Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2014, Article ID 890354, 7 pages
http://dx.doi.org/10.1155/2014/890354

Research Article

Implementation of Special Function Unit for Vertex Shader Processor Using Hybrid Number System

Avni Agarwal, P. Harsha, Swati Vasishta, and S. Sivanantham

VLSI Division, School of Electronics Engineering, VIT University, Vellore 632014, India

Correspondence should be addressed to S. Sivanantham; ssivanantham@vit.ac.in

Received 28 May 2014; Revised 19 September 2014; Accepted 25 September 2014; Published 15 October 2014

Academic Editor: Lixin Gao

The world of 3D graphic computing has undergone a revolution in the recent past, making devices more computationally intensive, providing high-end imaging to the user. The OpenGL ES Standard documents the requirements of graphic processing unit. A prime feature of this standard is a special function unit (SFU), which performs all the required mathematical computations on the vertex information corresponding to the image. This paper presents a low-cost, high-performance SFU architecture with improved speed and reduced area. Hybrid number system is employed here in order to reduce the complexity of operations by suitably switching between logarithmic number system (LNS) and binary number system (BNS). In this work, reduction of area and a higher operating frequency are achieved with almost the same power consumption as that of the existing implementations.

1. Introduction

The various technology advancements have made the world of graphic computation more intensive. Most recent advancements of the graphic processing unit [1] include high level dynamic parallelism, which makes GPU computing easier and broadens the reach [2]. Since these GPUs are more power consuming, there has been an increasing demand for the real time 3D graphics application for mobile devices which have more stringent power constraints [3–7]. In order to meet these requirements, the application programming interface (API) has set up a standard OpenGL-ES (embedded system) [8]. At the time of this work, the latest standard was OpenGL-ES 3.1. One of the most popular versions of this standard is OpenGL-ES 2.0, which consists of a vertex shader and a fragment shader processor as the two major programmable components, along with pipelining. The programmable processors used for GPUs are termed as "shaders." The "dot product" computation, which is one of the key operations on mobile GPU, is carried out in a vectored processing unit. The lighting related operations require more complex instructions like square root, exponentiation, reciprocals, and multiply-accumulate to be performed, which are taken care of by the special function unit.

In the literature, [9] makes use of a hybrid number system (HNS) which consists of both binary and logarithmic number systems. The log and antilog conversion blocks, however, are based on a lookup table approach, which is lengthy.

This work focuses on the design of a vertex shader processor for mobile applications, consisting of logarithmic arithmetic unit and floating point arithmetic unit. In order to reduce the power consumption, the lookup table approach for the conversion of binary to log scale is done away with and an equation based method is made use of, which leads to significant reduction in the area and power.

The rest of the paper is organized into seven sections. Section 2 describes the overall architecture of SFU. The architecture of logarithmic and antilogarithmic converters is illustrated in Section 3. Section 4 describes logarithmic arithmetic unit followed by a floating point arithmetic unit in Section 5. The results are discussed in Section 6 with Section 7 concluding the paper.

2. Architecture

In this section, the architecture of the designed special function unit (SFU) is explained. The architecture, as a whole, can be divided into 4 main modules, namely, log converter (LOGC), antilogarithmic converter (ALOGC), floating point

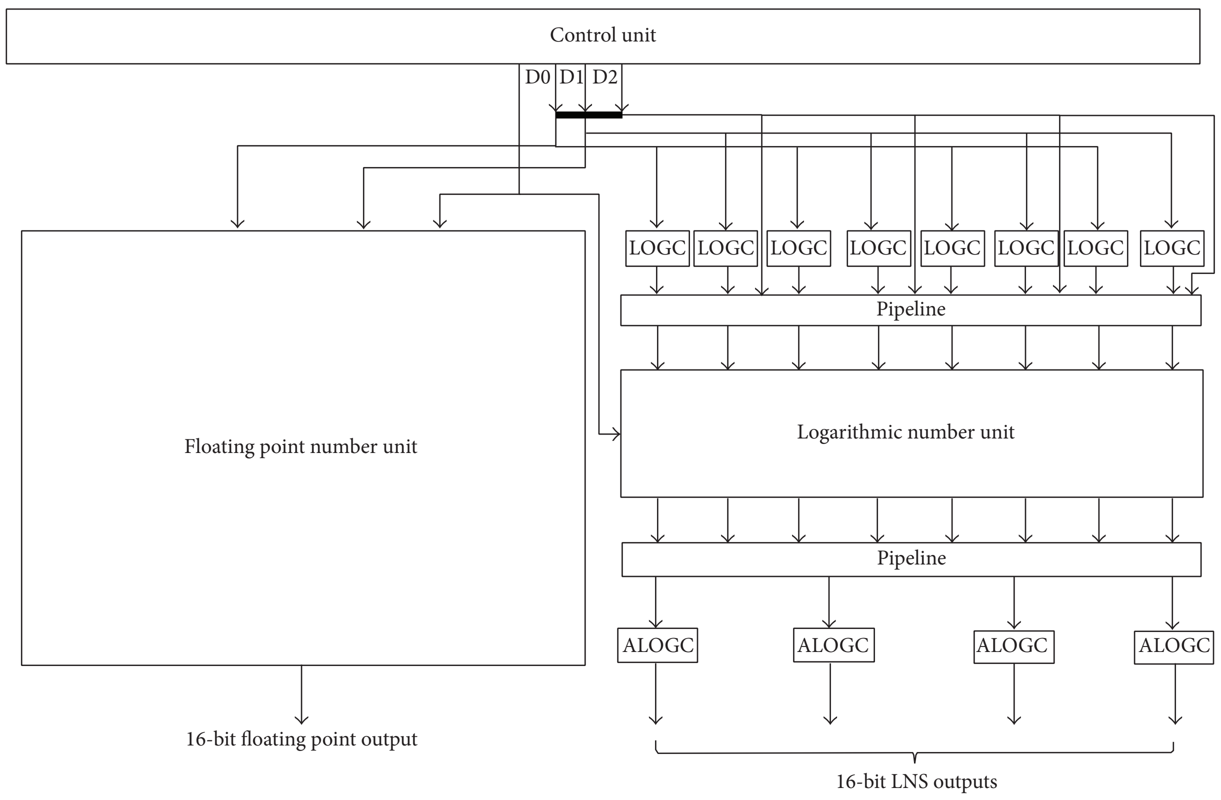

Figure 1: Architecture of special function unit.

unit (FPU), and logarithmic number unit (LNU), as shown in Figure 1. In this architecture, we have used the IEEE standard of representing floating point numbers, IEEE 754 standard, and the corresponding logic for conversions between the two formats has also been formulated. The IEEE 754 standard is depicted in Table 1.

The logarithmic converter, at its most basic functional level, converts a 16-bit floating point number into a 16-bit logarithmic number. The base of the logarithmic number in this design is 2 since the processing is internally in binary arithmetic. This logarithmic operation is performed so as to reduce the complexity of operations such as powering and computation of square root, since these operations reduce to simple algebraic operations in logarithmic number system. For example, $(X * Y)$ reduces to $(\log X + \log Y)$ and X^Y reduces to $(Y * \log X)$, and so forth. The operations presented in this paper are listed in Table 2.

Although logarithmic conversions reduce the complexity in few operations, it also complicates certain simple operations such as addition and subtraction, as can be seen from Table 2. Therefore, using LNS for all the operations may not be an optimal choice. This brings the need of embedding both logarithmic number system and floating point number system, thus making it a hybrid number system. The usage of these number systems is dictated by a control logic which takes in the operation to be performed as an input and suitably activates the required number system.

The logarithmic arithmetic unit (LAU) is involved in processing of logarithmic numbers. The outputs of LAU are in logarithmic scale which is later converted back in an antilogarithmic converter. However, when the operations to be performed are simple and straightforward there is no need for logarithmic/antilogarithmic conversions. In [9], the operands are passed through all the blocks irrespective of the operation being performed. But, in this design, for simple operations, the operands are directed to FAU bypassing

Table 1: (a) FLP-16—half precision floating point formation (FLP-16), (b) LNS-16—Q5.11 format for logarithmic numbers.

(a)

Sign (1 bit)	Exponent (5 bits)	Mantissa (10 bits)

(b)

Sign (1 bit)	Integer (5 bits)	Fraction (10 bits)

Table 2: Selection of number system based on operation.

Operation	Number systems BNS	Number systems LNS	System used
Multiply	$X * Y$	$\log X + \log Y$	LNS
Divide	X/Y	$\log X - \log Y$	LNS
Square-root	$\sqrt{X}$	$(1/2) * \log X$	LNS
Power	X^Y	$Y * \log X$	LNS
Add	$X + Y$	$\log X + \log(1 + 2^{\log Y - \log X})$	BNS
Subtract	$X - Y$	$\log X - \log(1 + 2^{\log Y - \log X})$	BNS

LAU. This prevents unnecessary switching operations in LAU, thereby reducing power.

If the logarithmic number system is used, it is preceded with a logarithmic converter and followed by an antilogarithmic converter. If the FAU is being used, logarithmic/antilogarithmic conversions are not required. In this architecture, three stages of pipelining have been introduced—one stage consisting of the logarithmic converters, second stage consisting of LAU, and the final stage consisting of antilogarithmic converters. However, pipelining is made internal in FAU. The internal architecture of individual modules is explained in the subsequent sections.

3. Logarithmic and Antilogarithmic Converters

As discussed in Section 2, operations such as calculation of power, square root, multiplication, and division can be tedious and time consuming if the conventional binary number system (BNS) is adopted. Hence, it brings in the need for logarithmic number system which effectively reduces the complexity of such operations. The most basic block in the logarithmic number system is logarithmic/antilogarithmic converter. The logarithmic converter (LOGC) is used to convert a 16-bit binary number into a 16-bit logarithmic number. The width size of 16-bits corresponds to the usage of IEEE 754 floating point standard representation. The antilogarithmic converter (ALOGC) performs the reverse of operation of LOGC.

Logarithmic converters can be implemented in multiple ways. The Lookup table (LUT) approach has been employed in [9]. However, LUTs can have a massive area overhead as the size of the operands and the number of operations increase. Since the core application of this design is vertex processing, although there can be a small-scale precision variation, it can be traded for higher speed. In this paper, we propose an architecture that uses an equation based method (EBM) that eliminates the need for large LUTs, thereby, reducing considerable number of gates.

The architecture of LOGC and ALOGC is based on Algorithm 1. For logarithmic conversion the coefficients of a and b are calculated depending on the computation of the value y. The coefficients of a and b for LOGC are given in Table 3. A similar but reverse approach is used for antilogarithmic conversion. The coefficients of a and b for ALOGC are given in Table 4.

Algorithm 1. It computes the base-2 logarithmic value of a nonzero binary number.

Consider B to be a nonzero binary number.

Step 1. $B = b_m b_{m-1}, \ldots, b_{k+1} b_k$, where $b : 0/1$;

$$m, k = 0, \pm 1, \pm 2, \ldots; \quad m \geq k. \tag{1}$$

It follows logically that

$$B = \sum_{i=k}^{m} b_i 2^i. \tag{2}$$

Table 3: Coefficients of a and b for logarithmic conversion.

Range	Mantissa
$0 \leq y < (1/4)$	$y^* = y + 37y/128 + 1/128$
$(1/4) \leq y < (1/2)$	$y^* = y + 3y/64 + 1/16$
$(1/2) \leq y < (3/4)$	$y^* = y + 37\overline{y}/64 + 1/32$
$(3/4) \leq y < 1$	$y^* = y + 29\overline{y}/128$

Table 4: Coefficients of a and b for antilogarithmic conversion.

Range	Mantissa
$0 \leq y < (1/4)$	$y^* = y + 1/4\overline{y} + (3/4)$
$(1/4) \leq y < (1/2)$	$y^* = y + 13/128\overline{y} + 55/64$
$(1/2) \leq y < (3/4)$	$y^* = y + 9/128y + 7/8$
$(3/4) \leq y < 1$	$y^* = y + 35/128y + 23/32$

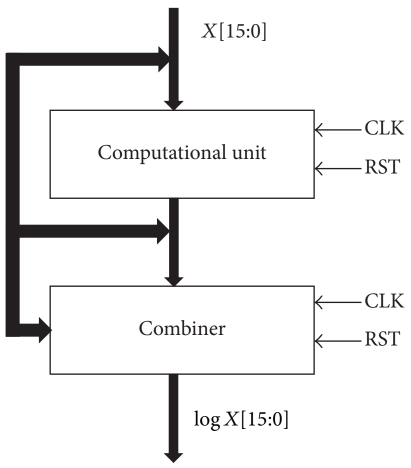

Figure 2: LOGC and ALOGC basic architecture block diagram.

Step 2. Splitting the MSB and the other LSBs,

$$B = 2^j + \sum_{i=k}^{j-1} b_i 2^i. \tag{3}$$

Step 3. Reducing the equation by taking 2^j as a common factor,

$$B = 2^j \left(1 + \sum_{i=k}^{j-1} b_i 2^{i-j} \right) = 2^j (1 + y), \tag{4}$$

where $y = \sum_{i=k}^{j-1} b_i 2^{i-j}$.

Step 4. $B = 2^j(1 + y) \rightarrow \log_2 B = j + \log_2(1 + y)$.

Step 5. Approximating the equation obtained in Step 5 using a linear approximation,

$$\log_2 B \approx j + ay + b. \tag{5}$$

The basic architecture of the proposed LOGC is as shown in Figure 2. Since the operands use IEEE 754 format, 1-bit of sign, 5-bits of exponent and 10-bits of mantissa and processed separately. The architecture consists of two main units.

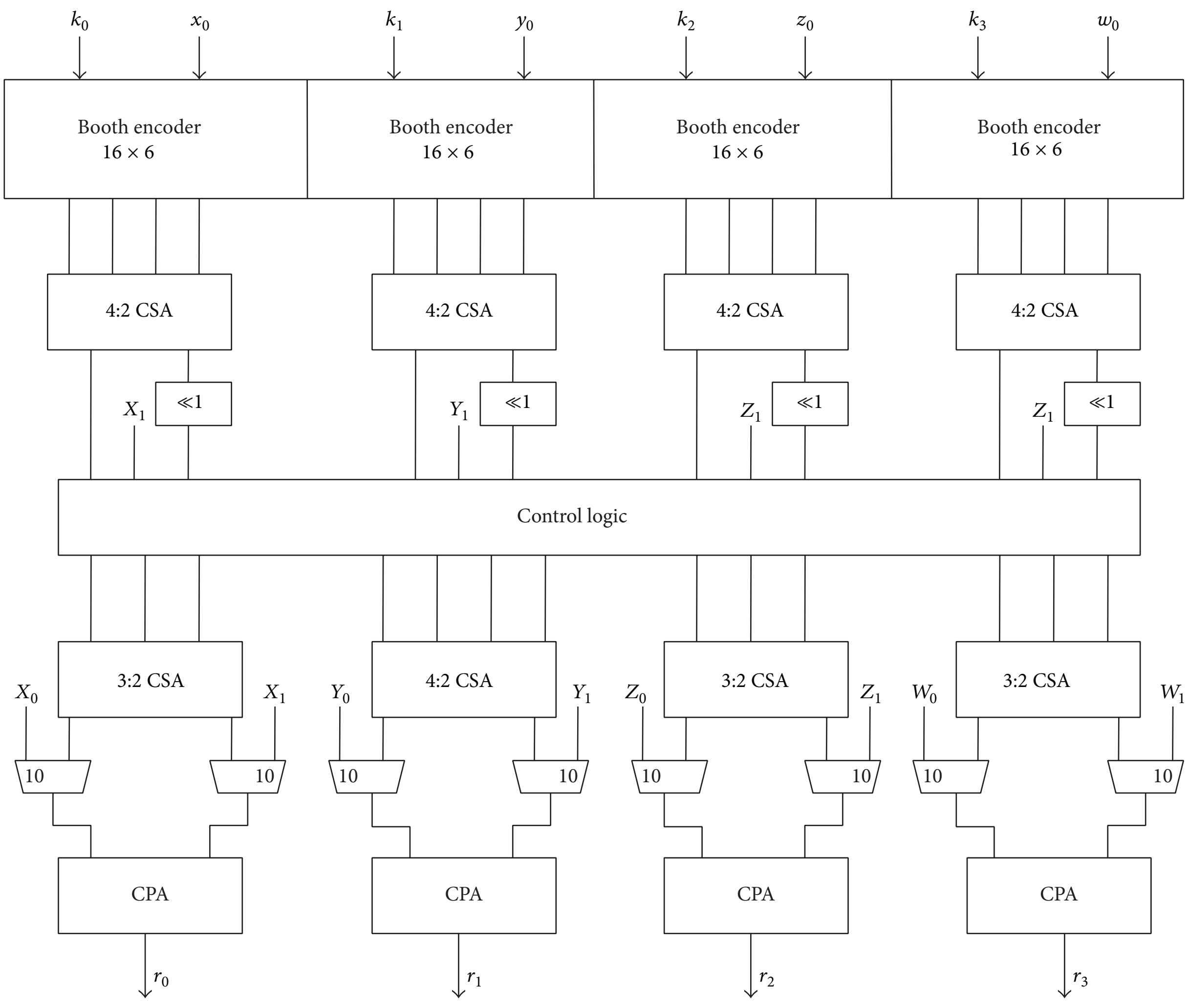

FIGURE 3: Architecture of LAU.

3.1. Computational Unit. In this unit, the mantissa in the logarithmic number is computed. The mantissa of the input is processed by the EBM as explained above. According to the value of the input mantissa, the corresponding equations are implemented and the logarithmic mantissa is obtained.

3.2. Combiner Block. As mentioned earlier, the exponent and mantissa parts are computed separately and combined at the final stage. In this block, the exponent bits of the logarithmic number are computed taking into consideration the sign bit. Depending on the sign bit, either of the two operations is executed. In the positive way, the exponent bits are simply obtained by subtracting the base of IEEE 754 format (base = 15) whereas in the negative way, 1's complement of the positive logarithmic exponent is considered as the final exponent. Since 1's complement is considered for the exponent, 2's complement of mantissa is considered to correct the consistency of the answer.

4. Logarithmic Arithmetic Unit

Figure 3 shows the block diagram architecture for the logarithmic arithmetic unit (LAU). This block is capable of performing three different operations like four multiplications $x_i y_i$ corresponding to the dot product $\sum_{i=1}^{4} x_i y_i$, four multiplications that correspond to the Taylor series expansion $\sum_{i=1}^{4} a_i x^{k_i}$ and one powering operation y^x as four additions of $x_i + y_i$, four MAC operations of $\log a_i + k_i * \log x$, and one multiplication as $x * \log y$, respectively. The word sizes of k_i and x_i are, respectively, 6-bits and 16-bits. For powering operation of x^y, the range of the exponent is extended to 12-bits and its output will be obtained in the channel r_1. As shown in Figure 3, the multiplication required for the MAC and powering operations are performed by using a radix-4 booth multiplier of 16 × 6 input. A series of carry save adders (CSA) and carry propagate adders (CPA) are used to sum up the partial products obtained after multiplication. The control logic block is used to determine which operation is to be performed in the LAU and is responsible for routing of the signals and inputs to the corresponding adder blocks. For this end, a global control signal is made use of. If this control is 00, powering operation is performed, if 01, the Taylor series expansion is to be done, and if 10, the dot product operation is carried out.

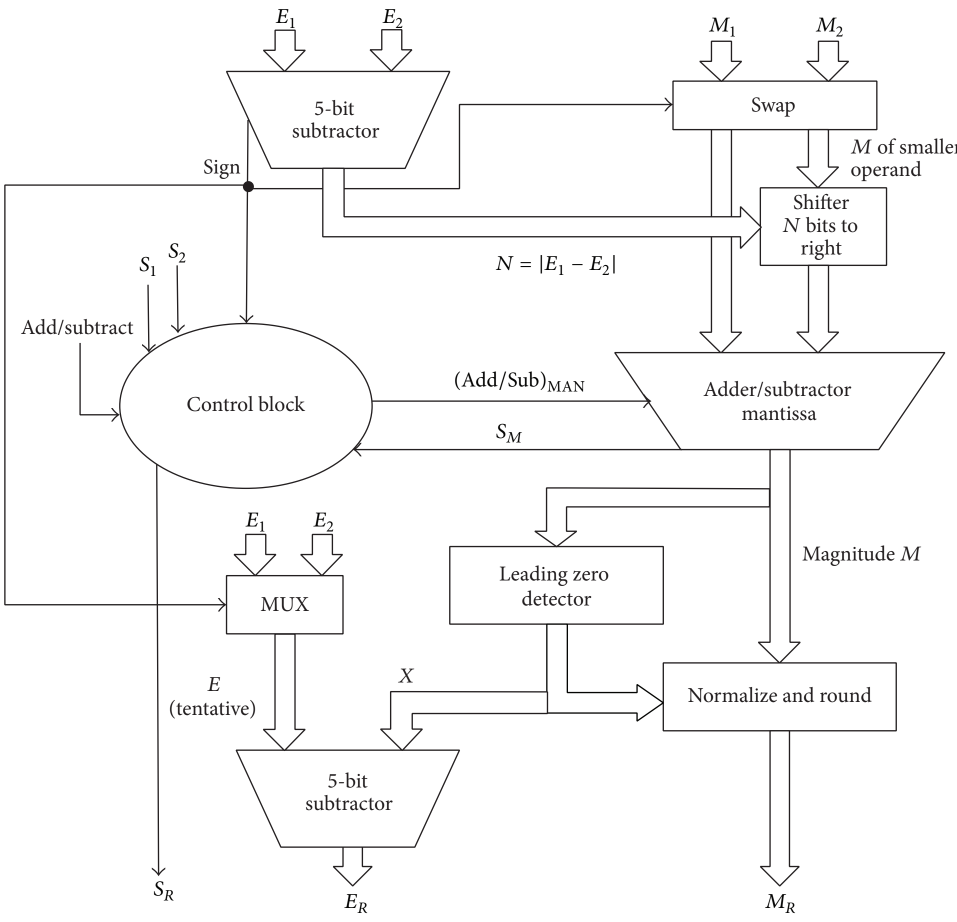

Figure 4: FAU architecture block diagram [10].

5. Floating Point Arithmetic Unit

Floating point arithmetic unit unpacks the operands in the IEEE 754 format into mantissa and exponent to carry out the computation of the two, independent of each other. At the end, it packs the resultant mantissa and exponent back to IEEE 754 format. The architecture of the FAU is as shown in Figure 4.

The basic flow of the computation includes setting the tentative resultant exponent as the larger exponent of the two operands and shifting right the mantissa of the smaller operand by the difference between the two exponents. The extracted exponents are subtracted and the sign bit of the operation is used to determine which of the two exponents is to be set as tentative exponent of the result through a multiplexer. The swap block determines the mantissa to be shifted right according to the sign bit from the subtractor. The shifter right shifts the mantissa of the smaller operand by the amount equal to the difference of the extracted exponents. The control logic determines which operation has to be performed on the mantissa and the sign bit of the result depending on the sign bit of the two operands based on the truth tables shown in Table 5. When the exponent of the two operands is equal the sign of the operation performed on the mantissa is used to determine the sign of the result. The final result is normalized by detecting the number of leading zeros and then shifting the mantissa accordingly to obtain the resultant mantissa. The number of leadings is subtracted from the tentative exponent to obtain the final resultant exponent.

6. Results and Discussion

This work has been implemented in TSMC 180 nm CMOS process technology. The RTL synthesis was performed using Cadence RTL Compiler and the physical design of gate level netlist (GLN) to Gerber Data Stream Information Interchange (GDSII) was carried out using Cadence SoCEn-counter. Table 6 reports the area occupied, number of cells used, and percentage of occupancy of the units of SFU. Table 7 reports the area of logic elements, inverters, sequential elements, and clock gating elements. The total power of SFU as well as the individual blocks is as shown in Table 8.

Tables 9 and 10 illustrate the comparison with respect to area, power, and frequency between this work and the work presented in [9]. It is observed that there is approximately 68% reduction in area. A higher frequency is achieved at the cost of slight increase in power.

Table 5: FAU control logic truth tables.

(a) Add/Sub = 0 (subtraction)

Inputs				Outputs
S_1	S_2	S_M	S_R	$(\text{Add/Sub})_{\text{MAN}}$
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	1	0

(b) Add/Sub = 1 (addition)

Inputs				Outputs
S_1	S_2	S_M	S_R	$(\text{Add/Sub})_{\text{MAN}}$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

Table 6: Area report of each unit.

Units	Cell area	Number of cells	Percentage
LAU	43374	3738	50.45%
FAU	4958	396	5.34%
ALOGC	15832	1092	14.73%
LOGC	27760	2224	30.01%
SFU	91768	7450	100%

Table 7: Gate utilization report for each unit.

Units	Area	Percentage
Logic	66156.165	72.1%
Inverter	2318.648	2.5%
Sequential elements	22865.675	24.9%
Clock gating elements	427.745	0.5%
SFU	91768.234	100%

As mentioned in Section 3, due to the linear approximation used in the EBM, there is a trade-off between accuracy and hardware complexity. An error analysis has been carried out to quantify this trade-off. Since we are using IEEE 754 floating point format representation, the coefficients in the linear approximation are quantized to 10 bits which leads to a quantization error. An inherent error also occurs due to approximation which is also considered in the error analysis. Table 11 shows the percentage errors for different ranges of the variable y in Algorithm 1. A detailed analysis on mean square error and its mathematical formulation has been done in [11]. We observe that the percentage error is less than 2% which is acceptable according to the standards. It can also be observed that, in each interval, the maximum error occurs about the mid-point of the interval.

Table 8: Report of the power consumed by each unit.

Units	Leakage power (mW)	Dynamic power (mW)	Total power mW
LAU	0.0980	1.5322	1.6302
FAU	0.0093	0.6138	0.6231
ALOGC	0.0100	0.1658	0.1758
LOGC	0.0087	0.1034	0.1121
SFU	0.2178	3.7786	3.9964

Table 9: Comparison of results in terms of area.

Units	Gates [9]	Gates [this work]	Gate reduction
SFU	23230	7450	67.9%
LAU	5170	3738	27.7%
FAU	3690	396	89.26%
ALOGC	3880	1092	71.85%
LOGC	7830	2224	71.5%

Table 10: Comparison of power and frequency.

Parameter	[9]	This work
Power	3 mW	3.99 mW
Frequency	200 MHz	500 MHz

Table 11: Error analysis.

Range	y at E_{MAX}	Max. error (E_{MAX})	Error %
$0 \le y < (1/4)$	0.125	0.0034	2.000
$(1/4) \le y < (1/2)$	0.370	0.0029	0.638
$(1/2) \le y < (3/4)$	0.620	0.0032	0.459
$(3/4) \le y < 1$	0.870	0.0027	0.296

7. Conclusion

A special function unit using hybrid number system was implemented. To improve the area and power consumed by lookup tables, an equation based method was employed in the logarithmic and antilogarithmic converters. A control logic was used to choose between the logarithmic and binary number systems depending on the operation to be performed. It is reported that there has been a significant improvement in the overall performance of the SFU. The design occupies a die area of 870 μm × 872 μm (Figure 5).

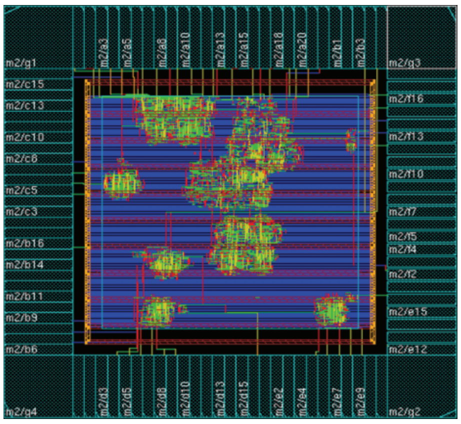

Figure 5: Die image of SFU.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Mirco*, vol. 30, no. 2, pp. 56–69, 2010.

[2] "NVIDIA's Next Generation CUDATM Compute Architecture: Kepler TM GK110," white paper.

[3] B.-G. Nam, H. Kim, and H.-J. Yoo, "A low-power unified arithmetic unit for programmable handheld 3-D graphics systems," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 8, pp. 1767–1778, 2007.

[4] B.-G. Nam, H. Kim, and H.-J. Yoo, "Power and area-efficient unified computation of vector and elementary functions for handheld 3D graphics systems," *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 490–504, 2008.

[5] B.-G. Nam and H.-J. Yoo, "An embedded stream processor core based on logarithmic arithmetic for a low-power 3-D graphics SoC," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 5, pp. 1554–1570, 2009.

[6] C.-H. Yu, K. Chung, D. Kim, S.-H. Kim, and L.-S. Kim, "A 186-Mvertices/s 161-mW floating-point vertex processor with optimized datapath and vertex caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 10, pp. 1369–1382, 2009.

[7] S.-H. Kim, H.-Y. Kim, Y.-J. Kim, K. Chung, D. Kim, and L.-S. Kim, "A 116 fps/74 mW heterogeneous 3D-media processor for 3-D display applications," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 3, pp. 652–667, 2010.

[8] OpenGL ES Specification, 2014, http://www.khronos.org/opengles/.

[9] S.-F. Hsiao, C.-F. Chiu, and C.-S. Wen, "Design of a low-cost floating-point programmable vertex processor for mobile graphics applications based on hybrid number system," in *Proceedings of the IEEE International Conference on Integrated Circuit Design and Technology (ICICDT '11)*, May 2011.

[10] V. Carl Hamacher, Z. Vranesic, and S. Zaky, *Computer Organization and Embedded System*, TMH, 6th edition, 2012.

[11] E. L. Hall, D. D. Lynch, and S. J. Dwyer, "Generation of products and quotients using approximate binary logarithms for digital filtering applications," *IEEE Transactions on Computers*, vol. 19, no. 2, pp. 97–105, 1970.