An Embedded Stream Processor Core Based on Logarithmic Arithmetic for a Low-Power 3-D Graphics SoC

Byeong-Gyu Nam, *Member, IEEE*, and Hoi-Jun Yoo, *Fellow, IEEE*

***Abstract*—A low-power and high-performance 4-way 32-bit stream processor core is developed for handheld low-power 3-D graphics systems. It contains a floating-point unified matrix, vector, and elementary function unit. By exploiting the logarithmic arithmetic and the proposed adaptive number conversion scheme, a 4-way arithmetic unit achieves a single-cycle throughput for all these operations except for the matrix-vector multiplication that takes 2 cycles per result, which were 4 cycles in conventional way. The processor featured by this functional unit and several proposed architectural schemes including embedded register index calculations, functional unit reconfiguration, and operand forwarding in logarithmic domain achieves 19.1% cycle count reduction for OpenGL transformation and lighting (TnL) operation from the latest work.**

The proposed stream processor core is integrated into a 3-D graphics SoC as a vertex shader to show its effectiveness. The entire SoC is fabricated into a test chip using 1-poly 6-metal 0.18 μm CMOS technology. The 17.2 mm^2 chip contains 1.57 M transistors and 29 kB SRAM. The stream processor core takes 9.7 mm^2 and dissipates 86.8 mW at 200 MHz operating frequency. It shows a peak performance of 141 Mvertices/s for geometry transformation (TFM) and achieves 17.5% performance improvement and 44.7% and 39.4% power and area reductions for the TFM from the latest work.

For power management of the SoC, the chip is divided into the triple power domains separately controlled by dynamic voltage and frequency scaling (DVFS). With this scheme, it shows 52.4 mW power consumption at 60 fps, 50.5% power reduction from the latest work.

***Index Terms*—GPU, handheld systems, logarithmic arithmetic, low-power circuits, power management, stream processor, vertex shader, 3-D computer graphics, 3-D graphics processor.**

I. Introduction

REAL-TIME 3-D graphics have been widely adopted on the handheld devices such as the third-generation (3G) cellular phones and personal digital assistants (PDAs). However, the realization of the 3-D graphics on these power- and area-limited systems has been a challenging issue because of the inherently high computational complexity of the 3-D computer graphics. Consequently, the standard embedded 3-D graphics application programming interfaces (APIs) like OpenGL-ES [1] and Direct3D-Mobile [2] were defined with the fixed function graphics pipelines to provide a higher graphics performance within limited power and area overheads. Furthermore, in the latest versions of these APIs, they introduced the programmability into the graphics pipeline to support the increasing demands for various and more realistic 3-D graphics effects even on the handheld devices. Accordingly, there have been several recent works realizing the programmable 3-D graphics pipeline within limited power and area budgets [3]–[7]. For the programmable graphics, the numerical stream processor cores, known as *shaders*, are incorporated in their pipeline stages. However, the programmable shaders in the pipeline caused large power and area overheads because their datapath cannot be tailored to a specific function and they require additional program and data storages. In this work [8], [9], a power- and area-efficient embedded stream processor core is proposed to reduce the overheads of the programmable graphics pipeline by adopting the logarithmic arithmetic scheme for its datapath design. The proposed stream processor core is integrated into a handheld graphics processing unit (GPU) as its vertex shader in order to show its effectiveness.

A. Backgrounds

The 3-D computer graphics produces virtually realistic scenes by rendering elaborately modeled 3-D objects. The 3-D objects are usually modeled out of several triangles and thereby, the processing of the vertices and pixels for these triangles is the role of the 3-D graphics pipeline. The pipeline consists mainly of the geometry and rendering stages. The geometry stage computes the per-vertex operations related with geometrical manipulations such as coordinate transformation and intensity calculation for each vertex. From this results, the rendering stage sets up the triangle parameters and computes the pixel attributes like color, coordinates, and depth information.

The stream processing model was proposed to exploit localities in signal processing applications, where data are passed directly from the producer to consumer in a pipelined manner [10]. In this model, applications are organized with *streams* and *kernels*; the streams for expressing data and the kernels for computations. A stream is defined as a set of elements of the same type requiring same computation and a kernel is a series of compute-intensive operations to be applied for each element in the stream. A stream processor applies a kernel to all the elements of an input stream and places results on an output stream. In this

Manuscript received December 30, 2007; revised December 17, 2008. Current version published May 01, 2009.

B.-G. Nam was with the Department of Electrical Engineering and Computer Science, KAIST, Daejeon, Korea. He is now with Processor Architecture Laboratory, SOC Platform Team, Samsung Electronics Co., Ltd., Yongin-si, Gyeonggi-do 446-711, Korea (e-mail: byeonggyu.nam@gmail.com).

H.-J. Yoo is with the Department of Electrical Engineering and Computer Science, KAIST, Daejeon 305-701, Korea (e-mail: hjyoo@ee.kaist.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2009.2016698

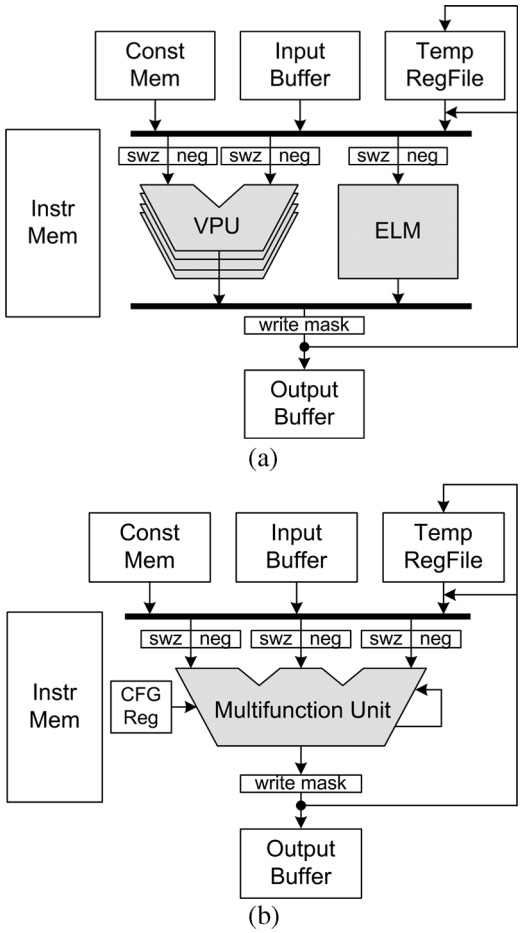


Fig. 1. Vertex shader architecture. (a) Conventional one. (b) Proposed scheme.

model, the stream processor architecture incorporates stream register files in order to exploit the producer-consumer localities since the register files provide much higher bandwidth than on-chip cache or off-chip memory.

In 3-D geometry stage, the *vertex shader*, a kind of a stream processor, is defined to process geometry transformation and lighting kernels on a stream of input vertices and produce an output vertex stream. This output stream is given to the rendering stage and the final output fragment stream is produced to realize 3-D graphics scenes.

Fig. 1(a) shows the conventional architecture of the vertex shader defined in [11]. It incorporates the stream register files for input and output streams of vertices and the vector and elementary function units to support the operations for compute-intensive vertex shading kernels. In addition, it includes the temporary register file to preserve the intermediate results from the processing of vertex shading kernels and the constant memory to store the coefficients required for the processing.

There have been several works on the vertex shader designs for low-power 3-D graphics applications [3], [5], [7]. A vertex shader based on fixed-point (FXP) arithmetic was proposed in [5]. It adopted the fixed-point arithmetic for the power- and area-efficient design. However, it suffered from the limited dynamic range of the fixed-point arithmetic. Although the vertex shaders in [6] and [7] were based on the floating-point arithmetic, [6] incorporated separated vector and elementary function units and [7] used 16 multiply-and-accumulate (MAC) units, which resulted in significant area and power overheads for resource limited handheld systems.

B. Overview

The proposed stream processor exploits the logarithmic arithmetic scheme to reduce the power and area overheads for the programmable 3-D graphics pipeline. Even though the logarithmic arithmetic carries certain amount of computation errors, we adopt it for our processor design because the handheld systems with limited display size can tolerate a reasonable amount of computation errors. Based on the logarithmic arithmetic and the proposed adaptive number conversion scheme, the matrix, vector, and elementary functions are unified into a single 4-way multifunction unit. The unit achieves a single cycle throughput for the most of the operations except for the matrix-vector multiplication which takes 2 cycles for a result. Compared with the conventional scheme, our processor adopts novel architectural features depicted in Fig. 1(b) to fully exploit its novel arithmetic unit. It supports operand forwarding in the logarithmic domain to reduce the pipeline latency and the computation errors. It also incorporates the datapath reconfiguration scheme for the multifunction unit to expand its operation set by allowing user-defined operations. In addition, it allows the register index calculations for the floating-point operands to be embedded into floating-point instructions for the reduced cycles per instruction (CPI). Based on these, our stream processor core shows 17.5% performance improvement for the 3-D geometry transformation, while reducing 44.7% and 39.4% power and area overheads, respectively, compared with the latest work [7].

The proposed stream processor is integrated into a GPU to show its effectiveness as a vertex shader. The 3-D graphics SoC including a RISC processor, a vertex shader, and a rendering engine is divided into triple power domains with dynamic voltage and frequency scaling (DVFS) scheme, which results in 50.5% power reduction from the latest work [7].

This paper is organized as follows. The datapath design using our proposed arithmetic scheme will be discussed in Section II, and the architecture of the proposed stream processor will be covered in Section III. The performance evaluation for the proposed processor will follow in Section IV. The integration into a handheld GPU and the implementation results will be presented in Section V and finally, we will summarize our work in Section VI.

II. Datapath Design

As shown in Fig. 2, the matrix-vector multiplication takes the largest operation cycles in OpenGL transformation (TFM) and transformation-and-lighting (TnL) kernels. Therefore, the optimization of the matrix-vector multiplication is critical to improve the geometry processing performance, and thus the datapath design for our processor is focused on it. There was a prior work optimizing the matrix-vector multiplication [7]. However, it incorporated 16 MAC units and required special operand distribution logic for them to achieve a single-cycle throughput for matrix- vector multiplication, which resulted in large area overhead.

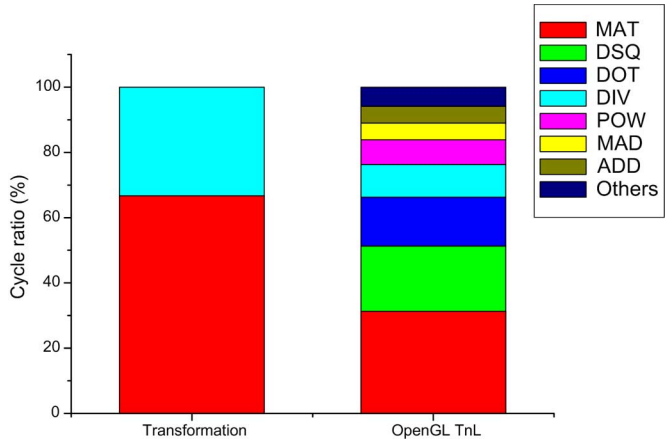
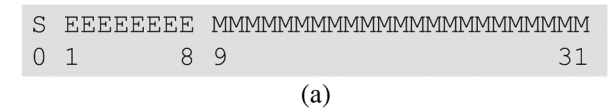


Fig. 2. Operation cycle breakdown for vertex shader kernels.



(a)

(b)

Fig. 3. Number formats for the HNS. (a) Format for floating-point numbers. (b) Format for logarithmic numbers.

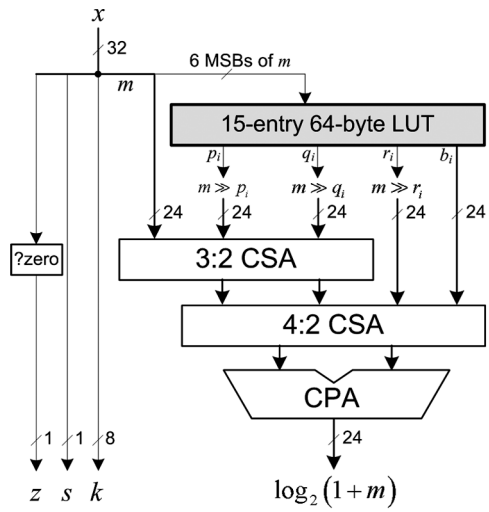
In this work, a high-throughput 4-way floating-point functional unit is proposed to perform the matrix-vector multiplication in every 2 cycles, which were 4 cycles in conventional ways [3]–[6], and achieve a single-cycle throughput for the vector and elementary functions. The unit exploits a novel implementation of the floating-point operations such that it works on the floating-point input and output data while performing the internal arithmetic in logarithmic domain to reduce its hardware complexities. This 4-way unit supports the same matrix-vector multiplication performance with the previous work [7] despite half the throughput since its operating frequency can be doubled by its simple hardware exploiting the logarithmic arithmetic.

A. Number System

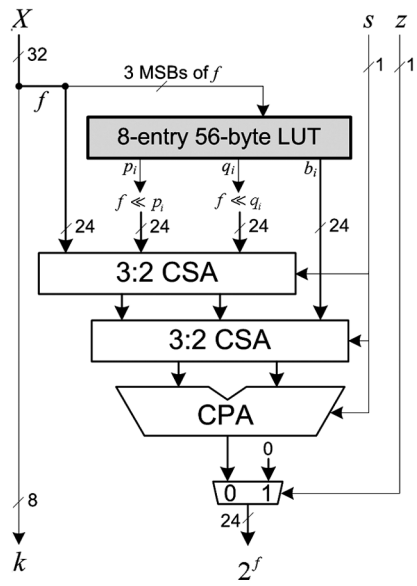
The proposed arithmetic unit is based on the hybrid number system (HNS) [12] of floating-point (FLP) and logarithmic numbers (LNS). In this approach, operations are processed in the LNS, where they become simple, except for the addition and subtraction, which become nonlinear and thus are carried out in the FLP.

1) Number Formats: Fig. 3 shows the number formats of the FLP and LNS used in this work. The FLP format is the IEEE single-precision FLP standard representation that consists of 1-bit sign (S), 8-bit exponent (E), and 23-bit mantissa (M). The most significant bit (MSB) of the mantissa is always 1, which is the hidden bit.

The LNS format is defined with 34 bits with 1-bit zero (Z), 1-bit sign (S), 8-bit integer (K), and the 24-bit fraction (F). The Z and S bits are used to indicate whether the corresponding FLP value is zero or negative since these values have no counter parts from logarithm. The 24-bit fraction part is defined for direct conversion from 24-bit mantissa including the hidden MSB of the FLP format.



(a)



(b)

Fig. 4. Number converters. (a) Logarithmic converter. (b) Antilogarithmic converter.

2) Number Converters: The logarithmic and antilogarithmic converters between the FLP and LNS are required for the HNS. For the 32-bit FLP input $x = (-1)^s 2^{(e-127)}(1+m)$, its logarithmic number representation is $\log_2 x = \overline{z} \cdot (-1)^s \cdot [k + \log_2(1+m)]$, where $k = e - 127$ is integer and $\log_2(1+m)$ is fractional. In the logarithmic converter (LOGC), the nonlinear term $\log_2(1+m)$ is approximated using piecewise linear expressions as $\log_2(1+m) \approx a_i \cdot m + b_i$, where a_i and b_i are the approximation coefficients for each approximation region i. The multiplication of $a_i \cdot m$ is implemented with shifts and additions as $m \gg p_i + m \gg q_i + m \gg r_i$. The structure of the LOGC is shown in Fig. 4(a). The LOGC achieves maximum 0.41% conversion error using 15 approximation regions.

For antilogarithmic conversion, the FLP counterpart of the logarithmic number X can be represented as $2^X = 2^{k+f} = 2^k \cdot 2^f$. In the antilogarithmic converter (ALOGC), the integer

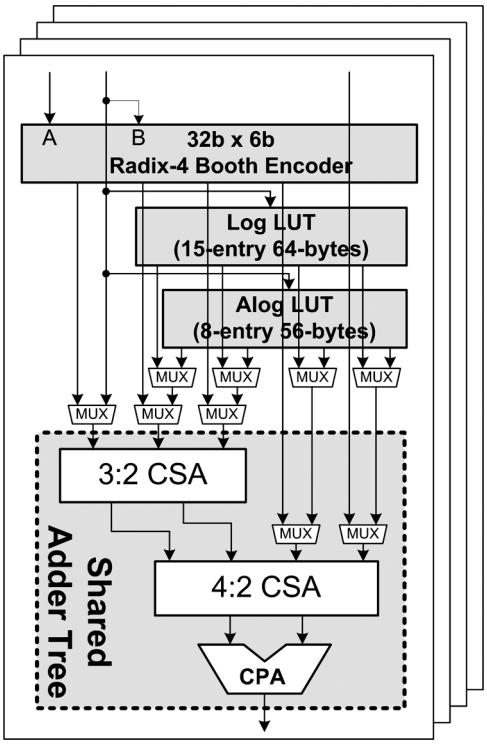


Fig. 5. Programmable multiplication scheme.

part k directly becomes the exponent of the FLP number and the nonlinear term 2^f is also approximated by the piecewise linear expressions as $2^f \approx a_i \cdot f + b_i$, where a_i and b_i are the approximation coefficients defined for each approximation region i. The multiplication of $a_i \cdot f$ is also implemented with shifts and additions as $f \gg p_i + f \gg q_i$. The structure of the ALOGC is shown in Fig. 4(b). The ALOGC achieves maximum 0.08% conversion error using 8 approximation regions.

3) Adaptive Number Conversion: In previous work [13], 8 LOGCs and a Booth multiplier (BMUL) were incorporated in the first and second stages of the pipeline, respectively, to implement the vector and elementary functions. However, the elementary functions required a BMUL with only 1 LOGC, while the vector operations needed 8 LOGCs without a BMUL. Thus, in that approach, the BMUL was a waste for the vector operations and 7 LOGCs were useless for the elementary functions. In order to avoid these redundancies, we propose an adaptive number conversion (ANC) scheme, which divides the 8 LOGCs into 2 groups and places 4 in E1 and the other 4 in E2, and only converts 4 FLP operands in E1 and the other 4 are converted in E2 only if the operation is a vector operation. Thus, the BMUL in E2 is made into a programmable multiplier (PMUL) that can be programmed into a BMUL or 4 LOGCs, by just adding 64-byte 15-entry LOGC coefficient table to the BMUL and sharing the adder tree composed of carry save adders (CSAs) and a carry propagate adder (CPA), as shown in Fig. 5. The PMUL is also made programmable into 4 ALOGCs for the matrix-vector multiplication by adding 56-byte 8-entry ALOGC coefficient table.

Even though the PMUL introduces the multiplexers (MUXes) to share the adder tree among the 32 b × 24 b BMUL, 4 32 b × 6 b BMUL, 4 LOGC, and 4 ALOGC, it saves area by 38% from the separate implementation of all these units. The additional MUXes in the PMUL increases the timing by 16% from that of BMUL. The area and timing between the PMUL and other units are summarized in Table I.

TABLE I
THE AREA AND TIMING COMPARISON BETWEEN PMUL AND OTHER UNITS

	BMUL (32b×24b)	BMUL (32b×6b)	LOGC	ALOGC	PMUL
Area (gates)	13,405	2,989	2,957	1,810	27,525
Delay (ns) @ 1.8V	3.38	2.18	2.3	1.55	3.92

B. Multifunction Unit

The floating-point datapath used in our processor is described in this section. The proposed unit unifies matrix-vector multiplication (MAT), vector operations (VECs) such as vector multiplication (MUL), division (DIV), square-root (SQRT), multiply-add (MAD), and dot-product (DOT), and elementary functions (ELMs) including power (POW), logarithm (LOG), and trigonometric functions (TRGs) into a single 4-way arithmetic unit based on the HNS. It performs the matrix-vector multiplication in every 2 cycles and achieves a single-cycle throughput for all the other operations.

1) Overall Organization: The functional unit is organized as 4 channels and 5 pipeline stages of E1 through E5 as shown in Fig. 6. The 4 of 32-bit FLP input operands are converted into the logarithmic numbers through the 4 LOGCs in the E1 stage. For the ANC, the E2 stage includes the PMUL, which can be used for one of the BMUL in the logarithmic domain for elementary functions including power and various trigonometric functions, 4 LOGCs for vector operations such as vector multiplication, division, and dot-product or 4 ALOGCs for matrix-vector multiplication.

In this way, the number of LOGCs in E1 stage is reduced to 4, which was 8 in [13]. In E3 stage, 4 adders are provided in logarithmic domain for the vector operations and their operational results are converted into the FLP through the 4 ALOGCs. The FLP programmable adder (PADD) in the E4 stage can be programmed into a 5-input FLP adder tree or a 4-way 2-input single-instruction multiple-data (SIMD) FLP adder according to target operations. The E5 stage provides a SIMD FLP accumulator for the final accumulation required for the matrix-vector multiplication.

2) Matrix-Vector Multiplication: The multiplication of 4 × 4-matrix and 4-element vector is required for 3-D geometry transformations. As shown in (1), it requires 16 multiplications and 12 additions.

$$\begin{bmatrix} c_{00}c_{01}c_{02}c_{03} \\ c_{10}c_{11}c_{12}c_{13} \\ c_{20}c_{21}c_{22}c_{23} \\ c_{30}c_{31}c_{32}c_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} c_{00} \\ c_{10} \\ c_{20} \\ c_{30} \end{bmatrix} x_0 + \begin{bmatrix} c_{01} \\ c_{11} \\ c_{21} \\ c_{31} \end{bmatrix} x_1 + \begin{bmatrix} c_{02} \\ c_{12} \\ c_{22} \\ c_{32} \end{bmatrix} x_2 + \begin{bmatrix} c_{03} \\ c_{13} \\ c_{23} \\ c_{33} \end{bmatrix} x_3 \quad (1)$$

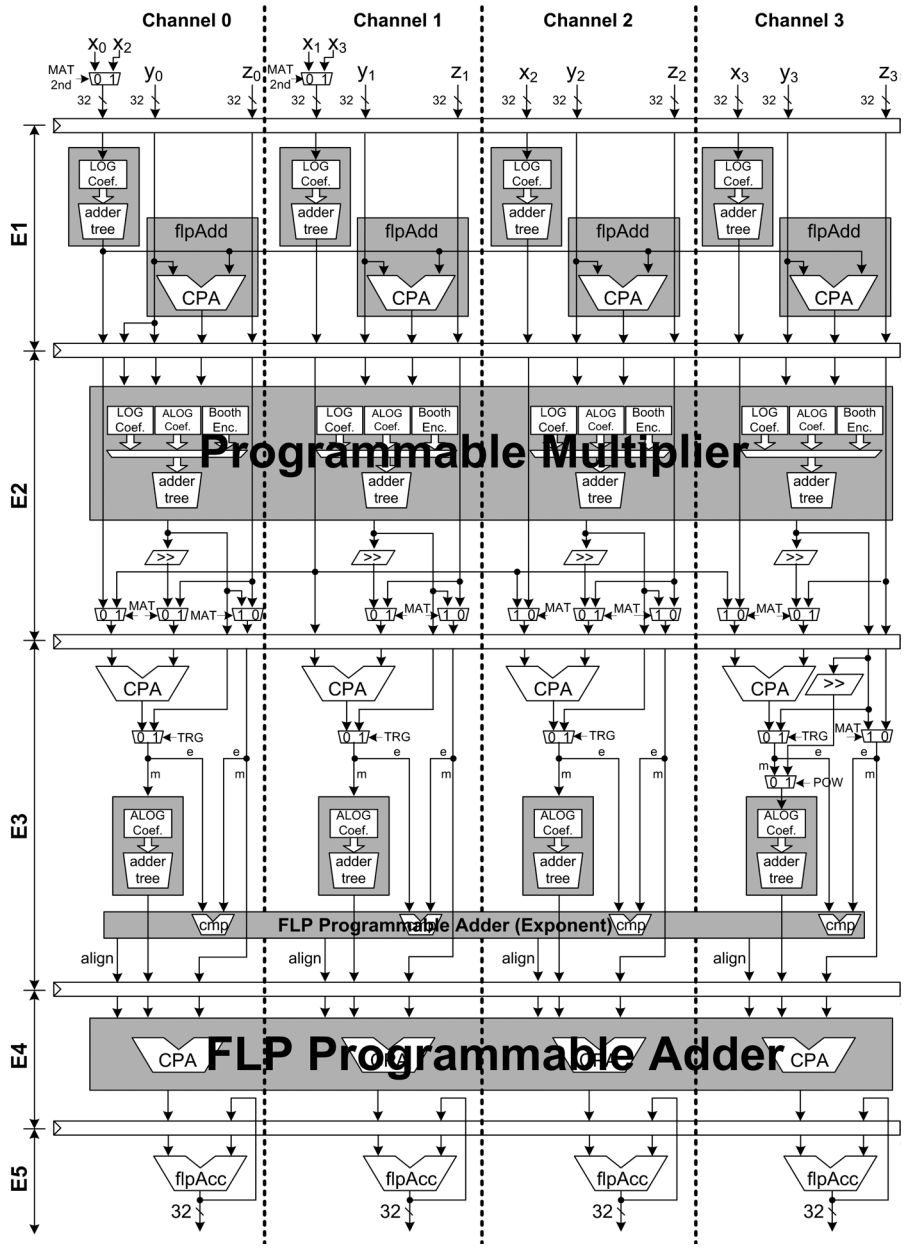


Fig. 6. Floating-point unified arithmetic unit.

In order to implement (1) on our 4-way arithmetic unit, the expression (1) can be converted into an HNS operation as (2), which requires 20 LOGCs, 16 adders, 16 ALOGCs, and 12 FLP adders.

$$\vec{2}^{\left(\begin{bmatrix}\log_2 c_{00}\\ \log_2 c_{10}\\ \log_2 c_{20}\\ \log_2 c_{30}\end{bmatrix}+\log_2 x_0\right)} +\vec{2}^{\left(\begin{bmatrix}\log_2 c_{01}\\ \log_2 c_{11}\\ \log_2 c_{21}\\ \log_2 c_{31}\end{bmatrix}+\log_2 x_1\right)} +\vec{2}^{\left(\begin{bmatrix}\log_2 c_{02}\\ \log_2 c_{12}\\ \log_2 c_{22}\\ \log_2 c_{32}\end{bmatrix}+\log_2 x_2\right)} +\vec{2}^{\left(\begin{bmatrix}\log_2 c_{03}\\ \log_2 c_{13}\\ \log_2 c_{23}\\ \log_2 c_{33}\end{bmatrix}+\log_2 x_3\right)} \quad (2)$$

In the implementation of (2), the log-converted coefficients can be pre-converted into the logarithmic domain and be used as constants during the geometry processing since the coefficients of a geometry transformation matrix are fixed during the processing of a 3-D object. Thus, the matrix-vector multiplication requires only 4 LOGCs for converting the 4-element vector, 16 adders in the logarithmic domain, 16 ALOGCs, and 12 FLP adders. These can be implemented in 2 phases on our 4-way arithmetic unit as illustrated in Fig. 7.

In this way, 8 adders in logarithmic domain and 8 ALOGCs are required per phase. The CPAs in E1 and E3 stages can be used for the 8 adders and the 8 ALOGCs can be obtained from 4 ALOGCs in E2 stage by programming the PMUL into 4 ALOGCs together with the 4 ALOGCs in E3 stage. The 4 multiplication results from the ALOGCs in E2 stage and the other 4 from the E3 stage are added together in the E4 stage by programming the PADD into 4-way SIMD FLP adder to get the first phase result. With the same process repeated, the accumulation with the first phase

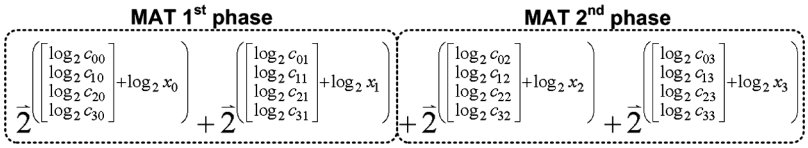


Fig. 7. Two-phase matrix-vector multiplication scheme.

result in E5 stage completes the matrix- vector multiplication. Thus, the matrix-vector multiplication is accomplished in every 2 cycles on this 4-way functional unit.

3) Vector Operations: The vector multiplication, division, square root, and multiply-add can be represented by a single generic operation like in [13] and converted into the HNS as (3).

$$\begin{aligned}&(x_i \otimes y_i^s \oplus z_i)_{i\in\{0,1,2,3\}}\\ &\quad=\left(2^{(\log_2 x_i)\oplus(s\times\log_2 y_i)} \oplus z_i\right)_{i\in\{0,1,2,3\}}\\ &\qquad\text{where}\quad \otimes\in\{\times,\div\},\oplus\in\{+,-\},\ s\in\{0.5,1\}\end{aligned}\tag{3}$$

Since the HNS operation in (3) requires 2 LOGCs for 2 operand conversions per channel, the PMUL is programmed into 4 LOGCs to make the 8 LOGCs for 4 channels together with the 4 LOGCs in E1 stage. This operation achieves the 4-way vectored implementation of the division, the square root, and the division by square root as well without any additional cycles, which is useful for processing vector normalizations. For vector addition and subtraction, the CPAs in E1 stage are also made to operate as the FLP adders. This approach reduces the latencies for vector addition and subtraction since they do not need to wait until E4 stage. The dot-product can also be converted into the HNS as (4).

$$\sum_{i=0}^{i=3} x_i \times y_i = \sum_{i=0}^{i=3} 2^{\log_2 x_i+\log_2 y_i}\tag{4}$$

The HNS operation in (4) is implemented with the vector multiplication followed by the final summation of the product terms. For the final FLP summation, the FLP PADD is programmed into a single 5-input FLP adder tree as depicted in Fig. 8. Since the exponent can be determined directly from the value in logarithmic domain, the exponent logic for the summation tree can proceed in parallel with the ALOGCs in the E3 stage. When the mantissa path is programmed into a summation tree, the shift amount for each mantissa alignment is determined with regard to the maximum exponent to avoid repeated normalization and denormalization circuits in every level of the summation tree. To reduce the critical path of the exponent logic, the duplicated subtractors are used in parallel with the final comparator to compute the alignment values speculatively for both cases of the comparison results. The actual alignment values are determined when the comparison result becomes available.

4) Elementary Functions: Since the power function is converted into the multiplication in the logarithmic domain, it requires multipliers, which can be implemented by programming the PMUL into BMULs. The trigonometric functions can be represented by certain Taylor series expansions and they can be approximated by a single generic power series like (5).

$$\sum_{i=0}^{4}\left[\oplus_i(c_i x^{k_i})\right] = c_0 x^{k_0} + \sum_{i=1}^{4}\left[\oplus_i 2^{(\log_2 c_i + k_i\times\log_2 x)}\right]\tag{5}$$

Since the k_is in (5) are small integer values that can be represented within 6 bits, this power series requires a 4-way 32 b × 6 b multiplier in the logarithmic domain and final FLP summation of the power terms. This can be implemented by programming the PMUL into a 4-way 32 b × 6 b BMUL and the FLP PADD into a 5-input FLP summation tree, as was done for dot-product in previous section.

The power function x^y can also be implemented using a multiplier as (6).

$$x^y = 2^{y\times\log_2 x} = 2^{(2^{e_y}\times m_y)\times\log_2 x} = 2^{(m_y\times\log_2 x)\ll e_y}\tag{6}$$

This expression requires a 32 b × 24 b multiplier in the logarithmic domain, which can be implemented by programming the PMUL into a single 32 b × 24 b BMUL. The result from this BMUL should be shifted by e_y in the logarithmic domain as depicted in the channel 3 of E3 stage in Fig. 6. Fig. 9 illustrates the complete PMUL which can be programmed into 4 LOGCs for vector operations, a single 32 b × 24 b BMUL for power function, a 4-way 32 b × 6 b BMUL for trigonometric functions or 4 ALOGCs for matrix-vector multiplication.

In order to handle the arithmetic exceptions, the Z or S bit of the LNS format is set when the value of an operand is found to be zero or negative. The final computation result is adjusted according to the Z and S bits of the operands as specified in Table II. In our design, the overflow of the arithmetic result is saturated to the maximum representable value and the NaN does not have a specific encoding as in [14], since our target application is focused on the 3-D graphics.

In summary, a power- and area-efficient multifunction unit was proposed for the matrix, vector, and elementary functions. It achieves a single cycle throughput for all the supported operations except for the matrix-vector multiplication which takes 2 cycles per result. The proposed ANC scheme and the adoption of the logarithmic arithmetic enabled the unified implementation of various arithmetic operations on a single 4-way unit and the high-throughput computations.

III. Processor Architecture

In this section, our processor architecture is described in terms of the instruction set architecture (ISA) and microarchitecture. The processor is based on the 4-way vector SIMD architecture and has 5 groups of vector register files: one 16-entry 4-way 8-bit integer register file (INTR) and four 4-way 32-bit FLP register files including 32-entry vector input register (VIR), 32-entry general purpose register (GPR), 256-entry constant memory (CMEM), and 16-entry vector output register (VOR). The INTR is used for integer operations such as register indexing, loop counting, etc. The VIR supplies input data streams to the processor core. The GPR stores temporary processing results and the CMEM supplies constant

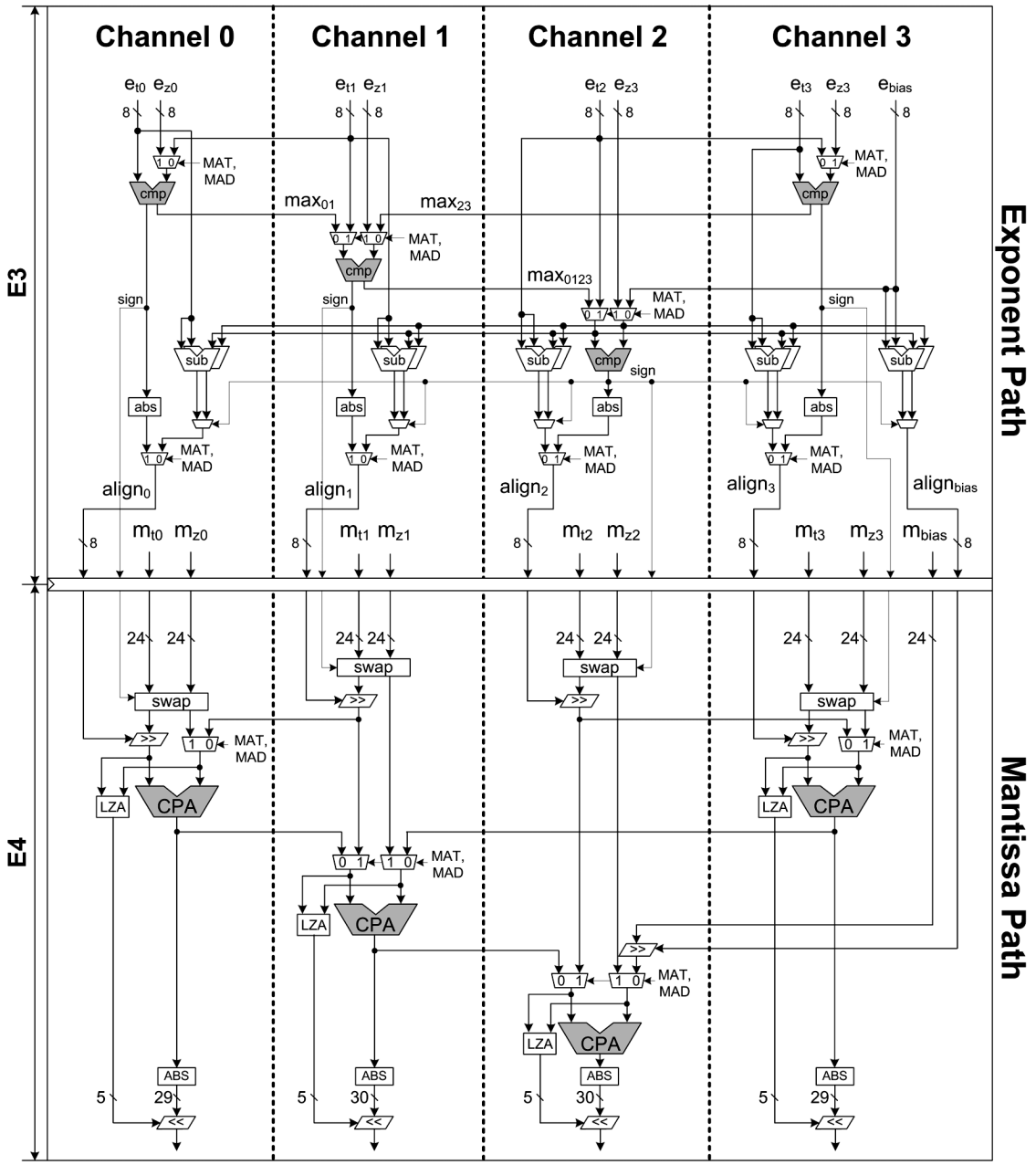


Fig. 8. FLP programmable adder (PADD).

TABLE II
THE EXCEPTION HANDLING SCHEME

Operation		Sign	Zero
MUL		S_x **XOR** S_y	Z_x **OR** Z_y
DIV		S_x **XOR** S_y	Z_x
SQRT		0	Z_x
POW		0	Z_x
TRG	SIN	S_x	Z_x
	COS	0	Z_x
	ATAN	S_x	Z_x

coefficients during a program run. The final results are stored in the VOR so that the following stream kernels start with them.

A. Instruction Set Architecture

1) Dual Instruction Formats: Although the proposed processor is based on a RISC machine architecture, its ISA has two types of instruction formats: 32-bits and 64-bits, optimized for their different usages. In this ISA, the 32-bit format is for the integer (INT) and control instructions, while the 64-bit one is used for the FLP instructions. This separated format rather than using a single 64-bit one for all types of instructions results in the reduced instruction memory (IMEM) size and its power dissipation. Fig. 10 shows the instruction formats of this processor.

2) Embedded Index Calculations: The ISA supports the indexed addressing mode for the operand fetches from the FLP instructions. This addressing mode is useful for the simple calculations of the FLP register indices determined dynamically in runtime. For example, in a loop execution, the register indices

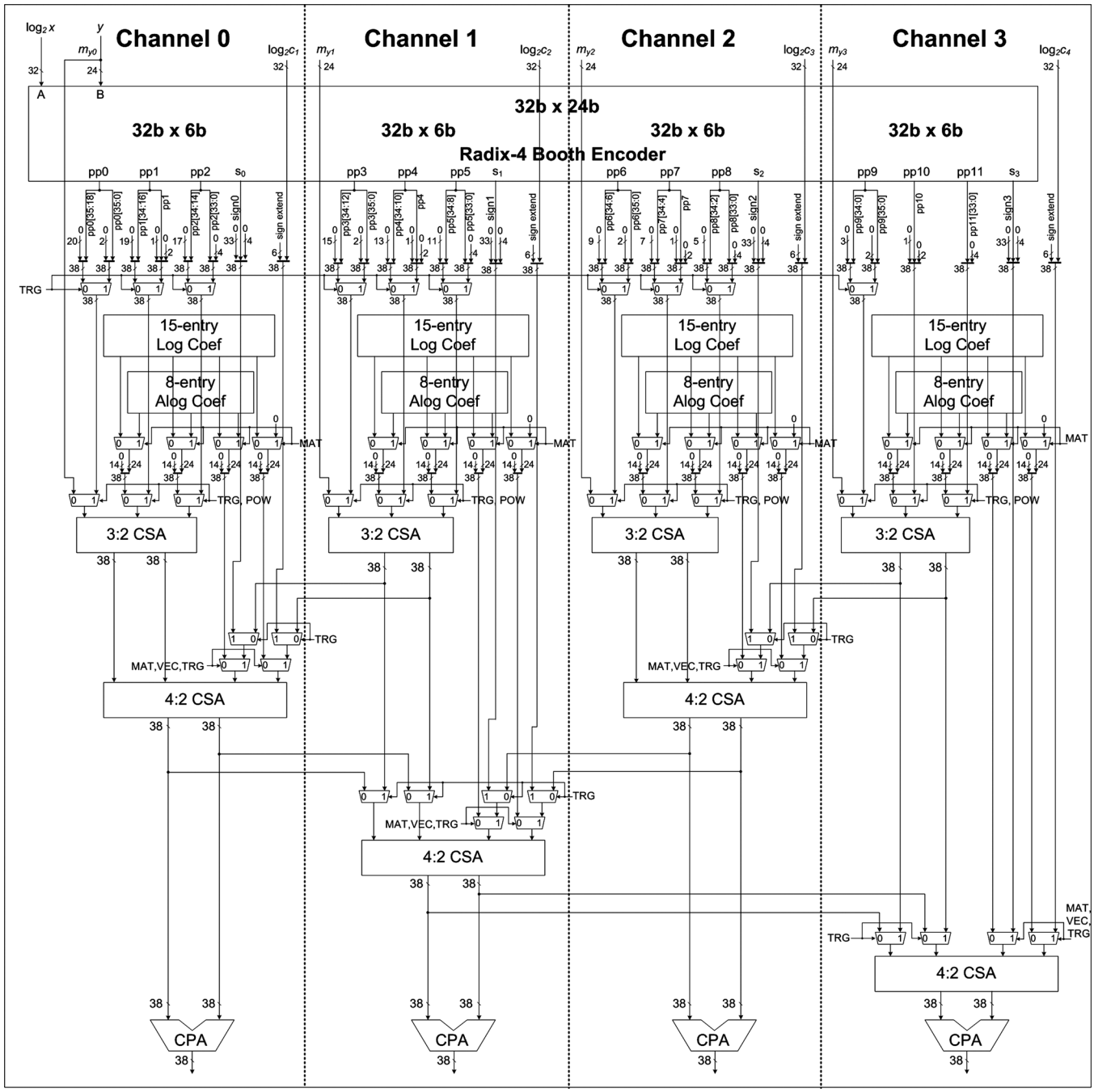


Fig. 9. Complete diagram of programmable multiplier (PMUL).

for FLP operands can be calculated dynamically as a simple function of the loop counter i as follows:

$$\texttt{OP Dst}[\texttt{i}+\texttt{k}_\texttt{D}],\ \texttt{SrcA}[\texttt{i}+\texttt{k}_\texttt{A}],\ \texttt{SrcB}[\texttt{i}+\texttt{k}_\texttt{B}],\ \texttt{SrcC}[\texttt{i}+\texttt{k}_\texttt{C}].$$

Thus, the INT instructions calculating the FLP register indices can be embedded into FLP operand fields of the FLP instructions for efficient index computations, as depicted in Fig. 10. The 'idxD', 'idxA', 'idxB', and 'idxC' fields enable the indexed addressing mode. In this case, index calculations in INT are followed by the FLP computation and are carried out without any influence on pipeline throughput and additional code memory space.

3) Instruction Set Extension: The ISA can be extended by user defined instructions according to the application requirements. This is accomplished by exploiting the reconfigurability of the datapath proposed in Section II. For this reason, the configuration instruction (CFG) is provided as a 4-operand instruction that has a configuration register as its source operand, represented as 'srcD' in Fig. 10(d). This scheme is effective in expanding the limited instruction space of our processor. For example, various trigonometric functions supported in the datapath can be utilized by a single CFG instruction with an appropriate configuration data rather than packing all the trigonometric instructions into the limited instruction space.

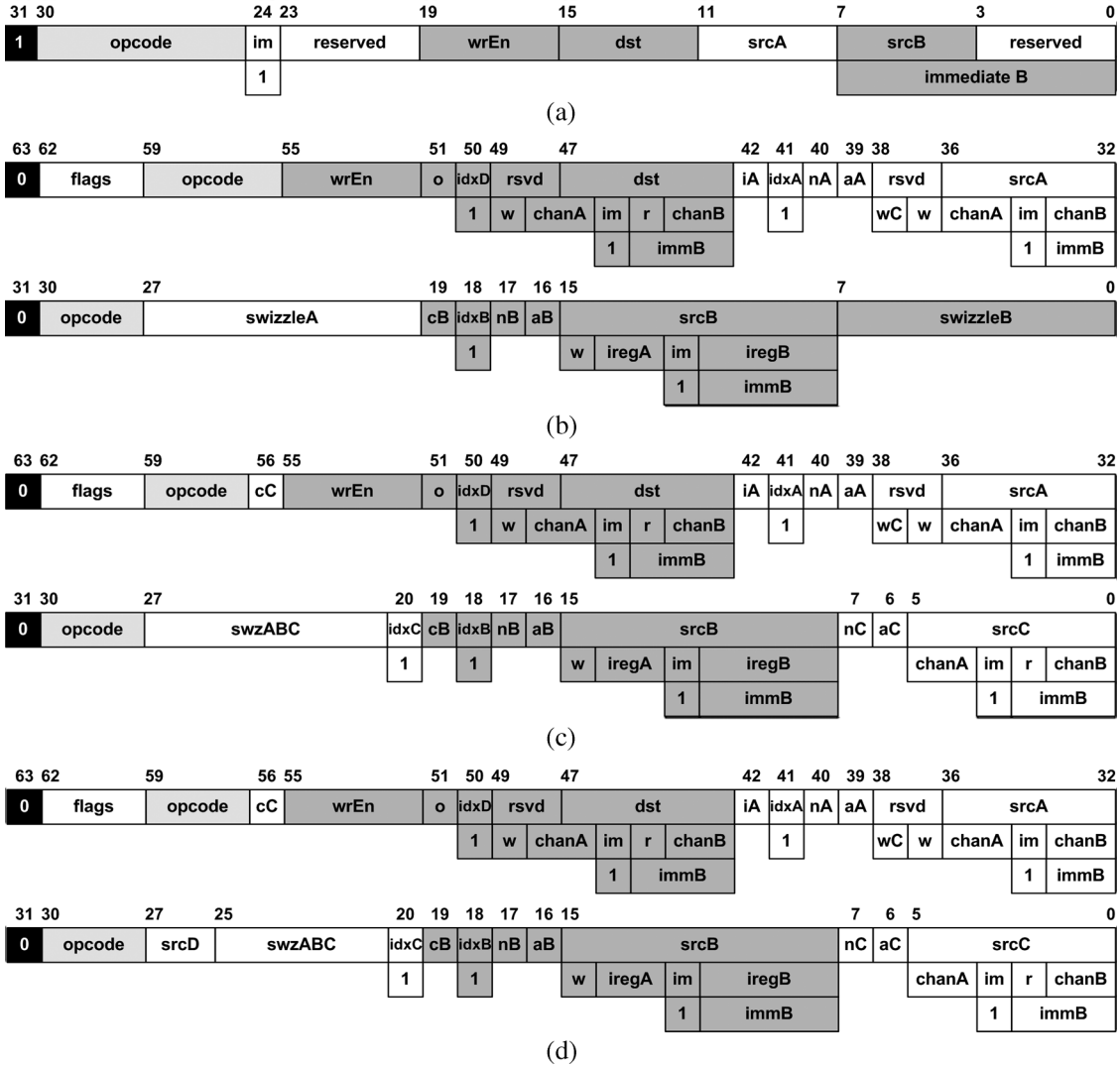


Fig. 10. Instruction formats of the proposed processor. (a) 32-bit INT vector instructions. (b) 64-bit FLP instructions with 2 operands. (c) 64-bit FLP instructions with 3 operands. (d) 64-bit FLP instructions with 4 operands (i.e., CFG instruction).

4) Operand Modifiers: Since all the channel mismatches between the operands and the SIMD hardware should be resolved before processing the data in SIMD architectures, these kinds of pre-processing can severely limit speedup of the SIMD processors [15]. Thus, the swizzling fields for the source operands are packed into our instruction format even if it takes a large bit widths in the instruction format. Likewise, the negation fields like 'nA', 'nB', and 'nC' and fields for absolute values such as 'aA', 'aB', and 'aC' found in Fig. 10 are also included as source modifiers to improve the SIMD performance.

B. Microarchitecture

Fig. 11 shows the microarchitecture of the proposed processor. The FLP operands are fetched from the GPR, VIR or CMEM and the result is written back to the GPR or the VOR. The FLP operands can be swizzled, negated, and converted into the absolute values by the source modifiers. It has a 64-byte INT register file and FLP register files including 512-byte VIR, 512-byte GPR, 4 kB CMEM, and 256-byte VOR. The arithmetic unit proposed in Section II is adopted for the FLP unit of this processor. The data memory (DMEM) is embedded to support the procedure calls in shader programs. The procedure call stacks can be implemented using this memory and thus, the GPR data can be saved in a call stack whenever a procedure call is made. Its 2 kB capacity can provide enough space for deep procedure calls and the internal scratch pad memory as well. For the integer data, parts of the CMEM can be used for preserving them since the CMEM, rather than the DMEM, is directly visible as a storage for the integer path in this microarchitecture.

1) Pipeline Control: The 10-stage pipeline of the proposed processor is depicted in Fig. 12. Since an instruction can be either 32 bits or 64 bits, the MSB of the instruction gives the information of instruction format. The fetch unit always fetches a 64-bit word and makes a 64-bit instruction packet by taking 32 bits from current fetch and padding it with 32-bit zeros for 32-bit instructions. For 64-bit instructions, it takes 32 bits from current and previous fetches, respectively, or the whole 64 bits from current fetch. The finite state machine (FSM) in the instruction decode (ID) stage generates the interlock control signals for multi-cycle operations like the matrix-vector multiplication. All of the dependencies and hazards among the issued in-

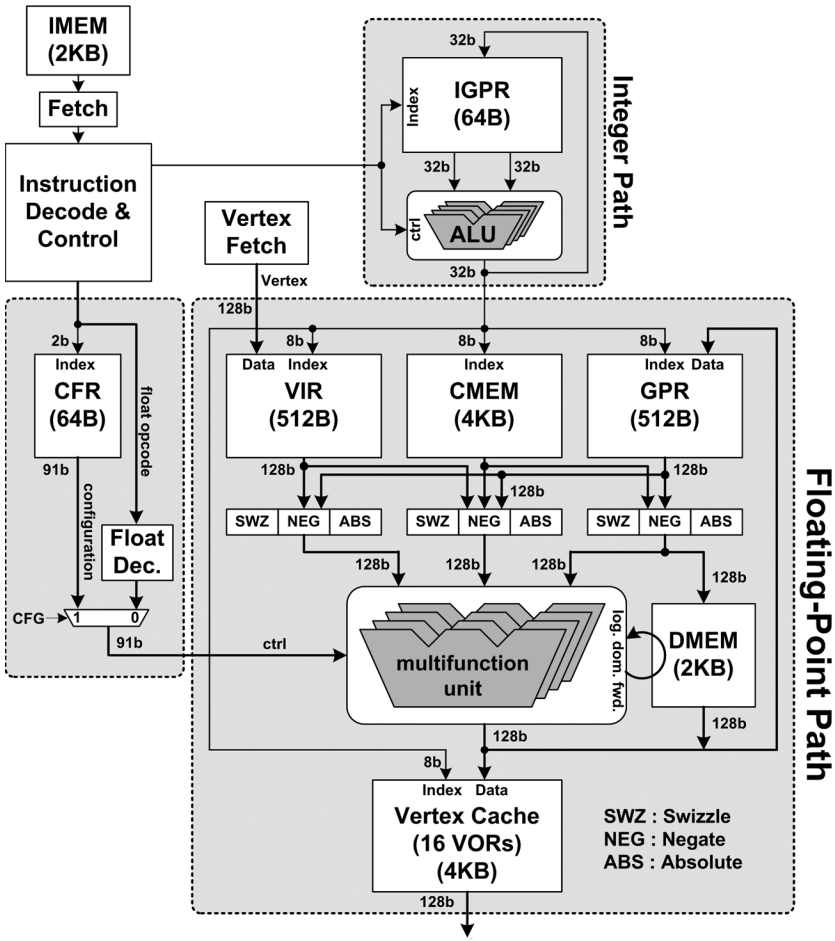


Fig. 11. Microarchitecture of the proposed processor.

structions are identified and scheduled using the register scoreboarding mechanism [6], [16].

2) Pipeline Datapath: This processor has a cascade architecture of 4-way 8-bit INT and 4-way 32-bit FLP datapaths to implement the embedded index calculations of the FLP operands without requiring additional clock cycles for them. For flexible index calculations, this processor includes a 4-way 8-bit SIMD integer ALU and a 64-byte 16-entry integer register file (INTR). The 4 of 8-bit results from this unit can address 3 source operands and 1 destination of an FLP operation.

When an INT multiplication is required for the index calculation, the PMUL in FLP arithmetic unit is programmed into a 32 b $\times$ 24 b integer multiply-add (IMAD) unit. The vector SIMD multiplication in FLP unit is not used for this purpose since the computation errors are not acceptable for the index calculations.

3) Datapath Reconfiguration: The FLP unit in Section II has several MUXes to unify various arithmetic operations into a single arithmetic unit. In this architecture, the FLP unit reveals all of its control signals for the MUXes to the programmer so that arbitrary operations can be implemented on demand by programming the 91-bit control signals. Using this scheme, several optimizations are possible. For instance, the operation of $x^2 + y^2 + z^2$, required for vector normalization, can be programmed using the CFG instruction such that the squares can be implemented by right-shifts of the log-converted values rather than the addition of two identical log-converted values. Thus, the accumulation of the logarithmic conversion errors can be reduced. These kinds of configuration data are stored in the 64-byte 4-entry configuration register file (CFR) and accessed as an operand from the CFG instruction. In consequence, the configuration can be changed in a single clock cycle.

4) Operand Forwarding: The operand forwarding is a way to improve the throughput of a processor pipeline. In this processor, the intermediate results in the logarithmic domain can be directly forwarded to the next instruction or stored into the FLP register file without going through the entire antilogarithmic conversion. For example, as in (7), the consecutive FLP operations without requiring final FLP additions, the antilogarithmic and logarithmic conversions cancel each other. Thus the intermediate logarithmic value from the previous FLP operation can be directly forwarded to the logarithmic domain of the next FLP operation bypassing the antilogarithmic and logarithmic converters of two operations, as illustrated in Fig. 13.

$$\begin{aligned} x \otimes_{\mathrm{FLP}} y \otimes_{\mathrm{FLP}} z &= 2^{(\log_2 x \oplus_{\mathrm{LNS}} \log_2 y)} \otimes_{\mathrm{FLP}} z \\ &= 2^{\left(\log_2 \left(2^{(\log_2 x \oplus_{\mathrm{LNS}} \log_2 y)}\right) \oplus_{\mathrm{LNS}} \log_2 z\right)} \\ &= 2^{(\log_2 x \oplus_{\mathrm{LNS}} \log_2 y \oplus_{\mathrm{LNS}} \log_2 z)} \end{aligned} \tag{7}$$

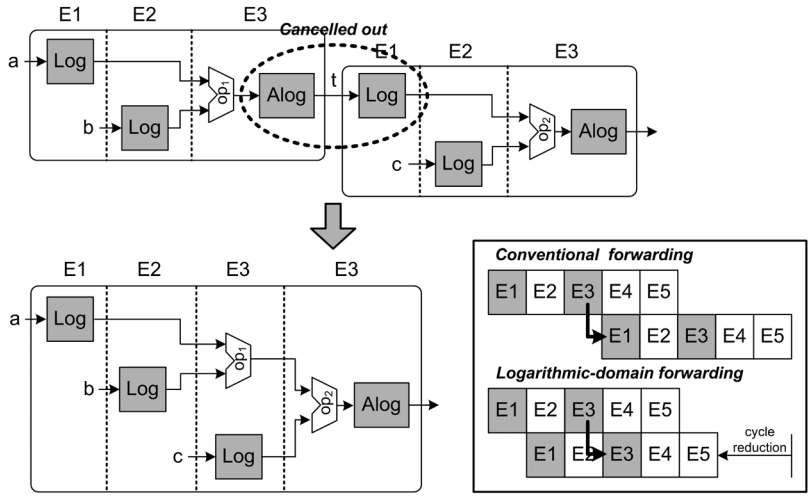


Fig. 13. Operand forwarding in logarithmic domain.

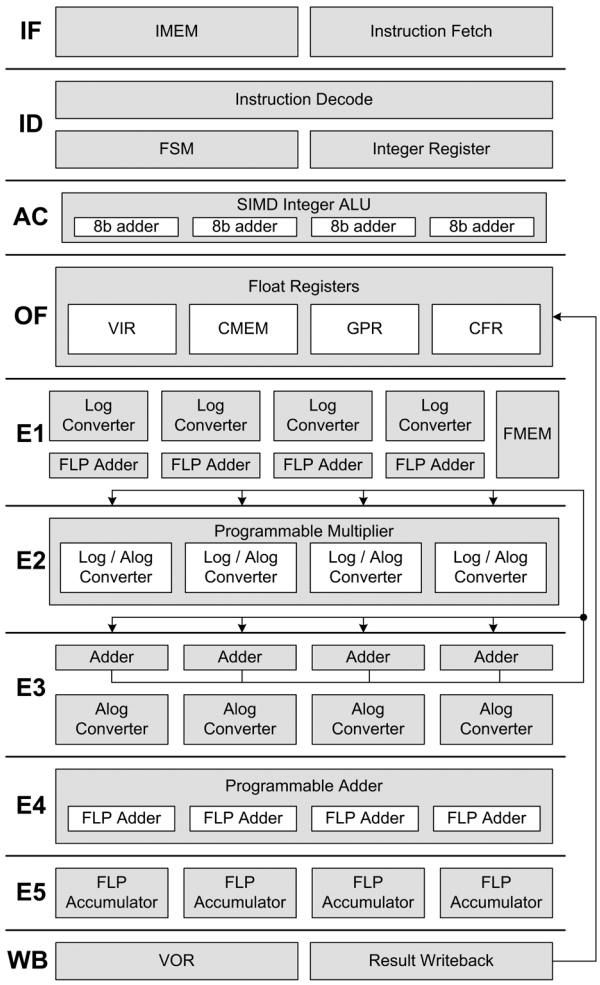


Fig. 12. Pipeline stages of the proposed processor.

This operand forwarding scheme improves the overall pipeline throughput since it finishes the previous operation in advance and starts the next operation immediately. The computation error is also reduced from bypassing the repeated antilogarithmic and logarithmic conversions, which are the sources of the computation errors.

In this section, the architecture of the proposed processor has been described. Its ISA and microarchitecture are co-optimized for the power and area efficiencies. Its ISA supports two types of instruction formats to make better use of the instruction memory space and power budgets. It also supports the embedded index calculations for the FLP register indices determined dynamically at runtime. Its microarchitecture supports the operand forwarding method in the logarithmic domain and the datapath reconfiguration scheme to reduce the latencies and computation errors.

IV. Performance Evaluations

A. Accuracy Evaluation

The errors of the operations are quantified for the worst case conditions as listed in Table III. Each operation is tested for 10,000 vectors randomly selected from [0, 1), the trigonometric functions are tested for the vectors from $[0, \pi/2)$. The operations that consist of the MUL and the ADD, for example, MAT, MAD, and DOT, exhibit the maximum errors at the same point with the MUL since the ADD does not cause the computation error.

Fig. 14 shows the comparison of the scenes from the FLP and HNS libraries, respectively. The test model consists of 5,878 triangles and the screen size is QVGA i.e., 320 $\times$ 240. The test model was rendered from the OpenGL TnL kernel with three light sources to show the accumulated error effects for the mixture of various operations. The test scenes are compared in terms of the peak-signal-to-noise ratio (PSNR) and the result shows that the PSNR of 25.94 dB, which is considered to be acceptable for the wireless applications [17]. Thus, the computation error from our arithmetic unit is within a tolerable range for the handheld systems with limited display size.

B. Performance Comparison

1) Datapath: In Table IV, we quantified the latency of our direct computation approach for the datapath design and compared it with the table-based approach for several graphics kernels. The table-based functional unit, in this comparison, is assumed to have four MAC units to implement the vector opera-

TABLE III
SELECTED FLP ARITHMETIC INSTANCES OF THE PROPOSED PROCESSOR

Mnemonic		Description	Type	Max. Error	Max Error Condition	Latency/ Throughput
MAT		Matrix-vector multiply	(M, V) → V	0.00383	0.686116	6 / 0.5
ADD / SUB		Addition / Subtraction	(V, V) → V	0	-	1 / 1
MUL		Multiply	(V, V) → V	0.00096	0.686116	3 / 1
MAD		Multiply-and-add	(V, V, V) → V	0.00096	0.686116	5 / 1
DOT		Dot product	(V, V) → S	0.00383	0.686116	5 / 1
RCP		Reciprocal	V → V	0.00059	0.686712	3 / 1
SQRT		Square-root	V → V	0.00065	0.687111	3 / 1
POW		Power	(S, S) → (S, S, S, S)	0.00094	0.687604	3 / 1
CFG	SIN	Sine	S → (S, S, S, S)	0.00115	1.374747	5 / 1
	COS	Cosine	S → (S, S, S, S)	0.00220	1.374749	5 / 1
	ATAN	Arctangent	S → (S, S, S, S)	0.05586	1.011957	5 / 1

(a) (b)

Fig. 14. Comparison of test scenes. (a) Scene from FLP library. (b) Scene from HNS library.

TABLE IV
LATENCY COMPARISON BETWEEN TABLE-BASED AND OUR DIRECT COMPUTATION METHODS

Reference	Latency (cycles)			
	Transform	OpenGL TnL	C-T TnL	O-N TnL
Table-based (4-MAC + LUT-based interpolate) [5]	4	69	95	147
This approach	2	43	64	80

TABLE V
COMPARISON WITH OTHER GEOMETRY PROCESSORS

Reference	Performance (Mvertices/s)	Arithmetic scheme	Power (mW)	Area (mm^2)	Freq. (MHz)	Process (μm)
[3]	36	FLP	250	N.A.	400	0.13
[5]	50	FXP	60.3	10.2	200	0.18
[6]	33	FLP	N.A	N.A.	166	0.13
[7]	120	FLP	157	16	100	0.18
This work	141	HNS	86.8	9.7	200	0.18

tions and a lookup table (LUT)-based interpolator for the elementary function interpolations, which is a common approach adopted for graphics processors [6]. The latency and throughput for each operation are assumed to be seven and one, respectively, according to the reported cycle counts for their operations. The tested graphics kernels include the 3-D geometry transformation, full OpenGL TnL [18], Cook-Torrance (C-T) [19], and Oren-Nayar (O-N) [20] lighting models. The C-T model produces improved specular lighting for metals and plastics, which requires several power function evaluations. The O-N model enhances the diffuse lighting model for rough faces and shows the outstanding performance improvement from our processor because it includes several trigonometric function computations which are not directly supported from others [3]–[7]. As shown in Table IV, our approach achieves 50%, 37.6%, 32.6%, and 45.5% cycle count reductions for the each kernel compared with the table-based method.

2) Processor: In Table V, the performance, power, area, clock frequency, and process technology of the proposed logarithmic stream processor are compared with those reported in the previous works. Here, the comparison is made for the 3-D geometry transformation since it is the common test case for

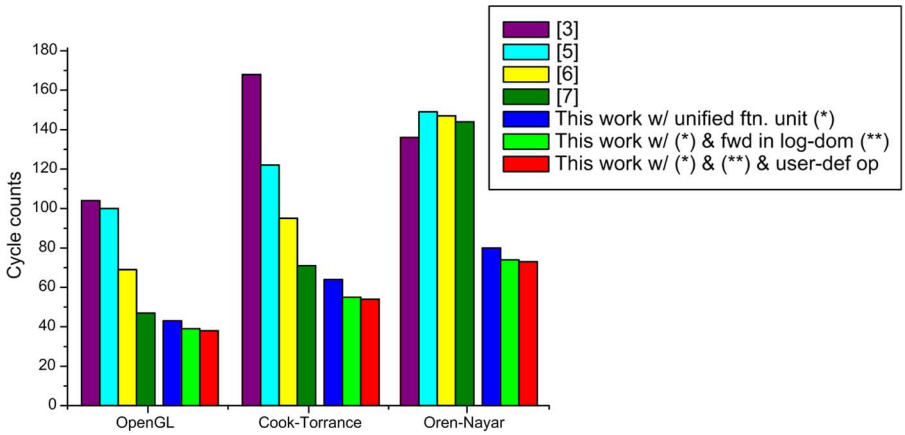


Fig. 15. Cycle count comparison.

which the performance and power dissipation data are reported from the compared works.

The latencies for the full OpenGL TnL, C-T, and O-N kernels are also compared with others in Fig. 15 to show the performance characteristics for more serious graphics kernels. In this comparison, the latencies for other works are estimated from the reported cycle counts of their instruction set or from the description of their pipeline architectures. Our processor achieves 19.1%, 23.9%, and 49.3% latency reductions for the OpenGL TnL, C-T, and O-N kernels, respectively, from the latest work [7].

V. Embedding Into a Graphics SoC

A. Handheld GPU

The proposed processor is integrated into a handheld GPU as its vertex shader [8], [18]. In this section, the architecture, chip implementation, and power management of the handheld GPU are described and compared with related works.

1) SoC Architecture: Fig. 16(a) shows the overall architecture of our handheld GPU. It consists of an ARM10-compatible 32 b RISC processor, a 128 b vertex shader (VS), and a rendering engine (RE) for the processing of application, geometry, and rendering stages, respectively, and it also incorporates three power management units (PMUs) for power management of these modules. The RISC works as the main control processor and runs application programs producing the geometry transformation matrices for 3-D objects. The VS moves the matrix from the matrix FIFO to its CMEM and then performs various geometry operations on fetched vertices by running vertex shading kernels from IMEM. The rendering engine fills up the pixels inside the triangle whose vertices come from the stream cache.

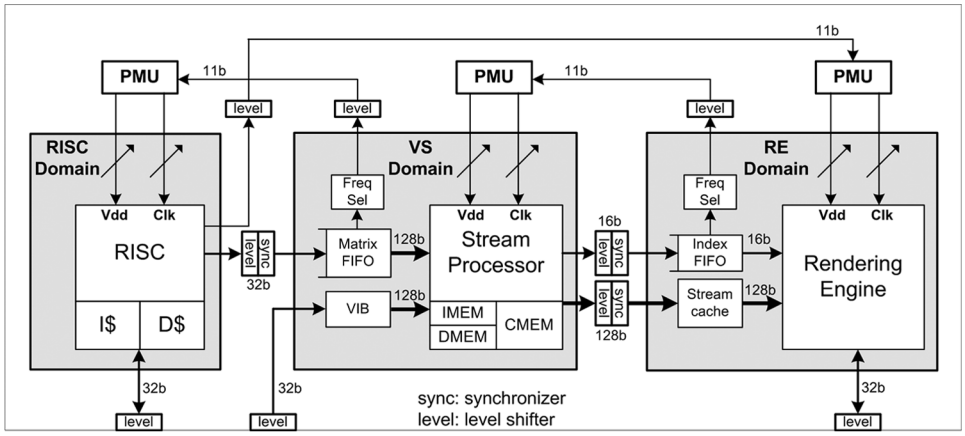
Since the objects in a scene are composed of a number of triangles and these triangles again consist of a number of pixels, the workloads for the RISC, VS, and RE, which operate per object, per triangle, and per pixel, respectively, can be completely different. Therefore, the RISC, VS, and RE are divided into different power domains and their frequencies and voltages are controlled separately according to the individual workloads. We divide the GPU into three different power domains of RISC, VS, and RE domains as illustrated in Fig. 16(a). The PMU for each power domain is based on the PLL structure and embeds a linear regulator inside the loop to synchronize the scaling of clock frequency and supply voltage [8]. Level shifters and synchronizers are inserted between the power domains to adjust signal levels and avoid the metastability of transferred data. The 4 kB matrix FIFO and 64-bytes index FIFO are also used across the modules to keep the throughput stable despite the response delay of the PMUs. In these FIFOs, the Gray coding, where two consecutive values differ in only one bit, is used for the read/write pointer for reliable operation between different read/write clock domains.

2) Stream Cache: In the geometry processing, a stream cache is provided to reuse the previously processed vertices without executing the TnL routine. The stream cache consists of a cache controller and a physical entry memory as shown in Fig. 16(b). It is implemented using the existing domain interface FIFO between the VS and the RE domain by simply augmenting the cache replacement capability without causing overheads. The 4 kB physical entry memory contains 16 VORs and shows 58% hit rate. Once a vertex is found in this cache, the TnL is achieved in a single cycle.

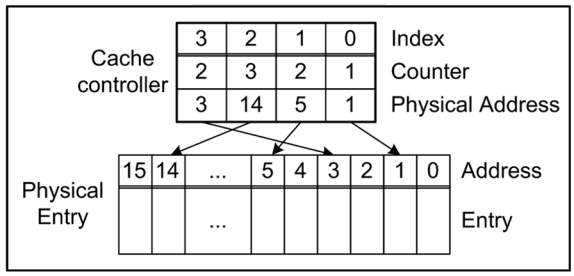
B. Chip Implementation

The SoC is fabricated using 0.18 μm 6-metal CMOS technology. It integrates RISC, VS, and RE into a small area of 17.2 mm^2 and incorporates 1.57 M transistors and 29 kB SRAM. The chip micrograph is shown in Fig. 17. The regions taken by each module are also depicted in the figure. The stream processor core as the VS takes 9.7 mm^2 and dissipates 86.8 mW power consumption at 200 MHz operating frequency and 1.8 V supply voltage. The clocks to pipeline registers are fully gated to disable unnecessary switching under the control of each instruction. The 2-cycle geometry transformation and the 58% hit-rate stream cache achieve the 141Mvertices/s for TFM and 12.1Mvertices/s for OpenGL TnL at 200 MHz clock frequency. Fig. 18 shows the shmoo plot of the processor. The measured waveform in Fig. 19 shows the MAT taking 2 cycles per result and 6-cycle latency. Each input is sustained for 2 cycles due to the interlock control from the FSM in ID stage.

With the triple-domain power management scheme, the entire chip consumes 52.4 mW when the scenes are drawn at 60 frames/s. The PMUs for the RISC and the VS cover from 89 MHz to 200 MHz with 1 MHz step and from 1.0 V to 1.8 V with 7.2 mV step while the PMU for the RE covers from



(a)



(b)

Fig. 16. SoC architecture of the proposed handheld GPU. (a) Overall GPU architecture. (b) Stream cache organization.

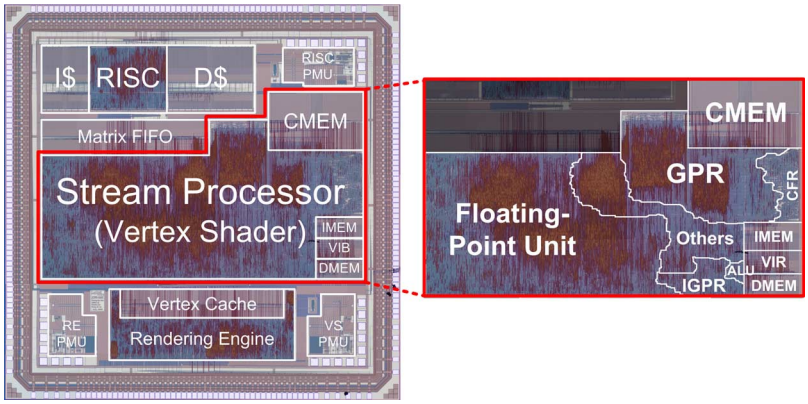


Fig. 17. Chip micrograph.

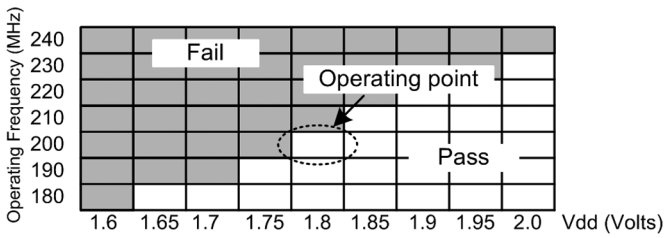


Fig. 18. Shmoo plot for vertex shader.

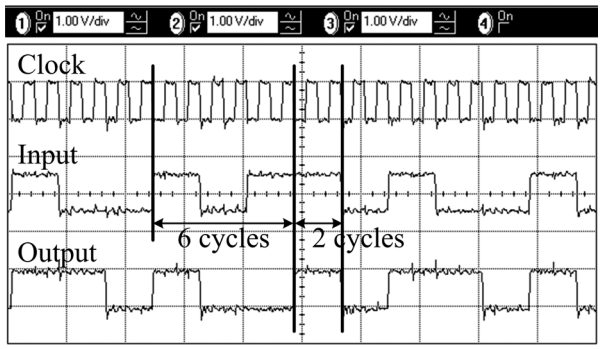


Fig. 19. Measured waveform of the MAT taking 6-cycle latency and 2 cycles per result.

22 MHz to 50 MHz with the 250 kHz step with the same voltage range and step. Since all of the modules are designed with static logic, the modules can keep running while the frequency and voltage scale. Each PMU takes 0.45 mm^2 and consumes less than 5.1 mW. The maximum power consumption of the entire chip is 153 mW when all components are running at their full speeds: 200 MHz at 1.8 V for RISC and VS and 50 MHz at 1.8 V for RE. Table VI summarizes the features of the chip.

TABLE VI
CHIP CHARACTERISTICS

Technology	TSMC 0.18um 1-poly 6-metal CMOS technology	
Area	Stream Processor: 9.7mm^2 GPU: 17.2mm^2	
Transistor counts	1.6M Logic (968K for Stream Processor) 29Kbyte SRAM	
Processing Speed	141Mvertices/s for TFM 12.1Mvertices/s for OpenGL TnL	
Power supply	1.0V - 1.8V for core, 3.3V for I/O	
Operating frequency	RISC : 89MHz - 200MHz VS : 89MHz - 200MHz RE : 22MHz - 50MHz	
Power consumption	Stream Processor: 86.8mW GPU: 52.4mW @ 60fps, 153mW @ full speed	
Features	Standards	OpenGL-ES 2.0 Shader Model 3.0 (w/o texture lookup instruction)
	Power Management	Triple DVFS power domains

C. Comparison

Since the area and the power overheads as well as the performance are important factors for the handheld devices, the figure of merit (FoM) should take them into account for its definition. Thus, the FoM is defined as (8), in which the performance is normalized by the area and power dissipation:

$$FoM = \frac{Performance(Kvertices/s)}{Power(mW) \times Area(mm^2)}. \quad (8)$$

According to this FoM, the proposed stream processor core shows 167.42 $\mathrm{Kvertices/s/mW/mm^2}$. Fig. 20 compares the FoM with other chips, whose performance, power, and area are listed in Table V. In terms of the proposed FoM, the proposed stream processor shows maximum 3.65 times improvement from the latest work [7], which incorporated 16 FLP MAC units to speedup the matrix-vector multiplication. This work also shows 17.5% and 22.2% performance improvement for TFM and TnL, respectively, while reducing power and area overheads by 44.7% and 39.4%, respectively.

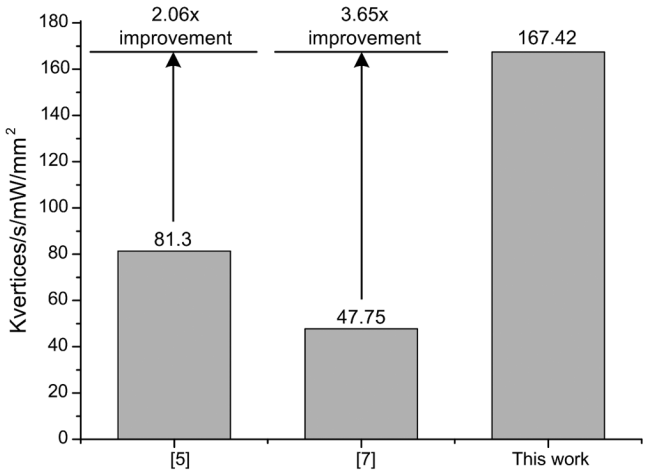


Fig. 20. Comparison of FoM.

By exploiting the logarithmic arithmetic, as shown in Table VII, our GPU shows similar processing speed, area and power dissipation to [7], while showing higher-level of integration: it incorporates the RISC and RE as well, which are not found in [7]. Moreover, when it does not have to operate its full speed, the clock frequency and supply voltage of each module can be scaled down dynamically and then it shows 50.5% power reduction from [7] when it runs at 60 fps.

VI. CONCLUSION

A high-performance, power- and area-efficient stream processor core is proposed for a low-power 3-D graphics SoC. It contains a 4-way 32-bit FLP unified arithmetic unit of matrix, vector, and elementary functions. Based on the logarithmic arithmetic and the proposed adaptive number conversion scheme, the unit unifies the matrix, vector, and elementary functions into a single 4-way arithmetic platform and achieves a single-cycle throughput for all the operations, except for the matrix-vector multiplication which takes 2 cycles per result. With the help of this unit and several proposed architectural features including embedded index calculations, functional unit reconfiguration, and operand forwarding in logarithmic domain, the proposed processor shows 19.1% cycle count reduction for the OpenGL TnL from the latest work.

TABLE VII
GPU COMPARISON WITH RELATED WORKS

Reference	Performance (Mvertices/s)	Power consumption		Area (mm^2)	Functions
		@ full speed (mW)	@ 60fps (mW)		
[4]	50	155	N.A.	23	RISC+VS+RE
[7]	120	157	106	16	VS only
This work	141	153	52.4	17.2	RISC+VS+RE

The stream processor core is integrated into a handheld GPU as a programmable vertex shader to show its effectiveness. A 17.2 mm^2 test GPU chip contains 1.57 M transistors and 29 kB SRAM and integrates the full 3-D graphics pipeline including an ARM-10 compatible RISC processor, a vertex shader, and a rendering engine. In addition, it contains three power management units for its triple-domain power management policy. The vertex shader achieves 141 Mvertices/s for the 3-D geometry transformation (TFM) at 200 MHz operating frequency and shows 17.5% performance improvement, while reducing 44.7% and 39.4% power and area overheads, respectively, from the latest work. It shows 3.65 times improvement in terms of our proposed figure of merits.

The SoC is divided into triple power domains of the RISC, VS, and RE with separate DVFS control and thereby, shows 52.4 mW power consumption when it runs at 60 fps, which is 50.5% power reduction from the latest work.

ACKNOWLEDGMENT

The authors would like to thank J. Lee, K. Kim, and S. Lee of KAIST for their contributions. The authors are also grateful to the anonymous reviewers for constructive comments on the manuscript.

REFERENCES

[1] Khronos Group, OpenGL-ES [Online]. Available: http://www.khronos.org/opengles
[2] Microsoft Corporation, Direct3D Mobile [Online]. Available: http://www.msdn.microsoft.com/en-us/library/Aa452478.aspx
[3] F. Arakawa, T. Yoshinaga, T. Hayashi, Y. Kiyoshige, T. Okada, and M. Nishibori *et al.*, "An embedded processor core for consumer appliances with 2.8 GFLOPS and 36 M Polygons/s FPU," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2004, pp. 334–335.
[4] J.-H. Sohn, J.-H. Woo, M.-W. Lee, H.-J. Kim, R. Woo, and H.-J. Yoo, "A 50 Mvertices/s graphics processor with fixed-point programmable vertex shader for mobile applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2005, pp. 192–193.
[5] J.-H. Sohn, J.-H. Woo, R. Woo, and H.-J. Yoo, "A fixed-point multimedia co-processor with 50 Mvertices/s Programmable SIMD vertex shader for mobile applications," in *Proc. Eur. Solid-State Circuits Conf. (ESSCIRC)*, Sep. 2005, pp. 207–210.
[6] D. Kim, K. Chung, C.-H. Yu, C.-H. Kim, I. Lee, and J. Bae *et al.*, "An SoC with 1.3 Gtexels/s 3-D graphics full pipeline for consumer applications," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 71–84, Jan. 2006.
[7] C.-H. Yu, K. Chung, D. Kim, and L.-S. Kim, "A 120 Mvertices/s multithreaded VLIW vertex processor for mobile multimedia applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2006, pp. 408–409.
[8] B.-G. Nam, J. Lee, K. Kim, S. J. Lee, and H.-J. Yoo, "A 52.4 mW 3-D graphics processor with 141 Mvertices/s vertex shader and 3 power domains of dynamic voltage and frequency scaling," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2007, pp. 278–279.
[9] B.-G. Nam and H.-J. Yoo, "A low-power vector processor using logarithmic arithmetic for handheld 3-D graphics systems," in *Proc. Eur. Solid-State Circuits Conf. (ESSCIRC)*, Sep. 2007, pp. 232–235.
[10] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens, "Programmable stream processors," *IEEE Computer*, vol. 36, no. 8, pp. 54–62, Aug. 2003.
[11] E. Lindholm, M. J. Kilgard, and H. Moreton, "A user-programmable vertex engine," in *Proc. SIGGRAPH 2001*, Aug. 2003, pp. 149–158.
[12] F.-S. Lai and C.-F. E. Wu, "A hybrid number system processor with geometric and complex arithmetic capabilities," *IEEE Trans. Computers*, vol. 40, no. 8, pp. 952–962, Aug. 1991.
[13] B.-G. Nam, H. Kim, and H.-J. Yoo, "A low-power unified arithmetic unit for programmable handheld 3-D graphics systems," *IEEE J. Solid-State Circuits*, vol. 42, no. 8, pp. 1767–1778, Aug. 2007.
[14] H.-J. Oh, S. M. Mueller, C. Jacobi, K. D. Tran, S. R. Cottier, B. W. Michael, H. Nishikawa, Y. Totsuka, T. Namatame, N. Yano, T. Machida, and S. H. Dhong, "A fully pipelined single-precision floating-point unit in the synergistic processor element of a CELL processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 4, pp. 759–771, Apr. 2006.
[15] K. Diefendorff, P. K. Dubey, R. Hochsprung, and H. Scales, "AltiVec extension to PowerPC accelerates media processing," *IEEE Micro*, vol. 20, no. 2, pp. 85–95, Mar./Apr. 2000.
[16] S. K. Srinivasan and M. N. Veley, "Formal verification of an Intel XScale processor model with scoreboarding, specialized execution pipelines, and imprecise data-memory exceptions," in *Proc. Formal Methods and Models for Codesign (MEMOCODE'03)*, Jun. 2003, pp. 65–74.
[17] N. Thomos, N. V. Boulgouris, and M. G. Strintzis, "Optimized transmission of JPEG2000 streams over wireless channels," *IEEE Trans. Image Processing*, vol. 15, no. 1, pp. 54–67, Jan. 2006.
[18] B.-G. Nam, J. Lee, K. Kim, S. J. Lee, and H.-J. Yoo, "A low-power handheld GPU using logarithmic arithmetic and triple DVFS power domains," in *Proc. SIGGRAPH/Eurographics Conf. on Graphics Hardware (GH)*, Aug. 2007, pp. 73–80.
[19] R. Cook, L. Torrance, and E. Kenneth, "A reflectance model for computer graphics," in *Proc. ACM SIGGRAPH'81*, 1981, pp. 307–316.
[20] M. Oren and S. Nayar, "Generalization of Lambert's reflectance model," in *Proc. ACM SIGGRAPH'94*, 1994, pp. 239–246.

Byeong-Gyu Nam (M'07) received the B.S. degree (*summa cum laude*) in computer engineering from Kyungbook National University, Daegu, Korea, in 1999, and M.S. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2001. From 2001 to 2002, he was with Electronics and Telecommunication Research Institute (ETRI), Daejeon, Korea. He received the Ph.D. degree in electrical engineering from KAIST in 2007. His Ph.D. research focused on low-power GPU design. He is currently working for Samsung Electronics, Giheung, Korea. His research interests include 3-D graphics processor, low-power arithmetic unit, and microprocessor design.

Hoi-Jun Yoo (M'95–SM'04–F'08) graduated from the Electronic Department of Seoul National University, Seoul, Korea, in 1983, and received the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 1985 and 1988, respectively. His Ph.D. work concerned the fabrication process for GaAs vertical optoelectronic integrated circuits.

From 1988 to 1990, he was with Bell Communications Research, Red Bank, NJ, where he invented the two-dimensional phase-locked VCSEL array, the front-surface-emitting laser, and the high-speed lateral HBT. In 1991, he became a manager of the DRAM design group at Hyundai Electronics and designed a family of fast-1M DRAMs to 256M synchronous DRAMs. In 1998, he joined the faculty of the Department of Electrical Engineering at KAIST and now is a full professor. From 2001 to 2005, he was the director of System Integration and IP Authoring Research Center (SIPAC), funded by Korean Government to promote worldwide IP authoring and its SOC application. From 2003 to 2005, he was the full time Advisor to Minister of Korea Ministry of Information and Communication and National Project Manager for SoC and Computer. In 2007, he founded System Design Innovation and Application Research Center (SDIA) at KAIST to research and to develop SoCs for intelligent robots, wearable computers and bio systems. His current interests are high-speed and low-power Network on Chips, 3-D graphics, Body Area Networks, biomedical devices and circuits, and memory circuits and systems. He is the author of the books *DRAM Design* (Hongleung, 1996; in Korean), *High Performance DRAM* (Sigma, 1999; in Korean), *Low-Power NoC for High-Performance SoC Design* (CRC Press, 2008), and chapters of *Networks on Chips* (Morgan Kaufmann, 2006).

Dr. Yoo received the Electronic Industrial Association of Korea Award for his contribution to DRAM technology in 1994, the Hynix Development Award in 1995, the Design Award of ASP-DAC in 2001, the Korea Semiconductor Industry Association Award in 2002, the KAIST Best Research Award in 2007, and the Asian Solid-State Circuits Conference (A-SSCC) Outstanding Design Awards in 2005, 2006 and 2007. He is serving as an Executive Committee Member and the Far East Secretary for IEEE ISSCC, and a Steering Committee Member of IEEE A-SSCC. He was the Technical Program Committee Chair of A-SSCC 2008.