MATLAB-based Arithmetic Engine generates structural RTL code, a testbench, and gives the designers more control. The MATLAB-based design and verification engine uses optimized algorithms for better accuracy at a better throughput.
Keywords: *FPGA, High Level Synthesis, MATLAB, Optimized Hardware, Power Efficient, RTL.*

I. Introduction

Today, a significant number of embedded systems are focused on multimedia applications with almost insatiable demand for low cost, high performance and low power hardware. Designing complex systems such as image and video processing, compression, face recognition, object tracking, 3G or 4G modems, multi-standard CODECs, and HD decoding schemes requires the integration of many complex blocks and a long verification process [1][2]. These complex designs are based on I/O peripherals, one or more processors, bus interfaces, A/D, D/A, embedded software, memories and sensors. In the past, complete systems were designed with multiple chips and connected together on PCBs, but with today's technology, all functions can be incorporated in a single chip. These complete systems are known as System-on-Chip (SoC) [2].System-on-chip (SoC) designs are mainly accomplished by using Register Transfer Languages (RTL) such as Verilog and VHDL. RTL design flow [1] [2] for both FPGA and ASIC is similar and is shown in Figure 1. An algorithm can be converted to RTL using the behavioral model description method or by using pre-defined IP core blocks. After completing this RTL code, formal verification must be done before implementation. After implementation of the RTL code, timing verification needs to be done for proper operation.

RTL design abstracts logic structures, timing and registers [1]. Because of this, every clock change causes a state change in the design. This timing dependency causes every event to be simulated, which results in a slower simulation time and longer verification period of the design. The design and verification of an algorithm in RTL in Figure 1 can take up 50-60% of the "Time to Market" (TTM). The RTL design becomes impractical for larger systems that have high data flow between the blocks, and it requires millions of gates.

Even though design time may improve by using behavioral modeling and IP cores, the difficulty in synthesis, poor performance results and rapid changes in the design make IP cores difficult to adapt and change. Therefore, systems rapidly become obsolete.The limitations of RTL and longer TTM forced designers to think of the design as a whole system rather than blocks. In addition, software integration in SoC was always done after the hardware was designed. When the system gets more complex, software integration is desirable during hardware

implementation. Over the last two decades, designers were forced to find new methods to replace RTL due to improvements in SoC and shorter TTM. Because of the extensive work done in Electronics System Level Design (ESLD), HW/SW co-design of a system and High Level Synthesis (HLS)[2][4] are integrated into FPGA and ASIC design flow.

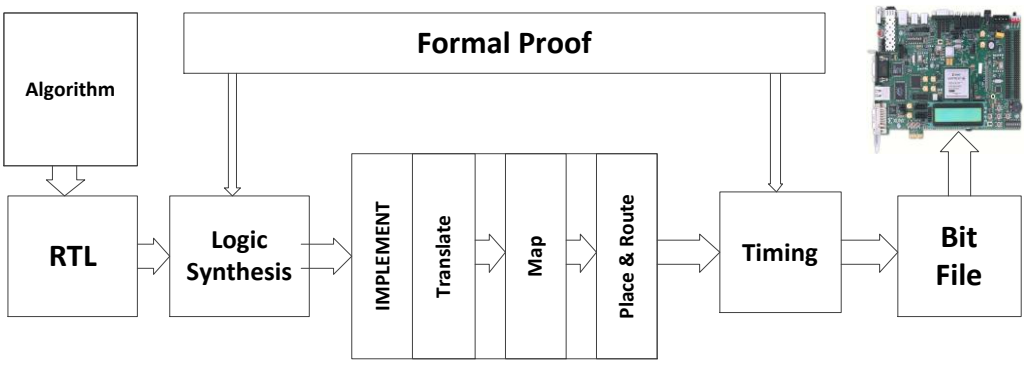

Figure 1. FPGA RTL level synthesis flow

The next section will describe the proposed MATLAB HLS Arithmetic (MHA) Engine design and implementation. Section III and IV will focus on the error analysis and testbench generation respectively and the conclusion will describe future work and improvements.

II. Mha Engine

RTL description of a system can be implemented from a behavioral description of the system in Perl, C, Python and MATLAB. This will result in a faster verification process and shorter TTM. It is also possible to have a hybrid design where RTL blocks can be integrated with HLS [2].The HLS design flow shows that a group of algorithms that represent the whole system or parts of a system can be implemented using a high level language such as Perl, C, C++ , Java, MATLAB [2][5]. Each part in the system can be tested independently before the whole system is tested. During this testing process, the RTL testbenches may also be generated. After testing is complete, the system can be partitioned into HW and SW. This enables SW designers to join the design process during HW design; in addition, RTL can be tested by using both HW/SW together. After the verification process, the design can be implemented using FPGA synthesis tools.The integration of HLS into FPGA design flow is shown in Figure 2.

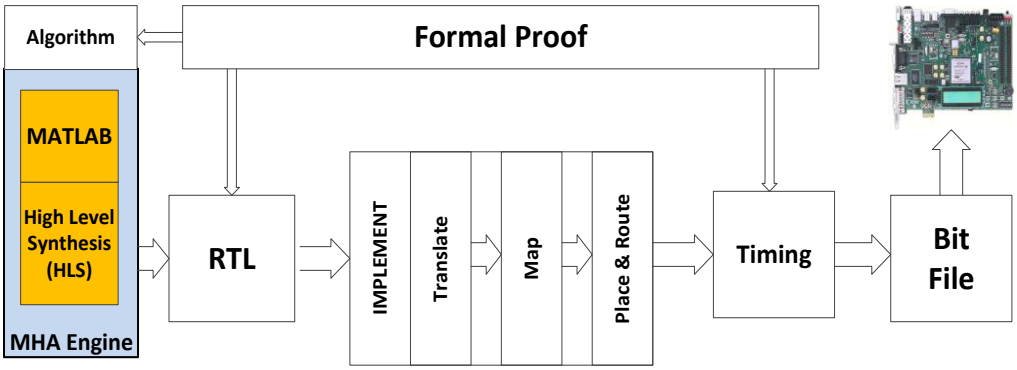

Figure 2. FPGA high level synthesis flow with MHA Engine

Since the early days of VLSI design, application-specific hardware has been used for optimal implementation of algorithms. This approach is considered the fastest design scheme but is also the most area consuming system due to the inherently redundant nature of a design that only computes one operation. However, there are other possible designs for DSP implementations that can be used for two or more operations [1][3]. This design approach consists of processing blocks that can compute multiple operations using dedicated hardware designed for a particular cluster of operations. An improved design approach should exploit the redundancy and common elements that exist among the sub-blocks. This would result in shared building blocks and dramatically reduced hardware requirements.

The proposed design focuses on designing a large system that will be faster with a design principle similar to HLS, asexplained above. The main work focuses on multi-purpose, reused hardware structures that produce a unified, area efficient reconfigurable system. This design can reduce the area by 64% [2][[6]. The components of the MHA Engineare shown in Figure 3.

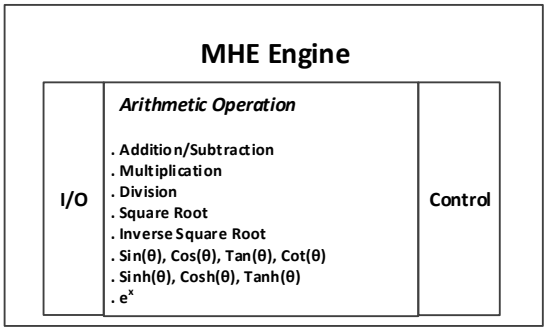

Figure 3. MATLAB Based HLS Arithmetic Engine (MHA) block

The MHA block has three important principles:

- Compute required arithmetic operations
- Customized range and accuracy
- Generate an area-efficient, fast system for low power applications

The MHA accepts inputs from the user via two GUIs to make it more user friendly and efficient. The "Main" GUI that is shown in Figure 4 below includes the following sections:

- FPGA or ASIC support
- Vendor based IP Core support
- Project Name (default is c:\MHA\MHA)
- Top Module Name (default is MHA)
- Language – Verilog or VHDL (design and verifications – current system only supports Verilog HDL)
- Rounding - Truncation or RNE (Rounding cannot be done without a selection)
- Number system – Fixed or Floating Point (Fixed point up to 64-bit - current system only supports Fixed Point Number system)

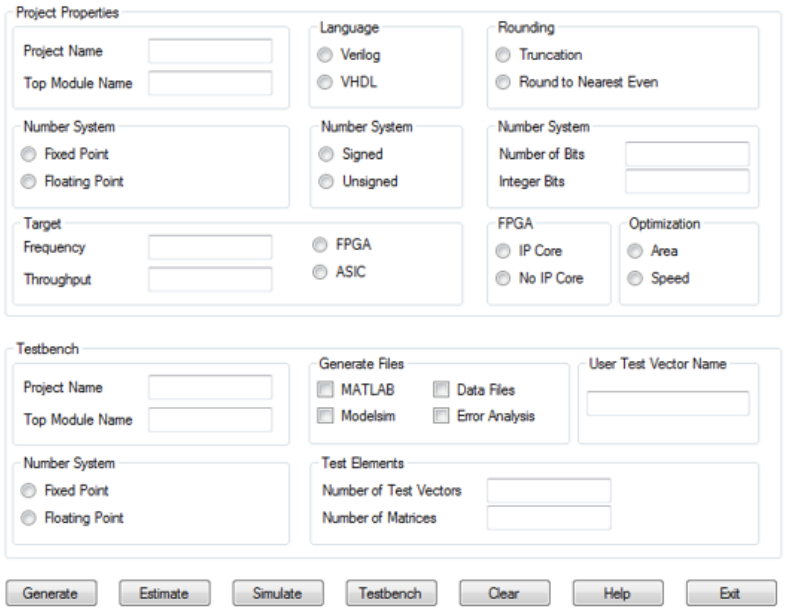

Figure 4. Main GUI for MHA Engine

- Signed or unsigned number systems
- Target – Frequency and throughput
- Area or speed based optimization

- Testbench generation
  - Automated testbench with MATLAB
  - Modelsim .do file for fast automation
  - Automated testbench file for Modelsim
  - Error comparison with MATLAB
  - User defined test data option

The "Arithmetic" tab shown in Figure 5 has the following sections:

- Basic arithmetic operations
  - Addition/Subtraction (Area or speed optimized based on Ripple-Carry Adder (RCA) or Carry-Lookahead Adder (CLA))
  - Multiplication (Array or Booth multiplier)
- Advanced arithmetic operations
  - Division (Newton-Raphson, Goldschmidt, or CORDIC)
  - Square Root (Newton-Raphson, Goldschmidt, or CORDIC)
  - Inverse Square Root (Newton-Raphson, Goldschmidt, or CORDIC)
- Elementary functions
  - Trigonometric functions – sine, cosine, tangent, and cotangent (table method, CORDIC or polynomial based design)
  - Hyperbolic functions – sinh, cosh, tanh (table method, CORDIC or polynomial based design)
  - Exponential function – exp(x) (table method, CORDIC or polynomial based design)

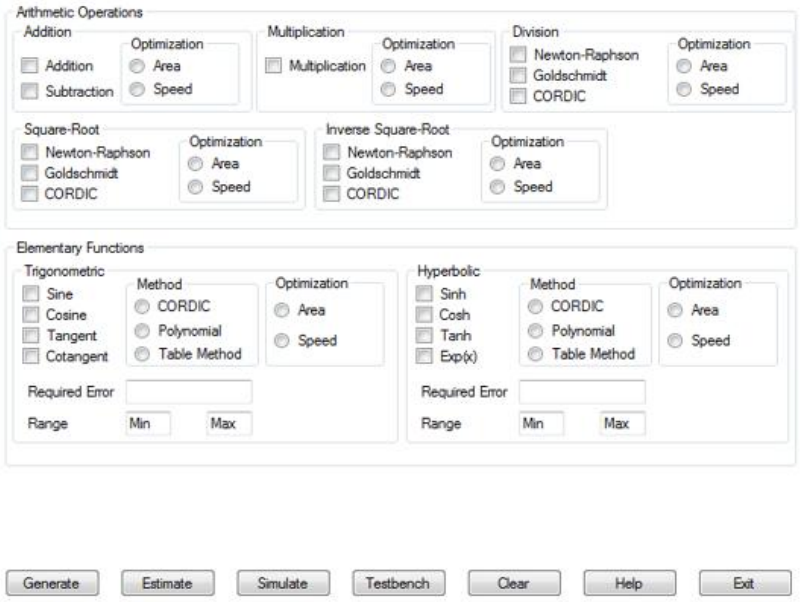

Figure 5. Arithmetic operations GUI

The MHA uses a bottom-up design process that starts with the elementary functions and then moves to the simplest arithmetic operations such as multiplication and addition. This is shown in Figure 6 below.

This design flow contains the following procedure: First, selection of elementary functions [7], selection of basic arithmetic operations, and generation of area efficient hardware for FPGA and VLSI. There are 2-64 bit selections that are suitable for a vast variety of applications with the requested precision. The section's addition and multiplications are used based on the previous designs. Division, inverse square root and square roots are designed based on the same architecture, and the modified design reduces the area by 64% [8]. Next, the CORDIC [9] [10] or polynomial methods [4] are used to calculate elementary functions [7] [11]. This area-efficient design is optimized for speed by implementing a smart control system. For performance evaluation and synthesis are implemented with Xilinx FPGAs [12] and Microwind [13] VLSI design tool.

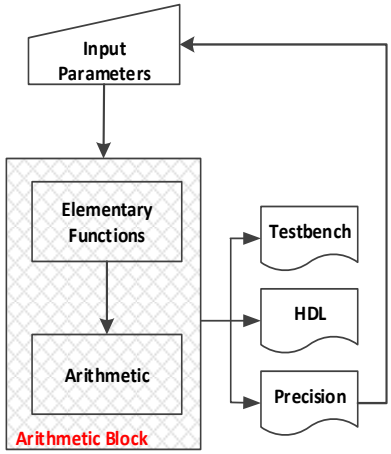

Figure 6. The MHA design flow

III. Error Analysis

When designing hardware with many arithmetic operations, one of the most important objectives is to produce results that have a minimal absolute and average computation error. Arithmetic operations in digital systems generally introduce three types of errors: number representation, rounding, and algorithmic or design error [14][15].The exact representation of some numbers or events in radix-n may not be possible due to the limitations in ADCs, the sampling rate, and the number of available bits. In addition, many numbers cannot be converted from radix-n to radix-m without an error. For example, number 0.1 and 0.2 in radix-10 cannot be represented in radix-2 without an error. This error can be reduced by increasing the number of bits. The reduction of this error with respect to the number of bits is shown in Figure 7.

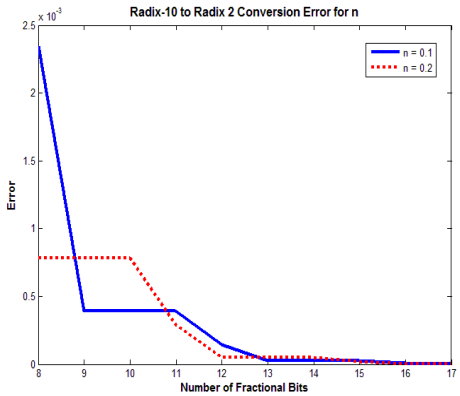

Figure 7. Error representation of number radix conversion

Figure 8 shows the error generated for radix-10 to radix-2 conversion of 128 fractional numbers (fractional part of 30 bits).

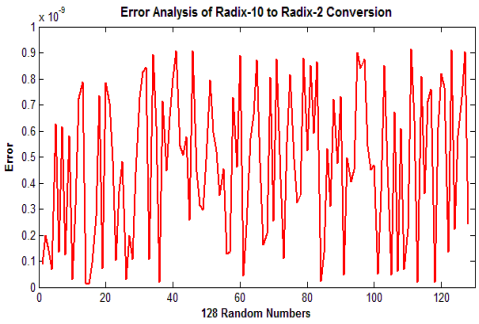

Figure8. Random 128-number conversion error from Radix-10 to Radix-2

During and after the calculation of certain arithmetic operations, the total number of bits may exceed the number of bits available; these values need to be rounded. For example, multiplying two n-bit numbers produces a product of 2n-bit and this result may need to be represented with n-bit.If there were more multiplications on the design path, the number of bits would increase in a linear fashion. To prevent this, each multiplier output needs to be rounded. There are a few ways to implement rounding in hardware, with the most commonly used methods being round to the nearest even, round towards zero (truncation), round down (floor), round up (ceiling) and round away from zero [14].
In this section, truncation (TRA)[14] and round to the nearest even (RNE)[14] schemes are compared. During these rounding operations, an error value is introduced. The RNE and TRA and their error values are shown in Table 1.

Table 1. RNE and TRA

Number	RNE		TRA	
	Rounded Value	Error	Rounded Value	Error
X0.00	X0.	0.00	X0.	0.00
X0.01	X0.	0.25	X0.	0.25
X0.10	X0.	0.50	X0.	0.50
X0.11	X0.+ulp	-0.25	X0.	0.75
X1.00	X1.	0.00	X1.	0.00
X1.01	X1.	0.25	X1.	0.25
X1.10	X1.+ulp	-0.50	X1.	0.50
X1.11	X1.+ulp	-0.25	X1.	0.75
Total	---	0	---	3.00

Figure 9 shows advantage of RNE over TRA when average error is considered. The requested number of precision and selected rounding scheme can affect the size of the hardware and overall speed and throughput. Users can change the selected accuracy to see area and throughput estimates simultaneously without increasing the overall design time. This will make it possible to select the optimal design for synthesis. The MHA Engine can create hardware for precision based on increase the bit size during the mid-operation and apply rounding before the output stage.

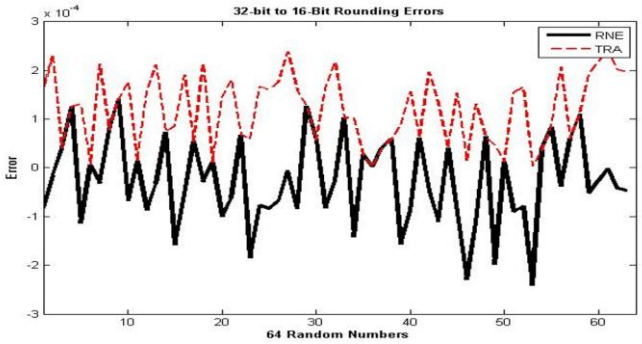

Figure 9. 32-Bit to 16-Bit Rounding Errors with RNE and TRA

Iv. Testbench Generation

One of the most important and complicated sections of the MHAEngine is generation of the testbench files and error checking using MATLAB and Modelsim. Before the RTL code is synthesized, it can be tested using a testbench that is created using MATLAB and Modelsim. The testbench generation and error checking block diagram is shown in Figure 10 below.

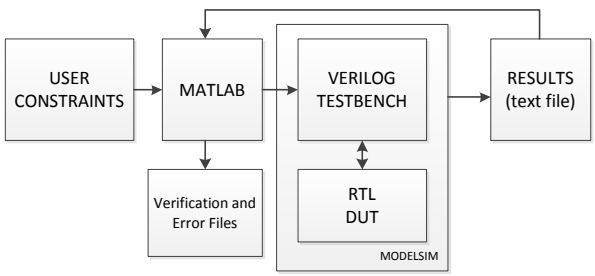

Figure 10. MATLAB and Modelsim flow

After generation of the design and testbench files,the user defined test vectors need to be generated. The first step is to generate positive random numbers [0,1]. These numbers must be converted into positive and negative numbers based on signed numbers. To generate random test values and test results, the following procedure is followed:

- Get the user defined testvector number n.
- Generate random test vectors (T{n})
- Generate random binary numbers using MATLAB
  - For fixed-point signed numbers:
    *T_Bin=dec2bin(T*2^n,n)*
  - For fixed-point unsigned numbers:
    *T_Bin=dec2bin(T*2^(n-i),n)*
- Generate the Modelsim testbench file and get results
- Compare results using MATLAB

After generation of the design, testbench, and test vector files, the next step is to generate a Modelsim tcl .do file that can be transferred into Modelsim all together. The .do file will generate the project file and will import all design files, including testbench, into Modelsim. This will run all files and generate the results as a text file. Once Modelsim-generated results are imported into MATLAB, correct operation and error analysis needs to be performed. An important issue which needs to be addressed during the verification process is working with negative fixed-point numbers in MATLAB. It is important because it does not convert negative binary numbers and binary floating numbers to a decimal number. This problem is addressed using the MATLAB codes given in Figure 11.

```
function [T]= signed_bin_dec(F,n,in)
   [i,t]=size(F);
for t=1:i
   K=F(t,:);
    if (K(1)=='0')
    N=bin2dec(K)/(2^(n-in));
    else
    N=(bin2dec(K)-2^n)/2^(n-in);
    end
    T(t,:)=N;
 end
```

Figure 11. Signed binary to decimal conversion

V. Conclusion

An area efficient, MATLAB based HLS engine for arithmeticoperations is designed for low power and high-speed applications. The MHA Engine decreases design system time and verification by up to 64% without compromising speed and efficiency. The MHA Engine uses a smart control system that is optimized based on the desired operations. The MHA Engine is a bridge between RTL and HLS. It uses RTL-based basic blocks to design most complicated arithmetic operations using structural model design and HLS-style fast and optimized verification. Any designed system can be reconfigured at any time in any way in MHA Engine without going through the same design and verification hassle.

MATLAB-based verification makes it possible to use all the features of MATLAB for faster and more efficient verification. The MHA Engine can be easily reconfigurable to systems available at any level, due to changes in the computer system and software.

As explained above, this system generates area efficient fast arithmetic and elementary functionsthat can be used over a wide area of applications in DSP, image processing, and communication systems. It can be used for FFT, DCT and DWT calculations and Chirplet transforms [16][17][18].

It can also be very important for educational institutions in order to test their systems using the verification testbench that, when desired, works as an independent design tool. Overall, this work will forge the way for those who need to make sudden changes in their systems and need fast verification. They can adopt and apply any changes using the MHA Engine or generate similar systems for faster design and verification. In addition, because the MHA Engine generated code is designed structurally, code can be changed easily at any level, if so desired. Future workwill integrate VHDL code and IEEE 754 floating point numbers (both single and double precision) implementation. Another future goal is to make this platform totally open source by using only Iverilog and replacing MATLAB with Octave.

References

[1]. Hendry, D.C., and A.A. Duncan. "Area Efficient DSP Datapath Synthesis." Design Automation Conference (1995): 130-135.

[2]. Aslan, S., Oruklu, E., and Saniie J., "A high-level synthesis and verification tool for fixed to floating point conversion", IEEE International Midwest Symposium on Circuits and Systems, 2012, Pages, 908-911.

[3]. Andrieux, J., M. Feix, G. Mourgues, P. Bertrand, B. Izrar, and V. Nguyen. "Optimum Smoothing of the Wigner Ville Distribution." IEEE Transactions on Acoustics, Speech, and Signal Processing 36.5(1987): 764-769.

[4]. Kilts, S. Advanced FPGA Design Architecture, Implementation, and Optimizations. New York: Wiley Inter-Science, 2007.

[5]. Chen, W. The VLSI Handbook. Boca Raton: CRC Publisher, 2007.

[6]. Dehon, A., and S. Hauck. Reconfigurable Computing The Theory and Practice of FPGA-Based Computing. Burlington, Massachusetts: Elsevier, 2008.

[7]. Walther, J.S. "A unified Algortihm for Elementary Functions." American Federation of Information Processing Societies Joint Computer Conferences (1971): 379-385.

[8]. Oruklu, E., J. Saniie, and S. Aslan. "Realization of Area Efficient QR Factorization using Unified Diivision, Square Root and Inverse Sqaure Root Hardware." IEEE Electro/Information Technology (2009): 245-250.

[9] Volder, J. "The CORDIC Trigonometric Computing Technique." IEEE Transactions Electronic Computers 8.3 (1959): 330-334.

[9]. 10] Striling, W. C., and T. K. Moon. Mathematical Methods and algorithms for Signal Processing. New Jersey: Prentice Hall, 2000.

[10]. Xilinx. (2016), http://www.xilinx.com/

[11]. Microwind (2106) http://microwind.net/

[12]. Stine, J. E. "Digital Computer Arithmetic Datapath Design using Verilog HDL." Digital Computer Arithmetic Datapath Design using Verilog HDL. Norwell, Massachusetts: Kluwer Academic Publishing, 2004.

[13]. Teukolsky, S. A., W. T. Vetterling, B. P. Flannery, and W. H. Press. "Numerical Recipes: The Art of Scientific Computing." Numerical Recipes: The Art of Scientific Computing, 3rd ed. New York, New York: Cambridge University Press, 2007.

[14]. Omar, J., E. E. Swartzlander Jr., and M. J. Schulte. "Optimal Initial Approximations for the Newton-Raphson Division Algorithm." Springer-Verlag Journal of Computing 53.3-4 (1994): 233-242.

[15]. Seidel, P-M, W. E. Ferguson, and G. Even. "A Parametric Error Analysis of Goldschmidt's Division Algorithm." Journal of Computer and System Sciences 70.1 (2005): 118-139.

[16]. Chunduri, K. C. "Implementation of Adaptive Filter Structures on a Fixed Point Signal Processor for Acoustical Noise Reduction" (2006).