High Speed Special Function Unit for Graphics Processing Unit

Abd-Elrahman G. Qoutb[1] , Abdullah M. El-Gunidy[1], Mohammed F. Tolba[1], and Magdy A. El-Moursy[2]
[1]Electrical Engineering Department, Fayoum University, Fayoum, Egypt.
[2]Design Creation Division, Mentor Graphics, Cairo, Egypt.
Emails: ag1186@fayoum.edu.eg, elgunidy@gmail.com, mf1173@fayoum.edu.eg, magdy_el-moursy@mentor.com

***Abstract*— A fixed-point ASIC design for high-speed, second-order, piecewise function approximation is presented. A Non-Uniform segmentation method based on Minimax approximation is used to get the interpolation coefficients. Non-Uniform segmentation, effectively, reduces the size of the coefficient table with a small area overhead for the address encoder. The proposed algorithm truncates the binary coefficients within the pre-allocated error. Radix-eight Booth multipliers are used to reduce the number of partial products to, around one third of the traditional multiplication, hence speeding up the evaluation process. Very fast reduction trees with four-to-two compressors are used to reduce the number of the resulting partial products. Also, a new radix-eight sign template which reduces the overall area of the multipliers is proposed. Hybrid carry-look ahead, carry-ripple adders are, also, used. The design has been verified on FPGA. Moreover, 45nm PDK is used to synthesize and layout the design. A maximum propagation delay of 5.251ns is achieved with a reduction of 19% in the total delay as compared to other traditional methods. A total chip area of 0.014mm^2 is also achieved.**

Keywords—Special Function Unit (SFU), Nmeric Function Generator (NFG), Vertix Shader Processor, GPU, Hybrid Multiplier, Minimax, Non-Uniform Segmentation, Booth Multiplier.

I. INTRODUCTION

The graphics processor implements 3D graphics pipeline to speed-up rendering. Rendering is the process of converting a 3D scene into a 2D image to be displayed on a computer screen [1]. Modern graphics processors are composed of different building blocks, such as Programmable Vertex Shader (PVS), Programmable Fragment Shader (PFS), and basic circuits such as the Power Management Unit [2]. PVS is responsible for performing all per-vertex operations while PFS is responsible for per-pixel operations. The block diagram for the Graphical Processing Unit (GPU) is shown in Figure 1.

Vertex Shaders are mainly composed of a Special Function Unit (SFU), and a Single Instruction Multiple Data core (SIMD). SIMD core is used to perform all kinds of matrix operations which are needed in the graphics pipeline, where SFU is dedicated to perform numeric functions. In this paper, Special Function Unit (SFU) is designed to improve the performance of the Graphical Processing Unit (GPU). SFU implementation is based on non-uniform segmentation so the area of LUT can be reduced.

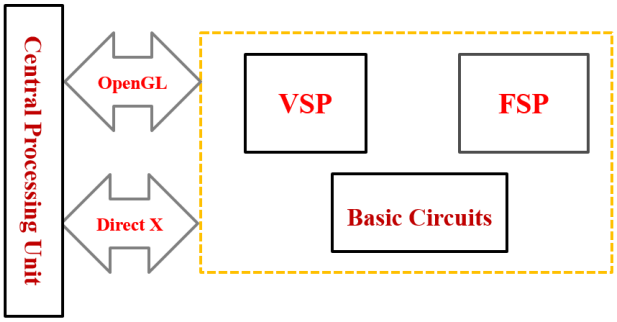

Figure 1 Abstract Block Diagram of Graphics Processing Unit (GPU)

Function evaluation algorithms are divided into two main categories; Iterative and Non-Iterative. Iterative methods are based on the multiplication operation and typically have quadratic convergence. They result in low latency, especially for high precision computations [3]. In Non-Iterative methods, specific operations are performed and the final value appears with single constant error approximation, so it is commonly used in Numeric Function Generators (NFGs). NFGs allow the computation of difficult mathematical functions in short time and with minimal hardware than the commonly employed methods. They compute piecewise linear (or quadratic) approximations that represent the value of the original function for a given input value. The domain of the NFG is divided into enough segments. The approximation results in errors which should remain within a required range. The overall hardware complexity and propagation delay depend on the number of the required segments, the arithmetic devices which are used to approximate the function, and the number of bits which are used to represent the numbers being calculated.

Polynomial coefficients are stored in Look-Up Table (LUT). The size of the LUT and the area of NFGs vary depending on

978-1-4799-8200-4/14/$31.00 ©2014 IEEE

the technique used to evaluate the function. Table-based methods reduce the number of arithmetic operations but increase the size of the LUT. Compute-bound methods, use a small LUT but require a fair amount of arithmetic hardware. They also require computation time to calculate the final function values. Hybrid methods [4, 5], represent a good compromise between computation resources and the table size. They are widely used in many applications that require high-speed function calculation [6].

This paper presents an enhanced Table-based method which divide the domain of the function into Non-Uniform segments to guarantee the required input approximation error. The proposed technique adopts Hybrid multiplier and Wallace Tree Multi-Operand Adder to determine the final result.

The paper is organized as follows; The design of Special Function Unit is presented in section II where the modified Non-Uniform Segmentation with Remapped addresses is presented. The Hybrid-Multiplier is used to merge the use of Booth-Three Partial Product Generator and is described in section III. The Wallace tree to sum the generated partial products and the final output using Multi-Operand Adder is presented in section IV. Some simulation results are provided in section V. Some conclusions are summarized in section VI.

II. Special Function Unit

The SFU is responsible for evaluating the mathematical functions which are used in the rendering pipeline. Among the widely used mathematical functions in GPU are Cosine and Sine functions which are heavily used to perform rotation operations on matrices [1]. In addition, the reciprocal function is used for the scaling matrix, and for perspective projection. Also, the power function is used in the Phong lighting model. The reciprocal of the square root function is used for calculating vector normalization [2]. Some of the functions which are used in the graphics applications are listed in Table 1. One of those functions is used to evaluate the presented design.

Table 1 Examples of widely used SFU Functions

Instruction	Operation
RCP	Reciprocal : X^{-1}
SQRT	Square Root : $X^{0.5}$
RSQ	Reciprocal Square Root : $X^{-0.5}$
LOG2	Base-2 Logarithm : $\log_2 X$
POW2	Base-2 Exponential : 2^X
DSQ	Division Square Root : $Y \cdot X^{0.5}$
DIV	Division : $Y \cdot X^{-1}$

Non-Iterative methods are based on reducing the elementary functions (such as the functions in Table 1 which are Examples of widely used SFU Function) using polynomial approximation. They involve approximating a continuous function with one or more polynomials on a closed interval [a; b] with the use of addition and multiplication in the processor. The degree of the employed polynomial is usually high and a large number of additions and multiplications must be performed to achieve high precision. The stored polynomial coefficients are generated using mathematical algorithms such as Chebyshev and Minimax depending on parameters such as the degree of the polynomial and the needed precision. Non-Uniform segmentation requires remapping the input value to the target address by the Address Encoder to get the polynomial coefficients. In addition, the approximated result is determined as shown in the block diagram in Figure 2. The main components of the interpolation function hardware are (1) address encoder to take some of the MSBs and return the address which has the interpolation coefficients for that input, (2) Two Booth multipliers for C_1 , C_2 multiplication by X, (3) a squarer for X^2, and (4) Multi-operand adder which contains a reduction tree for reducing the partial products and a hybrid carry-lookahead, carry-propagate adder for the final product.

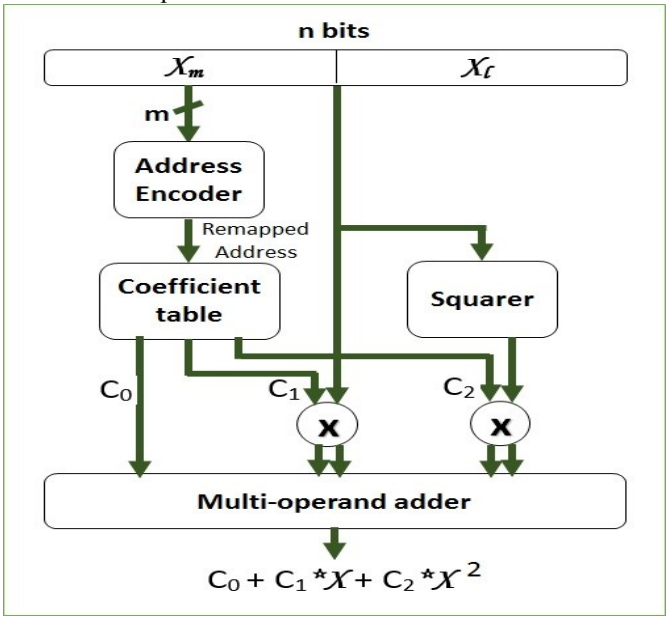

Figure 2 Non-Uniform Segmentation Block Diagram

A. Non-Uniform Segmentation

In the piecewise function evaluation with polynomial approximation, Non-Uniform segmentation can effectively reduce the size of Look-Up Tables (LUT) for many arithmetic functions as compared to uniform segmentation approaches. This is achieved at the cost of the extra segment address (index) encoder that results in area and delay overhead. Also, it is observed that the Non-Uniform segmentation has a design tradeoff between the ROM size and the area of the subsequent arithmetic computation hardware. Non-Uniform segmentation algorithm is adopted in this paper. The remapped addresses which are proposed in [7] are assumed. The goal is to minimize ROM, total area and overhead delay by searching for the optimal segmentation scheme. For arithmetic functions which have a lot of changes in the predetermined interval, the proposed segmentation method achieves significant area reduction as compared to the Uniform segmentation method.

The desired function is approximated by a second-order Minimax polynomial on the desired interval. The interval is

divided into many equal-size intervals to get a reasonable trade-off between the arithmetic circuits which are used in the evaluation process, the LUT which is used to store the interpolation coefficients, and the approximation error. All equal-size intervals are passed through the algorithm which is shown in Figure 3, where e is the approximation error, S_i stands for Uniform segment, C_i is the Non-Uniform segment and Size(C_i) represents how many uniform segments in the Non-Uniform segment C_i. This algorithm is used to reduce the number of segments achieving less number of entries to be saved in the LUT.

The algorithm starts with dividing the entire interval into uniform segments then merge all the uniform segments, to guarantee the approximation error, into Non-Uniform segments. Non-Uniform segments are divided into power of two sizes then sorted to make the indexing regular and avoid the use of additional hardware.

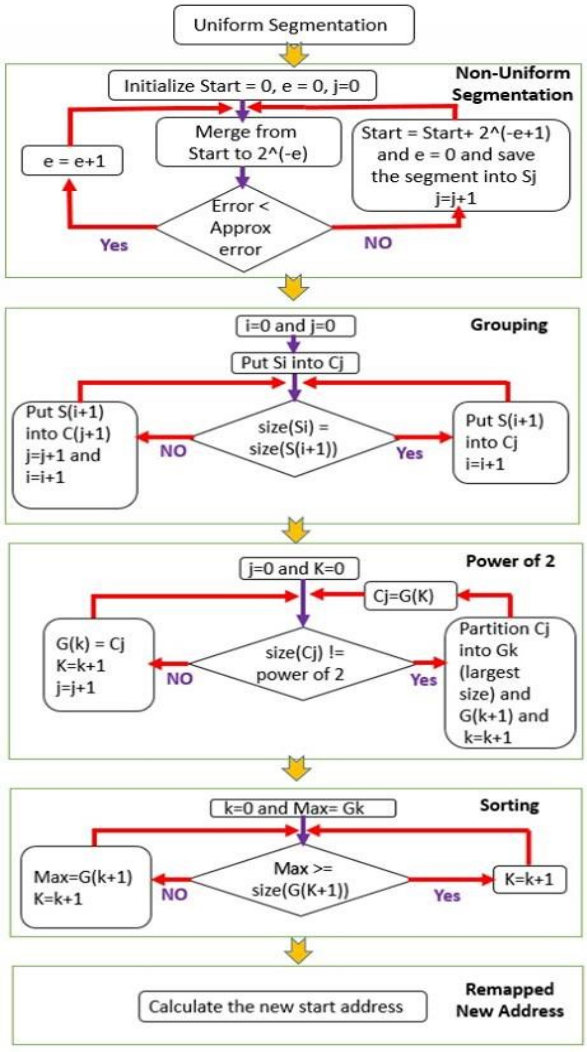

Figure 3 Remapped Non-Uniform Segmentation Flow Chart

Maple is a commercial computer algebra system tool which is used to generate the coefficients using Minimax() Function provided by the tool. A new step is added to choose the minimum widths for the coefficients which are stored in the memory to achieve the target approximation error. This is done by truncating the resulted coefficients to fixed widths, which come from the Maple program to reduce the memory requirement, while keeping the pre-specified approximation error. This step is based on iterating over all the possibilities for the coefficient widths, taking into account the error result due to rounding in each case. The minimum width which achieves the target approximation error that can be applied for each Non-Uniform segment is chosen. To evaluate the function, the polynomial equation needs to be performed. Hybrid multiplier is used by applying the concept of Booth-Multiplier and Wallace Tree as shown in Section III.

III. Booth-Three Partial Products Generator

Parallel multipliers are usually used in graphics processing applications as they achieve high performance. Hence, they are used for the evaluation of the polynomial equation (1) of degree two, where C_0, C_1 and C_2 are the polynomial coefficients which are stored in the LUT and generated by the Minimax mathematical algorithm.

$$f(x) \approx C_0 + C_1 x + C_2 x^2 \qquad (1)$$

High-Speed multipliers consist of high-radix Booth partial products generator and carry-save tree to reduce the number of the resulting partial products to two partial products. A carry-propagate, or carry-look ahead, adders are widely used to generate the final product [8]. A carry-look ahead adder is used because the carry bit *"i"* is computed without waiting for carry bit "i-1" (i=1,2,3,4, etc.) so solving the carry delay problem, Booth-3 can decode every four bits of the multiplier generating one partial product. This would reduce the number of the partial products by a factor of three or four, with a small area overhead of the Booth decoder. The number of the partial products which are generated for 10-bit (m_0 to m_9) signed multiplier are shown in Figure 4. For every four bits there is a code which is used to generate one partial product except for the first three bits which are added with '0' bit to generate the first partial product, bit *m9* is repeated twice to complete the fourth bit for the last partial product *pp3* without changing the multiplier value. Four partial products are generated (pp_0, pp_1, pp_2, pp_3).

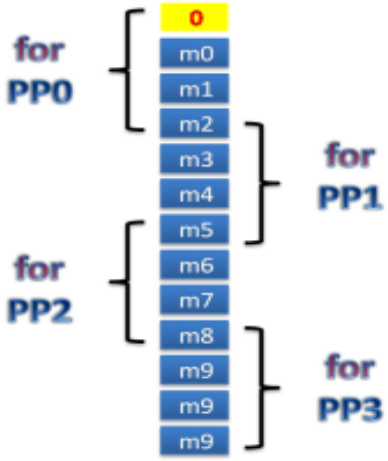

Figure 4 Multiplier Grouping for Booth-3 10-bit Signed Multiplier

A scalable, homogeneous, gate-level implementation of the Booth-3 decoder is used [8] to guarantee low-power operation and to achieve more regular layout and smaller area as shown in Figure 5. The four bits are provided to the Booth-3 decoder to generate the select *iM* signals. These signals are used to generate each bit of the partial product after being passed to a function with the multiplicand bits.

After the partial products are being generated, each partial product is shifted logically by three bits, the 1st partial product is shifted zero bits, the 2nd partial product is shifted by 3 bits and the 3rd is shifted by 6 bits and so on. Sign extension for each partial product is used to obtain correct results. This way is not suitable for high performance because it increases the capacitive load on the sign bit, hence increasing area and power consumption, and finally, degrades performance.

Instead, there are ready-made sign-templates to avoid the sign extension process. The one proposed in [8] works well for signed multiplication, but the conditions for generating the sign extension signal, called *E*, result in some area overhead. A summary of these conditions is given here:

- Partial product is positive if $S = 0$.
- Partial product is negative if $S = 1$.
- $E = 1$ if the select bits select positive multiple and the multiplicand is positive. Or if the select bits select negative multiple and the multiplicand is negative. Or finally if the select bits are +0.
- $E = 0$ if the select bits select positive multiple and the multiplicand is negative. Or if the select bits select negative multiple and the multiplicand is positive. Or finally if the select bits are -0.

The Logic function which performs the sign extension signal for the *E* is shown in Figure 6.

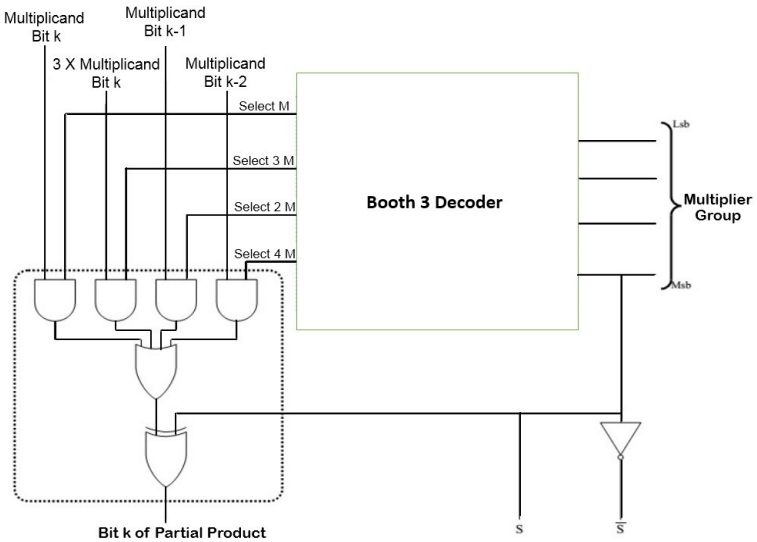

Figure 5 Partial Product Generator Block diagram.

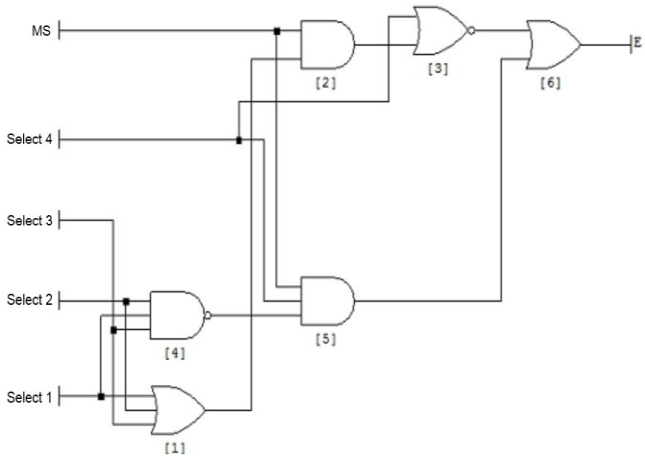

Figure 6 Logic Circuit to Generate E for Booth-3

The proposed method is about pre-extending the multiplicand by two additional bits to account for (*3*M*), and then execute the same sign-template as if it is unsigned multiplication as shown in Figure 7. The signal, *C*, is a copy of the sign bit of the multiplicand after extension. *S* is the sign bit of the select bits. This implementation increases the performance and decreases the overhead area because there is no need to use the logic for generating the *E* signal. The copy signal, *C*, does not need a logic circuit to be generated, yet ensuring right results. For the given example of 10-bit multiplication, the proposed method saves four times the overhead logic of generating the *E* signal. This improvement increases as the number of partial products increases.

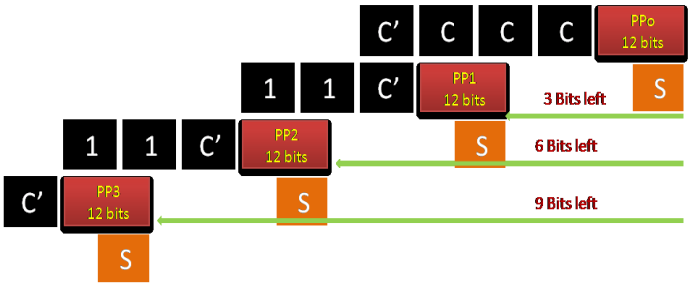

Figure 7 Proposed Sign Template for 10-bit Signed Multiplier

Another significant improvement in the proposed algorithm is achieved by using a circuit to disable the power-hungry adder circuitry in the case of generating $\mp 3 * M$ when it is not used, hence reducing the total power consumption as shown in Figure 8. There is no need to use a carry-propagate adder to generate the hard multiple $(\mp 3 * M) = M + 2*M$. given that it only operates when the select bits choose $\mp 3 * M$. As all the generated partial products need to be added together and the final outputs from each multiplier are to be calculated, the need of Multi-operand Adder is raised as discussed in section IV.

Segment width	Start segment	C_0	C_1	C_2
16	0	0.9999389648	-0.9927825928	0.8372802734
16	16	0.9963989258	-0.935333252	0.5992431640
16	32	0.9868774414	-0.8586997986	0.4434814453
16	48	0.9722290039	-0.7798805237	0.3374023437
16	64	0.9537353516	-0.7055931091	0.2626342773
32	80	0.9228515625	-0.609462738	0.1877441406
32	112	0.8764038086	-0.5019607544	0.1256103515

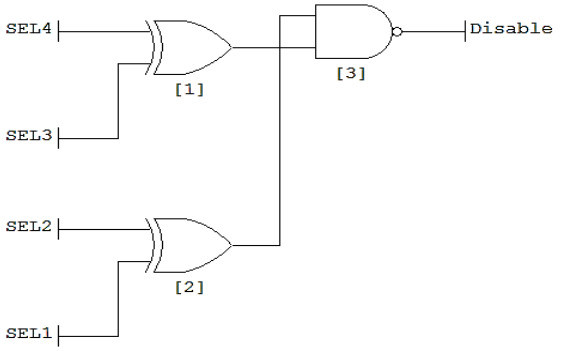

Figure 8 Proposed Circuit to Disable the Adder That Generates the Hard Multiple, 3M.

IV. Multi-Operand Adder

Wallace Tree with Carry Propagate Adder achieves Multi-Operand Addition with small area and propagation delay. Wallace Tree is known for their optimal computation time when adding multiple operands to two outputs using Half Adders, Full Adders and 4:2 compressors. It is used to speed up the computation by reducing the number of the sequential addition stages. Using 4:2 compressor enhances the speed of the multiplier leading to smaller number of stages in performing the final result. The critical path delay using the compressor is compared with Full-adders and Half-adders. The performance of the Wallace Tree varies depending on the structure and how many blocks are used in each stage. It is used here to add the partial products in a tree-like fashion in order to produce two rows of partial products that can be added in the last stage. The tree structure and the stages to generate the final result for summing the partial products are shown in Figure 9 where the black dots represent the partial product bits. They are generated by 10*10 Booth-3 partial product multiplier where 2, 3 and 4 bits are added with Half Adder, Full Adder and 4:2 compressors to achieve the final value in three stages.

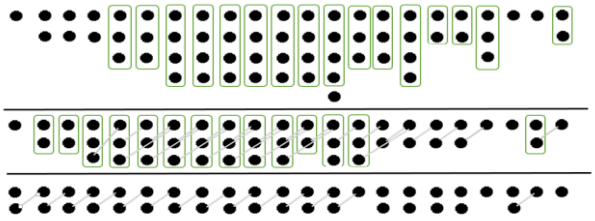

Figure 9 Wallace Tree Architecture for 10*10 Multiplier

The proposed design is prototyped and tested for the function 1/ (1+x) using ASIC/FPGA flows. The results are shown in section V.

V. Simulation Results

The function 1/ (1+x) is implemented on the input interval [0:1], with $2^{-13} \approx 0.000061035$ approximation error. The coefficients are truncated to have savings in LUT with widths of 14, 18, 14 bits with maximum truncation error of 0.000061035. The approximation coefficients C_0, C_1, and C_2 are truncated to 14, 18, 14 bits, respectively as shown in Table 2. The number of the Non-Uniform segments and the size of each segment are shown in the table. The Non-Uniform segmentation for this function resulted in 7 entries in the LUT compared to 13 entries if Uniform segmentation is used.

Table 2 Maple Interpolation Coefficients for 1/ (1+x)

As an example of how to evaluate the function for a specific input value to deal with fraction numbers, the input X value is shifted by 2^9 and the output value is shifted by 2^{13} and that is to accomplish a result of error less than 2^{-13}. As seen from these random input values in Table 3 for the function 1/ (1+x). The input value, the rounded input value after shifting back by 2^9, the output value Z from the SFU and the rounded final result after shifting back by 2^{13} are shown in the table.

Table 3 Random Test Input Values

Input X	Rounded $X \rightarrow X/2^9$	Actual $f(X)$	Output Z	Rounded $Z \rightarrow Z/2^{13}$	Error e
0	0	1	8191	0.999877	$1.23*10^{-4}$
219	0.4277	0.700427	5736	0.700195	$2.32*10^{-4}$
68	0.1328	0.882758	7230	0.88256	1.98610^{-4}
28	0.0547	0.94813	7766	0.94799	1.410^{-4}
178	0.3476	0.7420	6077	0.74182	1.7810^{-4}

The design was implemented on Xilinx XC3S500E FPGA to verify the output. ChipScope Pro is used for that purpose, and the result of the implementation is shown in Table 4.

Table 4 FPGA Implementation Results

Logic utilization	Used	Available	Utilization
Total No of 4 input LUT	1225	9312	13%
After synthesis			
Max combinational path delay	18.514 ns		
	2.717 ns logic	2.717 ns route	
After place and route			
Max combinational path delay	27.825 ns		

North Carolina State University (NCSU) 45 nm technology PDK [9] is used for synthesis and layout. The layout report is generated using Cadence SoC-Encounter and resulted in maximum propagation delay of 5.251 ns as compared to 6.53 ns for the Uniform segmentation method [10] and 6.70 ns for the Hierarchical method [11] achieving a reduction of up to 19% of the total propagation delay. A total chip area of 0.014 $mm^2$is also achieved.

VI. Conclusions

A new Special Function Unit for function evaluation is proposed. The SFU is based on the Non-Uniform segmentation and remapping algorithms. It was shown that the proposed technique achieves the minimum-size coefficient table in addition the amount of arithmetic units is improved due to the use of an enhanced Non-Uniform segmentation algorithm and Hybrid multiplier. The proposed design is faster than the previously developed methods. The function 1/ (1+x) is implemented to achieve the target accuracy of three digits after the decimal point. A new sign template for Booth multipliers is presented. The proposed technique improves the speed of these multipliers by reducing the number of partial products with no need to add specific hardware to develop the extended sign bit. The presented method is tested on Booth-3 multipliers. The Booth-3 multipliers reduces the partial products by three times the original ones. Also, a circuit for reducing the power consumption in the hard-multiple generation adder is proposed. A Maple code is written to get the best coefficients widths for truncation and storage in the LUT to minimize the memory size used. CLAs are implemented as 4-bit modules and are used in a hierarchical structure to realize adders that have multiples of 4-bits. A maximum propagation delay of 5.251 ns with a reduction of 19% as compared to other traditional methods is achieved. The total chip area is 0.014 mm^2.

References

[1] J. R. Warner, "Real Time 3-D Graphics Processing Hardware Design Using Field Programmable Gate Array" Masters Thesis, *University of Pittsburgh,* September 2008.

[2] P.-H. Wu, C.-S. Wen and L.-Y. Chen, "Design Of A Programmable Vertex Processor In Opengl ES 2.0 Mobile Graphics Processing Units*," The IEEE Proceseedings of the International Symposium on VLSI Design, Automation, and Test,* pp. 1 - 4, April 2013.

[3] J.-M. Muller, "Elementary Functions: Algorithm And Implementation, " 2nd Ed., 2006.

[4] J.-A. Pineiro, S. F. Oberman, J. M. Muller, and J. D. Bruguera, "High Speed Function Approximation Using A Minimax Quadratic Interpolator, " *The IEEE Transactions on Computers.*, vol. 54, no. 3, pp. 304–318, March 2005.

[5] M. J. Schulte and E. E. Swartzlander, "Hardware Designs For Exactly Rounded Elementary Functions," *The IEEE Transactions on Computers*, vol. 43, no. 8, pp. 964–973, August 1994.

[6] B.-G. Nam, H. KIm, and H.-J. Yoo, " Power And Area-Efficient Unfied Computation Of Vector And Elementary Functions For Handheld 3D Graphics Systems, " *The IEEE Transactions on Computers,* vol. 57, no. 4, pp. 490–504, April 2008.

[7] S. F. Hsiao; H. J. Ko; Y. L. Tseng; W. L. Huang; S. H. Lin; C. S. Wen, "Design Of Hardware Function Evaluators Using Low-Overhead Nonuniform Segmentation With Address Remapping," *The IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 875-886, May 2013.

[8] G. W. Bewick, "Fast Multiplication: Algorithms And Implementation," Ph.D. Thesis, *Stanford University*, February 1994.

[9] "http://www.eda.ncsu.edu/wiki/NCSU_EDA_Wiki,"

[10] J.-A. Pineiro, S. F. Oberman, J. M. Muller, and J. D. Bruguera, "High Speed Function Approximation Using A Minimax Quadratic Interpolator,"*The IEEE Transactions on Computers,* vol. 54, no. 3, pp. 304–318, March 2005.

[11] D. Lee, R. C. C. Cheung, W. Luk, and J. D. Villasenor, "Hierarchical Segmentation For Function Evaluation, " *The IEEE Transactions on Very Large Scale Integr. (VLSI) Systems*, vol. 17, no. 1, pp. 103–116, January 2009.