ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, Vol 8(20), DOI: 10.17485/ijst/2015/v8i20/78367, August 2015 ISSN (Online) : 0974-5645

Floating Point High Performance Low Area SFU

Shibin K. Hassan[1*] and P. Reena Monica[2]

[1]School of Electronics Engineering, VIT University, Chennai Campus, Chennai - 600127, India; shibin hassan2013@vit.ac.in
[2]School of Electrical Engineering, VIT University, Chennai Campus, Chennai - 600127, India; reenamonica@vit.ac.in

Abstract

Objectives: Designing a highly accurate high speed low area Special Function Unit (SFU) is the objective of this work. 32 bit IEEE-754 floating point data format is supported in this system. **Methods:** The SFU implements elementary functions like inverse, exponential, inverse square root, square root and logarithm accurately. The unit can be utilized in programmable graphics processors where high performance and high accuracy evaluation is needed. The coefficients of the elementary functions are optimized by Genetic algorithm. The simulations were carried out in Xilinx EDA tool and MATLAB. Synthesis reports were taken from Cadence RTL compiler. **Findings:** Coefficient optimization and extraction is done using genetic algorithm by doing curve fitting with a second degree polynomial. There is a significant reduction around 40% in the area when constraint piecewise quadratic genetic approximation scheme is used. The number of iterations performed in optimization algorithm is 104. The percentage of error is 0.2578 %. The circuits operated at a frequency of 228MHz and the power dissipation was found to be 3.94 mW. This results in a highly accurate SFU. **Conclusion:** A significant advantage in area when compared to other previous techniques is obtained. The SFU can be utilized in programmable graphics engine.

Keywords: Elementary Functions, Graphics Processor, Special Function Unit (SFU), Single Precision Computation

1. Introduction

Applications like DSP, 2D and 3D graphics, Computer-Aided Design (CAD), virtual reality and three dimensional graphics requires the computation of the basic elementary mathematical functions. Special function unit is a hardware module exclusively for the computation of elementary functions. Graphics Processing Units (GPUs) ought to have the capacity to perform billions of operations every second[1,2]. Modern day applications demand high performance operations of the order of billions of functions per second. So 3-D graphics is an opportunity for the future embedded technology applications or the System-on-Chip (SoC). In a nut shell the graphics processor must have high speed, high efficiency in terms of area and power[3].

In modern GPUs the flexibility and high rendering rates are a necessity for implementing various sophisticated 3-D algorithms. To fulfil this, the GPUs make use of very powerful programming processors, known as shaders. Shaders are proficient to execute the every vertex, every pixel reckonings of the 3-D data pipeline in an exceptionally adaptable way[1-4]. Shaders are fit for executing projects with branches, iterations and element stream control. Shaders computes the elementary functions like inverse, square root, logarithm, exponential, etc along with addition and multiplication.

For proper rendering of million vectors per second, Special Function Units (SFU) for the basic elementary functions are incorporated in advanced GPUs. SFUs for elementary function computation must be of high speed in nature[6-13]. Accuracy prerequisite requirement is a matter of importance for SFUs in addition to speed. Area and power are compromised for accuracy in[8-10] where fixed-point implementations are presented. However floating-point implementations[1] are required in high end graphics systems where maximum visual realism has to be achieved.

* Author for correspondence

The accuracy, programmability and steadily expanding throughput of the present day GPUs powers to utilize GPUs to understand complex calculations ideally equipped for the vector preparing ability of GPUs, for example, database operations, sub-atomic science, processing and complex physical simulations[2,14]. All these features are made possible by the introduction of 32 bit single precision floating point arithmetic within the GPUs. Each one of these highlights became feasible by the start of 32 bit single precision data usage in GPUs. For better support in the scientific computing applications in the forthcoming GPUs, the presence of 64 bit double-precision data has been announced.

The basic elementary calculations can be done by a wide range of algorithms[15,16] in a hardware level. Table based scheme is one of the algorithm that can be used for elementary functions computations. Single precision floating point SFU is the main focus in this work in which table based scheme is the better choice[12,16,17]. Direct table lookup methods require large memory in order to achieve the precision needed for the single precision data which is of floating point in nature. On the contrary table-based methods consume reduced silicon area along with the achievement of high speed of operation.

Multipartite table methods[18] uses some tables and to approximate function values, a simple multioperand addition can be employed. Multipartite strategy is established upon the first order approximation which is piecewise linear in nature. Multipartite strategies are in light of first request piecewise direct close estimation. The interim in which every elementary function must be processed is separated in subintervals on account of polynomial approximation. Approximation of the function is done in each sub-interval using a polynomial of a particular degree. For each subinterval, the polynomial coefficients are stored in lookup-table[19]. A very simple arithmetic circuitry is required for approximations of first order. This results in a reduced latency[20,21]. But for floating-point data format, these approaches are not practical because of the large lookup table. Due to the reason that several elementary functions have to be computed in SFU and for each functions lookup tables are replicated, the performances are further reduced. Because of the reason that many basic mathematical functions must be processed in SFU and for every function lookup tables are reproduced, the speed is further decreased.

Piecewise quadratic estimates are inherently more entangled from the purpose of math hardware But it leads to a very critical benefit due to the reduction in the size lookup-table. There are different types of approaches to do the processing of functions. One such approach is based on Chebysev approximation which is proposed in[22]. Segments of nonuniform length is used and investigated in[23]. For the close estimation of very nonlinear operation, the non-uniform division is exceptionally successful. However this method is not effective as it has another disadvantage of the presence of index encoder. Pineiro et al.[16] discusses a methodology where a fused accumulation tree and a squarer unit are utilized. Finite-wordlength effects are taken into consideration in the computation of the polynomial coefficients in this approach. Finite-wordlength impacts are contemplated in the polynomial coefficients processing in this methodology. This approach helps in reducing the coefficient's wordlength for a prescribed accuracy in addition to the benefits of reduced lookup table size and performance. This increases the performance of the circuit also. The approach of Pineiro et al.[16] is utilized in the SFU that supports the computation of elementary functions[12].

A SFU which process floating point data which works on a technique of quadratic approximation which employs Genetic Algorithm is presented in this paper. Genetic algorithm is employed in collecting function coefficients without compromising the accuracy. Exceptional constraints are likewise forced between the polynomial coefficients of contiguous fragments. This helps in decreasing the area by around 40% when compared to[16] without compromising the precision. This technique fetches the advantage of reduced hardware with small lookup tables, superior operating frequency with lesser power consumption. Lesser hardware helps in decreasing the area and increasing the performance of the SFU, which is very critical for the application.

Association of paper is done as takes after. Section 2 contains architecture of the SFU and Section 3 discusses approximation method and genetic algorithm used for the extraction of coefficients. Section 4 discusses the results of the work and section 5 is conclusion.

2. Architecture

For floating-point numbers IEEE-754 data format is used. A number X can be represented in 32 bit word in the following way as

$$X = (-1)^{Sx} \times x \times 2^{Ex}. \quad (1)$$

Sign bit S_x is one. Mantissa x is 23-bit fractional part and normalized in [1, 2). Exponent E_x is 8-bit.

The overall architecture of SFU is given in the Figure

1. Selection word is of 3 bits length which selects the function to be implemented from the list of function as inverse, logarithm, inverse square root, square root and exponential. To compute the result, there are 3 steps to be carried out[16,22].

- Pre computation block
- Computation block
- Post computation block

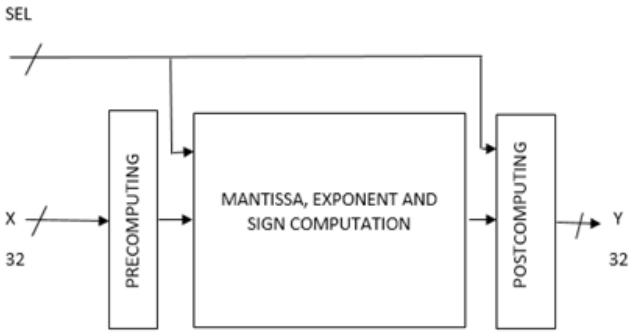

Figure 1. Architecture of SFU.

Pre computation block does the range of data to a foreordained interim. Pre computation piece does the scope of the data to a Mantissa, exponent and sign block performs the function computation under the shortened range. Post computation does the yield reproduction and standardization. The range reduction formulas are given as definitions[29]

Definitions:

$X = (-1)^{Sx} . x . 2^{Ex}$ with: $x \in [1,2)$ Input

$Z = (-1)^{Sz} . z . 2^{Ez}$ Intermediate value

Y = normalize(Z) Normalization operation

$Y = (-1)^{Sz} . y . 2^{Ey}$ with: $y \in [1,2)$ Output

Reciprocal:

$S_z = S_x$; $z = 1/x$; $E_z = -E_x$

Square Root (normalisation is not required)

$S_y = 0$

$Y = \sqrt{x}$ if E_x is even

$= 1/\sqrt{2x}$ if E_x is odd

$E_y = E_x / 2$ if E_x is even

$(E_x - 1)/2$ if E_x is odd

Reciprocal square root:

$S_z = 0$

$Z = 1/\sqrt{x}$ if E_x is even

$= 1/\sqrt{2x}$ if Ex is odd

$E_z = - E_x / 2$ if E_x is even

$= - (E_x - 1)/2$ if E_x is odd

Exponential (normalization is not required):

Denormalize: $X = K + x_{frac}$ K integer;

$x_{frac} \in [0,1)$

$S_y = 0$; $y = 2^{xfrac}$; $E_{y=} K$

Logarithm:

$S_z = 0$ if $E_x \geq 0$; $z = |E_x + \log_2(x)|$; $E_z = 0$

1 if $E_x < 0$

Range diminishment equations allude to a middle quantity Z in the cases of mathematical functions like inverse, inverse root and logarithm in which case mantissa z is not standardized in [1, 2). Post computation block does final normalization.

3. Piecewise Quadratic Genetic Approximation

Genetic Algorithm (GA) is a heuristic approach based on natural selection and the natural concept of survival of fittest. Genetic algorithm is employed here for the purpose of curve fitting using a second degree polynomial. A general flow graph of the iteration in genetic algorithm is given in Figure 2. To implement genetic algorithm for the purpose of curve fitting, the problem needs to be minimised or maximised. Here we utilise genetic algorithm for this. We convert the problem into a GA definable function by converting the real problem in to an objective function.

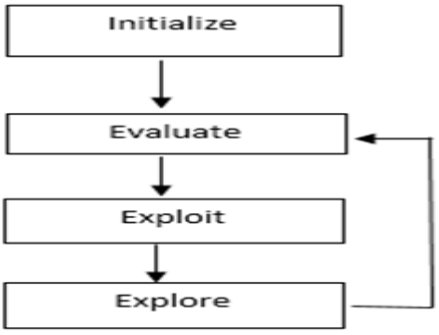

Figure 2. Genetic algorithm iteration.

Let us assume a mathematical function $y = f(x)$ which is between the range [a, b]. We apply the constrained

approximation on this function. To do the approximation, data x is divided in equivalent fragments under a scope [a, b]. So approximation is to be done in each segment.

$$f(x) \approx f_{app}(x) = y_k + m_k(x-x_k) + p_k(x-x_k)^2 \quad (2)$$

Genetic algorithm[27, 28] can be used to obtain the coefficients y_k, m_k, and p_k[12,16]. Approximation using evolutionary method will be helpful in maintaining the accuracy while using reduced table approach. Additionally, to decrease the lookup-table area for a specific recommended exactness, the coefficient-wordlength can be decreased[12,16].

The coefficient y_k require the largest word lengths because of comparative size of linear and other terms in (2). In the work done by Pineiro et al.[16], about 50% of the total memory size is reserved for storing coefficients y_k. In the work done by Pineiro et all, around 50% of the aggregate memory area is held for putting away coefficients yk.

Double contiguous sections are grouped in a fragment pair in this approach. Specify x^* as centrum of segment duo. f (x) can be approximated using the polynomial as given.

$$f(x) \approx g(x)$$
$$= y_{Lk} + m_{Lk}(x-x^*)+p_{Lk}(x-x^*)^2, \text{ for } x<x^*$$
$$y_{Rk} + m_{Rk}(x-x^*)+p_{Rk}(x-x^*)^2, \text{ for } x>x^* \quad (3)$$

The aggregate number of portion sets are demonstrated as M = 2m. Some constraints are imposed for the coefficients in (3). Several combinations of the following constraints were investigated in particular.

$$y_{Lk} = y_{Rk} \quad (4)$$
$$m_{Lk} = m_{Rk} \quad (5)$$
$$p_{Lk} = p_{Rk} \quad (6)$$

All the three constraints (4)-(6) helps us in reducing the memory usage. Case in point, if restraint (4) is utilized, then it is feasible to save one steady coefficient for every subinterval pair where dual coefficients were needed previously. Same is the case with constraints (5) or (6). But out of all the constraints, a larger lookup-table area reduction is possible by implementing the constraint (4), because the particular coefficient needs the maximum word-length. By applying two out of all constraints (4)-(6), it is possible to achieve larger memory reduction. It is clear that if we apply both (4) and (5), then for every segment pair, there will be a requirement of only 4 coefficients in the place of 6. So here the size of two large lookup tables is reduced.

The problem now is, while satisfying the required constraints, coefficients are to be optimized and selected in such a way that the approximation-error must be minimal. The application of constraints will lead to some increment in total approximation-error when compared to the case where no constraints are applied (2). But total error includes the error of coefficients quantization, other arithmetic errors and approximation error. So increasing approximation error alone cannot cause much increase in the total error. So our approach results in a genuine diminishment in the lookup table-size when compared with the proposal employed by Pineiro et al.[16].

The constraint $y_{Lk} = y_{Rk}$ causes some small increment of approximation error. When we apply the constraint $m_{Lk} = m_{Rk}$, error of a little more magnitude is found. Approximation goes bad in the case of $p_{Lk} = p_{Rk}$. So the best choice $y_{Lk} = y_{Rk}$ is imposed with which the smaller lookup table also can be achieved and the better approximation too.

3.1 Precomputing

The architecture diagram of Precomputing is shown in Figure 3. Precomputing does two main tasks. First task is the computation of conversion of excess-127 format to 2's complement. This is because input exponent is in excess-127 form as per the standard we are using for floating-point. Second task is input denormalization which is required by exponential function. Denormalization requires a shifter. This operation is regulated by the input which is exponent.

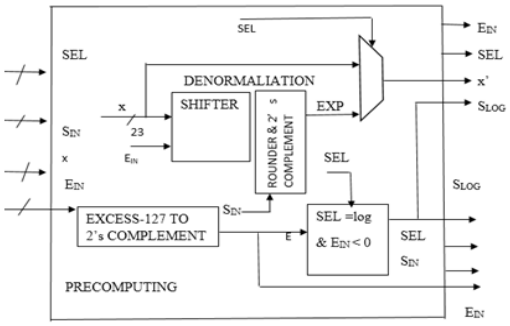

Figure 3. Precomputing.

The denormalized output is kept as 2's complement data. This task is done by 2's complementer. Here the control is done by the sign of input.

When input exponent is negative, the denormalization process causes precision loss. So in order to mitigate this error source, rounding is done. The rounder and the complementer are fused to a unique carry save adder.

The signal S_{LOG}, sign bit is generated by the block precomputation unit. S_{LOG} signal represents the sign,

which is for processing the task logarithm without error. The value S_{LOG} will become '1'when input exponent has negative sign. Here the function selected will be logarithm. S_{LOG} is used in computation block also. So precomputation unit does the conditioning of the signal for the computation in the mantissa and exponent part. Precomputation unit is necessary to process the denormaliation for exponent input and for the generation of S_{LOG} signal in the case of logarithm.

3.2 Computation

In order to realize (3), the mantissa x, which is of 23 bits, is divided into 3 parts. One upper-part, a middle-part and lower-part. The length of upper- part x_U is m bits, central-part x_p is single and the length of lower-part x_L is (23 – m -1) bits:

$$x=x_{22,}x_{21},\ldots,x_{22-m+1},x_{22-m},x_{22-m-\ldots}.,x_0 \quad (7)$$

The upper part starts from x_{22} to $x_{22\text{-m+1}}$ and points the segment pair. The content in x_p which is the middle part consists of only x_{22-m}, helps in understanding whether the value present in left or right. If $x_p = 0$ the data comes under left segment. If $x_p = 1$, data comes under right segment. The information $x - x^*$ of equation (3) can be found from the lower-part, x_L,. Right sided segment has positive values of $x - x^*$. The value of $x - x^*$ is same as x_L. The value of $x - x^*$ in the left side is negative. In the left segments the data will be in 2's complement form.

The length of the upper part is m=5 for square root and exponential, so for $x - x^*$, 18 bits are required to represent. In the case of all other functions, the value of m is 6, so part $x - x^*$ is sign extended to 18 bits from 17.

Internal architecture of the computation block is given in Figure 4. The function to be operated is selected as per the select data, SEL. The cases of odd and even data are differentiated with the help of LSB of exponent part of the data. This is required in the calculation of inverse and inverse square root.

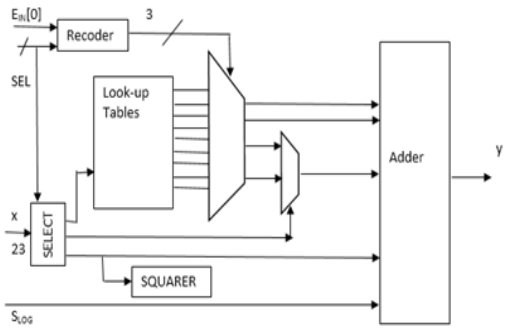

Figure 4. Computation block.

As per the equation (3), every elementary implemented function needs 4 ROMs, pointed by the upper part x_L. Constant and linear coefficients are stored in 2 ROMs and the other 2 ROMs stores quadratic-coefficients for the both side segments. Quadratic coefficient data is selected by x_p. The squarer output has a wordlength of 23 bit.

3.3 Postcomputing

The function of postcomputing unit is to do the normalization of result. The output data is to be in the IEEE-754 format. Figure 5 shows the postprocessing block. The leading-zero finder is one component of the postcomputing block. There is a shifter for the normalization of mantissa is also present.

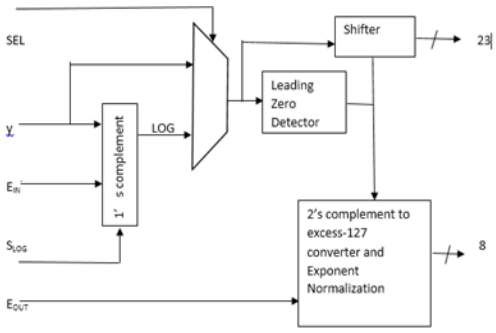

Figure 5. Postcomputing.

The output exponent is in 2's complement form. Since the exponent of single-precision IEEE-754 information is in excess-127 form, exponent is to be converted to that. If the sign of the result is negative, a change of sign is to be done in order to do the logarithmic calculation, which is done in postcomputing.

4. Results

Coefficient optimization and extraction is done using genetic algorithm in MATLAB by doing curve fitting with a second degree polynomial. The architecture design and simulation is done in using Xilinx EDA. The details regarding area, power consumption and operating frequency is extracted using Cadence tool. The output waveform of one of the elementary functions is given in Figure 6. Area, frequency and power details are given in Table 1 and 2 respectively. Comparison of area is given in the Table 3. There is a significant reduction around 40% in the area when constraint piecewise quadratic genetic approximation scheme is used. Comparison of

area is given in the bar chart in Figure 7. The number of iteration performed in optimization algorithm is 104. The percentage of error is 0.2578 %. The circuits operated at a frequency of 228MHz and the power dissipation is 3.94 mW.

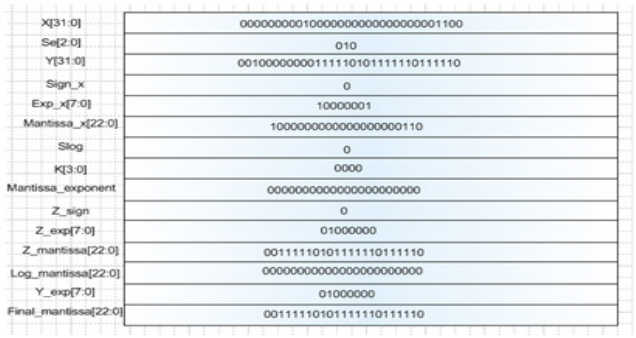

Figure 6. Output waveform.

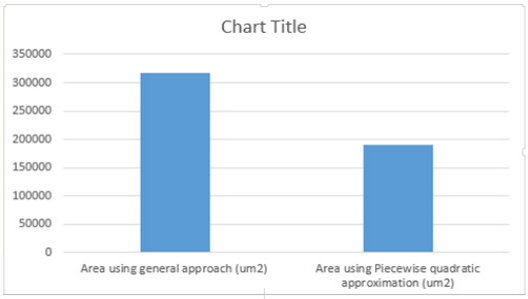

Figure 7. Area comparison.

Table 1. Cells and frequency

Cells	Frequency (MHz)
5219	228

Table 2. Power Consumption

Leakage Power (nw)	Dynamic Power (nw)	Total power (nw)
4570.68	3940967	3945538

Table 3. Area Comparison

Area using general approach (um^2)	Areausing Piecewise quadratic approximation (um^2)
317955	190773

5. Conclusion

A highly accurate floating point special-function unit is conveyed here. A special-function unit that supports single-precision IEEE-754 floating point standard which implements exponential, inverse, square root, inverse square root and logarithm function is implemented. The elementary functions are approximated utilizing piecewise quadratic genetic interpolation method. A significant advantage in area when compared to other previous techniques is obtained. The SFU can be utilized in programmable graphics engine.

6. References

1. Luebke D, Humphreys G. How GPUs work. IEEE Comp. 2007 Feb; 40(2):96–100.
2. Blythe D. Rise of the graphics processor. Proceedings IEEE. 2008 May; 96(5):761–78.
3. Chang Y, Wei J, Sun J. A high performance, area efficient TTA like vertex shader architecture with optimized floating point arithmetic unit for embedded graphics applications. Micro processors and Micro systems. 2012 Jun; 37(6):725–38.
4. Yu CH, Kim D, Kim L-S. A 33.2 M vertices/sec programmable geometry engine for multimedia embedded system. IEEE Proceedings of International Symposium Circuits Systems (ISCAS 2005); 2005 May 23-26. p. 4574–77.
5. Foley J, vanDam A, Feiner S, Hughes J. Computer graphics: principles and practice in C. 2nd edition. MA: Addison-Wesley. 1995.
6. Imai M, Nagasaki T, Sakamoto J, Takeuchi H, Nagano H, Iwasakii S, Hatakenaka M, Fujita J, Keino K, Motomura T, Ueda T, Niki T, Tomikawa H. A 109.5 mW 1.2 V 600 M texels/s 3-D graphics engine. IEEE Proceedings of International Solid State Circuits Conference (ISSCC 2004); 2004 Feb 15-19. p. 332–3.
7. Kim D. A SoC with 1.3 Gtexels/s 3D graphics full pipeline engine for consumer applications. IEEE Proceedings of International Solid State Circuits Conference (ISSCC 2005); 2005. p. 190–192.
8. Sohn JH, Woo JH, Lee MW, Kim HJ, Woo R, Yoo HJ. A 50 M vertices/s graphics processor with fixed-point programmable vertex shader for mobile applications. IEEE Proceedings of International Solid State Circuits Conference (ISSCC 2005). 2005:192–592.
9. Kim H, Nam B, Sohn J, Woo J, Yoo H. A 231 MHz, 2.18 mW 32-bit logarithmic aritmetic unit for fixed-point 3-D graphics system. IEEE J Solid-State Circuits. 2006 Nov; 41(11): 2373–81.
10. Nam B, Kim H, Yoo H. A low-power unified arithmetic unit for programmable handheld 3-D graphics system. IEEE J Solid-State Circuits. 2007 Aug; 42(8):1767–78.
11. Ide N. 2.44-GFLOPS 300-MHz floating-point vector-processing unit for high-performance 3D graphics computing. IEEE J Solid-State Circuits. 2000 Jul; 35(7):1025–10.
12. Oberman SF, Siu MY. A high-performance area- efficient multifunction interpolator. Proceedings of 17th Symposium on Computer Arithmetic (ARITH17); 2005 Jun 27-29. p. 272–9.

13. Montrym J, Moreton H. The GeForce 6800. IEEE Micro. 2005 Mar; 25(2):41–51.
14. Owens JD, Houston M, Luebke D, Green S, Stone JE, Phillips JC. GPU computing. Proceedings of IEEE. 2008 May; 96(5):879–99.
15. Parhami B. Computer arithmetic: algorithms and hardware designs. London, UK: Oxford University Press. 1999 Aug.
16. Pineiro JA, Oberman SF, Muller JM, Bruguera JD. High-speed function approximation using a minimax quadratic interpolator. IEEE Trans Comp. 2005 Mar; 54(3):304–18.
17. Kim D. AnSoC with 1.3 Gtexels/s 3-D graphics full pipeline for consumer applications. IEEE J Solid-State Circuits. 2006 Jan; 41(1):71–84.
18. De Dinechin F, Tisserand A. Multipartite table methods. IEEE Trans Comput. 2005 Mar; 54(3):319– 30.
19. Lee DU, Gaffar AA, Mencer O, Luk W. Optimizing hardware function evaluation. IEEE Trans Comput. 2005 Dec; 54(12):1520–31.
20. Golden M. A seventh-generation x86 microprocessor. IEEE J Solid-State Circuits. 1999 Nov; 34(11):1466–77.
21. Shin HC, Lee JA, Kim LS. A hardware cost minimized fast phong shader. IEEE Trans Very Large Scale Integr (VLSI) Systems. 2001 Apr; 9(2):297–304.
22. Schulte MJ, Swartzlander EE. Hardware design for exactly rounded elementary functions. IEEE Trans Comput. 1994 Aug; 43(8): 964–73.
23. Nagayama S, Sasao T, Butler JT. Compact numerical function generators based on quadratic approximation: Architecture and synthesis method. IEICE Trans Fundam. 2006 Dec; E89-A(12):3510–18.
24. Kolagotla RK, Griesbach WR, Srinivas HR. VLSI implementation of a 350 MHz 0.35 micron 8 bit merged squarer. Electron Lett. 1998; 34(1):47–8.
25. Wires KE, Shulte MJ, Marquette LP, Balzola P. Combined unsigned and two's complement squarers. Proceedings of Asilomar Conference Signals Systems And Compututers; 1999 Oct 24-27; Pacific Grove, CA: USA; p. 125–19.
26. Strollo AGM, Petra N, De Caro D. Dual-tree error compensation for high performance fixed-width multipliers. IEEE Trans Circuits Syst II Exp Briefs. 2005 Aug; 52(8):501–7.
27. Aigner M, Sír Z, Juttler B. Evolution-based least-squares fitting using Pythagorean hodograph spline curves. 4th International Conference on Geometric Modeling and Processing; 2007 Aug; p. 310–22.
28. Karr CL, Weck B, Massart DL, Vankeerberghen P. Least median squares curve fitting using a genetic algorithm. Engineering Applications of Artificial Intelligence. 1995 Apr; 8(2):177–89.
29. Caro DD, Petra N, Strollo GM. High-performance special function unit for programmable 3-D graphics processors. IEEE Trans Circuits and Systems. 2009 Sep; 56(9):1968–78.