Graphics Hardware (2007)
Timo Aila and Mark Segal (Editors)

A Low-Power Handheld GPU using Logarithmic Arithmetic and Triple DVFS Power Domains

Byeong-Gyu Nam, Jeabin Lee, Kwanho Kim, Seung Jin Lee, and Hoi-Jun Yoo

Department of EECS, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea

Abstract
In this paper, a low-power GPU architecture is described for the handheld systems with limited power and area budgets. The GPU is designed using logarithmic arithmetic for power- and area-efficient design. For this GPU, a multifunction unit is proposed based on the hybrid number system of floating-point and logarithmic numbers and the matrix, vector, and elementary functions are unified into a single arithmetic unit. It achieves the single-cycle throughput for all these functions, except for the matrix-vector multiplication with 2-cycle throughput. The vertex shader using this function unit as its main datapath shows 49.3% cycle count reduction compared with the latest work for OpenGL transformation and lighting (TnL) kernel. The rendering engine uses also the logarithmic arithmetic for implementing the divisions in pipeline stages. The GPU is divided into triple dynamic voltage and frequency scaling power domains to minimize the power consumption at a given performance level. It shows a performance of 5.26Mvertices/s at 200MHz for the OpenGL TnL and 52.4mW power consumption at 60fps. It achieves 2.47 times performance improvement while reducing 50.5% power and 38.4% area consumption compared with the latest work.

Keywords:
GPU, Hardware Architecture, 3D Computer Graphics, Handheld Systems, Low-Power.

1. Introduction

As the mobile electronics advances rapidly, the handsets like cell phones and PDAs are adopting the realtime 3D computer graphics with high performance processors consuming only limited power and area. Recently, the embedded 3D graphics API like OpenGL-ES [Khr05] is defined and it adopts the programmability into the 3D graphics pipeline to support various graphics effects. Thus, the vector processors called *shaders* are included into the GPU for the programmability and it is required to support matrix, vector, and elementary functions to process various geometry transformation and lighting kernels.

There have been several studies on the programmable graphics processors. The conventional shader architecture incorporates the 4-way vector SIMD unit for the vector multiply, multiply-add, and dot-product and special function unit (SFU) for reciprocal, reciprocal-squre-root, logarithm ($\log_2 x$), and exponential (2^x) functions [LKM01]. A graphics processor for the cell phones is proposed in [KKF*03] and it incorporates 2-way SIMD unit for multiply-add and special function unit for reciprocal, reciprocal-square-root, and power functions. However, it showed limited performance of 185Kpolygons/s, which is far below the performance level supported by modern handheld GPUs [SWY04, AYH*04, KCY*06, YCK*06]. The vertex shader based on the fixed-point arithmetic is proposed in [SWY04] for the power- and area-efficient design. It shows relative high graphics performance of 7.2Mvertices/s for OpenGL transformation and lighting (TnL). However, its dynamic range was limited by the fixed-point arithmetic.

In this paper, a power- and area-efficient GPU based on the logarithmic arithmetic is proposed. Although the logarithmic arithmetic carries a little computation error, it can significantly reduce the arithmetic complexity [Mit62, CCS*00]. Therefore, the logarithmic arithmetic is adopted for our GPU, which is targeted for the handheld systems with small size screens. In addition, the proposed GPU is divided into triple power domains with dynamic voltage and frequency scaling (DVFS) for the minimum power consumption at a given performance level. The concept of using separate power domains for the geometry and rendering stages was proposed in [SLS04]. It is extended into triple power domains and implemented in our GPU.

The paper is organized as follows. First, the GPU architecture is proposed. The number system using the logarithmic arithmetic is described for our GPU design. Then, the multifunction unit based on this number system is presented and followed by the vertex shader including this unit. The rendering engine using the logarithmic arithmetic is also presented. After that, the power management scheme is described and the evaluation results will be shown. Finally, conclusions are made.

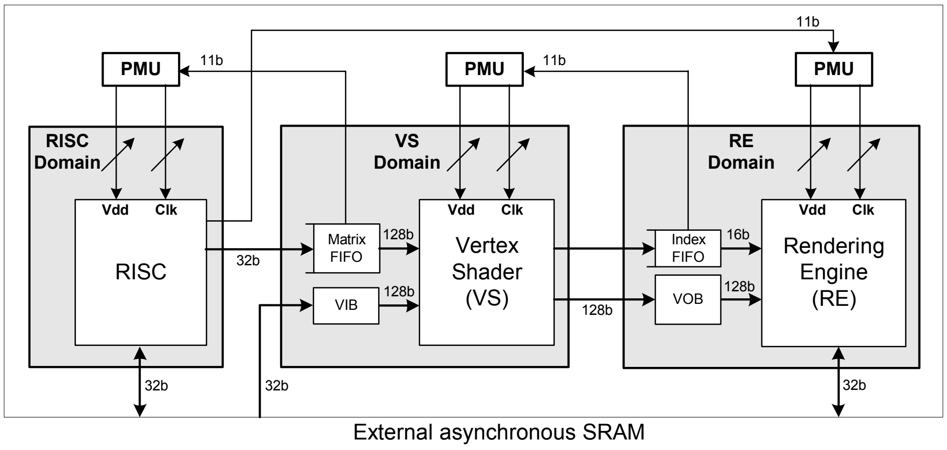

Figure 1: *Overall architecture of the proposed GPU*

2. Handheld GPU Architecture

Figure 1 shows the architecture of the proposed handheld GPU. It mainly consists of three major modules i.e. an ARM10-compatible RISC processor, a vertex shader, and a rendering engine. The RISC processor is included for simulation of the artificial intelligence and collision detections required for 3D gaming applications. The vertex shader (VS) and the rendering engine (RE) are designed based on the benefits of the logarithmic arithmetic, from which the complexities of arithmetic operations are reduced.

The GPU is divided into triple power domains to manage the power consumption of each module independently to get the lowest power consumption at a given performance level. Three power management units (PMUs) are included to manage the clock frequency and supply voltage of each domain. The FIFOs are used across the power domains for buffering the transferred data between each domain in spite of the long response delay of the power management units.

3. Hybrid Number System

It is known that the logarithmic number system (LNS) can simplify the arithmetic operations like multiplication, division, and square-root into addition, subtraction, and right shift, respectively. However, the addition and subtraction in the LNS become more complex and requires nonlinear function evaluations. Therefore, a hybrid number system (HNS) of the LNS and floating-point number system (FLP) was introduced to solve this problem in [LW91], where the addition and subtraction are carried out in the FLP while the other operations are performed in the LNS. For the number conversion between floating-point and logarithmic numbers in the HNS, the logarithmic and antilogarithmic converters are proposed. These are based on the piecewise linear approximation scheme to reduce area and power consumption [AS03, AbS03].

3.1. Logarithmic converter

When x is 32-bit FLP input, x and its logarithm can be represented by (1) and (2), respectively.

$$x = 2^e(1+m) \qquad (1)$$

$$\log_2 x = e + \log_2(1+m) \qquad (2)$$

The e is the integer part and $\log_2(1+m)$ is the fractional part of the logarithmic number. The nonlinear term $\log_2(1+m)$ is approximated by piecewise linear expressions as (3).

$$\log_2(1+m) \approx a \cdot m + b \qquad (3)$$

where a and b are the approximation coefficients defined for each approximation region. It achieves maximum 0.41% conversion error with 15 approximation regions.

3.2. Antilogarithmic converter

When X is the LNS, its FLP number can be represented by (4).

$$2^X = 2^{e+f} = 2^e \cdot 2^f \qquad (4)$$

The integer part e directly becomes the exponent of the FLP number and the nonlinear term 2^f is approximated by piecewise linear expressions like (5).

$$2^f \approx a \cdot f + b \qquad (5)$$

where a and b are the approximation coefficients for each approximation region. It achieves maximum 0.08% conversion error with 8 approximation regions.

4. Multifunction Unit

The proposed functional unit operates on the HNS to reduce the arithmetic complexities. It unifies the matrix (MAT), vector (VEC), and elementary (ELM) functions into a single arithmetic unit and its operation set includes the matrix-vector multiplication, vector multiply, divide, divide-by-square, multiply-add, lerp, dot-product, cross-product, power, logarithm, and trigonometric (TRG) functions like sine, cosine, and arctangent, etc.. It achieves single-cycle throughput with maximum 5-cycle latency for all the operations except for the matrix-vector multiplication which takes 2-cycle throughput and 6-cycle latency.

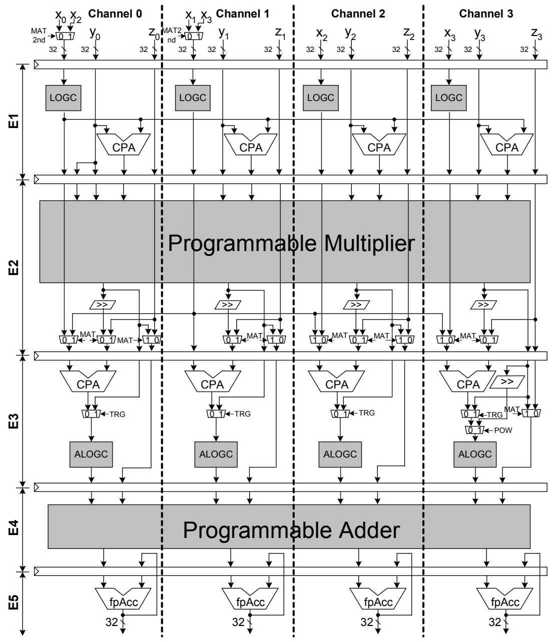

Figure 2: *Multifunction unit*

4.1. Overall architecture

The arithmetic unit is organized with 4-channels and 5 pipeline stages as shown in Figure 2. The 4 of input 32-bit FLP operands are converted into the logarithmic number with 8-bit integer, 24-bit fraction, 1-bit sign, and 1-bit zero through the 4 logarithmic converters (LOGCs) in the E1. The E2 includes the programmable multiplier (PMUL) as shown in Figure 3, which can be used for the Booth multiplier for ELM, 4 LOGCs for VEC or 4 antilogarithmic converters (ALOGCs) for MAT by just adding 15-entry 64B LOG and 8-entry 56B ALOG lookup-tables to the Booth multiplier and sharing the carry-save-adder (CSA) tree and a carry-propagate-adder (CPA). In this way, the number of LOGCs in E1 is reduced to 4, which were 8 in [NKY06]. In E3, 4 adders in logarithmic domain are provided for the VECs and the resulting values are converted into the FLP numbers through the 4 ALOGCs. In E4, the programmable adder (PADD) shown in Figure 4 can be programmed into a single 5-input FLP adder tree or 4-way 2-input SIMD FLP adders for target operations as proposed in [NKY06]. The E5 provides a SIMD FLP accumulator for the final accumulation required for the MAT. It is also used as a rounding logic for other operations.

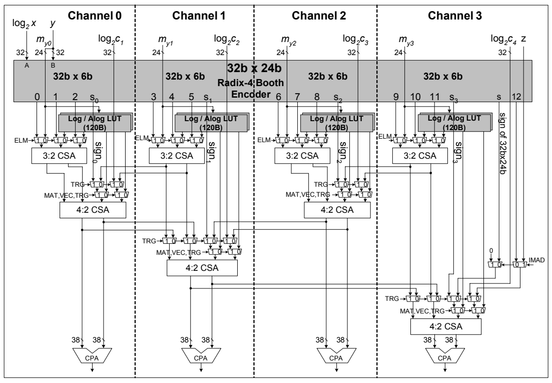

Figure 3: *Programmable multiplier (PMUL)*

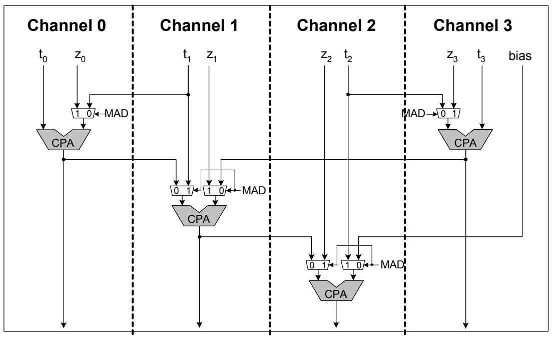

Figure 4: *Programmable adder (PADD)*

4.2. Matrix-vector multiplication

The geometry transformation in 3D graphics can be computed by the multiplication of 4×4-matrix with 4-element vector, which requires 16 multiplications and 12 additions. This can be converted into the HNS as (6) requiring 20 LOGCs, 16 adders, 16 ALOGCs, and 12 FLP adders. Since the coefficients of a geometry transformation matrix are fixed during processing of 3D object, these can be pre-converted into the logarithmic domain and used as constants during the processing. Thus, the MAT only requires 4 LOGCs for vector element conversion, 16 adders in logarithmic domain, 16 ALOGCs and 12 FLP adders. This can be implemented in 2 phases on this 4-way arithmetic unit as illustrated in Figure 5. In this scheme, 8 adders in logarithmic domain and 8 ALOGCs are required per phase and the 8 ALOGCs in the first phase are obtained from 4 ALOGCs in E2 by programming the PMUL into 4 ALOGCs together with the 4 ALOGCs in E3. The CPAs in E1 and E3 are used for the 8 adders in logarithmic domain. The 4 multiplication results from the ALOGCs in E2 and the other 4 from the E3 are added in the E4 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 result in E5 completes the MAT. Thus, the MAT is implemented with 2-cycle throughput on this 4-way arithmetic unit, where it was implemented with 4-cycle throughput in conventional way [NKY06, SWY04].

$$\begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} c_{00} \\ c_{10} \\ c_{20} \\ c_{30} \end{bmatrix} x_0 + \begin{bmatrix} c_{01} \\ c_{11} \\ c_{21} \\ c_{31} \end{bmatrix} x_1 + \begin{bmatrix} c_{02} \\ c_{12} \\ c_{22} \\ c_{32} \end{bmatrix} x_2 + \begin{bmatrix} c_{03} \\ c_{13} \\ c_{23} \\ c_{33} \end{bmatrix} x_3 = \tag{6}$$

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

$$\boxed{\underbrace{\underbrace{\boxed{2^{\left(\begin{bmatrix} \log_2 c_{00} \\ \log_2 c_{10} \\ \log_2 c_{20} \\ \log_2 c_{30} \end{bmatrix} + \log_2 x_0\right)} + 2^{\left(\begin{bmatrix} \log_2 c_{01} \\ \log_2 c_{11} \\ \log_2 c_{21} \\ \log_2 c_{31} \end{bmatrix} + \log_2 x_1\right)}}}_{\text{phase1}} + 2^{\left(\begin{bmatrix} \log_2 c_{02} \\ \log_2 c_{12} \\ \log_2 c_{22} \\ \log_2 c_{32} \end{bmatrix} + \log_2 x_2\right)} + 2^{\left(\begin{bmatrix} \log_2 c_{03} \\ \log_2 c_{13} \\ \log_2 c_{23} \\ \log_2 c_{33} \end{bmatrix} + \log_2 x_3\right)}}_{\text{phase2}}}$$

Figure 5: *2-phase implementation of MAT*

4.3. Vector operations

The vector multiplication, division, square root, and multiply-add (MAD) can be represented by a single generic operation and is converted into HNS like (7). For example, operations like $x \times y$, $x \div \sqrt{y}$, $x \times y + z$ can be represented with this generic operation.

$$\left(x_i \otimes y_i^s \oplus z_i\right)_{i\in\{0,1,2,3\}} = \left(2^{(\log_2 x_i)\oplus(s\times\log_2 y_i)} \oplus z_i\right)_{i\in\{0,1,2,3\}} \tag{7}$$
$$\textit{where } \otimes \in \{\times,\div\}, \oplus \in \{+,-\}, s \in \{0.5, 1\}$$

Since the equation (7) require 2 LOGCs for 2 operands per channel, the PMUL is programmed into 4 LOGCs to make the 8 LOGCs for 4 channels together with the 4 LOGCs in E1. This operation implements the square root in conjunction with division like $x \div \sqrt{y}$ without any additional cycle. This is useful for the vector normalization widely used in 3D graphics.

The dot product (DOT) in the HNS is defined as (8).

$$x_0 \times y_0 + x_1 \times y_1 + x_2 \times y_2 + x_3 \times y_3 = \tag{8}$$
$$2^{\log_2 x_0 + \log_2 y_0} + 2^{\log_2 x_1 + \log_2 y_1} + 2^{\log_2 x_2 + \log_2 y_2} + 2^{\log_2 x_3 + \log_2 y_3}$$

It is implemented with the vector multiplication and the final summation of the product terms. For the final summation, the PADD is programmed into a single 5-input FLP adder tree. The *bias* port is used for TRGs.

4.4. Elementary functions

Since the power is converted into the multiplication in logarithmic domain, it requires a 32b×24b multiplier and it is implemented by programming the PMUL into a single 32b×24b BMUL.

$$x^y = 2^{y\times\log_2 x} \tag{9}$$

The TRGs are unified with others using the Taylor series as proposed in [NKY06]. For the first five terms of the Taylor series computation, a new generic operation is defined and converter into the HNS as (10).

$$c_0x^{k_0} \oplus c_1x^{k_1} \oplus c_2x^{k_2} \oplus c_3x^{k_3} \oplus c_4x^{k_4} =$$
$$c_0x^{k_0} \oplus 2^{\log_2 c_1 + k_1\times\log_2 x} \oplus 2^{\log_2 c_2 + k_2\times\log_2 x} \oplus 2^{\log_2 c_3 + k_3\times\log_2 x} \oplus 2^{\log_2 c_4 + k_4\times\log_2 x}$$
$$\textit{where } \oplus \in \{+,-\}, c_i, \log_2 c_i, k_i \textit{ are coefficients}$$
$$\tag{10}$$

Since the k_i is small integer, this power series is implemented by a 4-way 32b×6b multiplier in logarithmic domain and final summation of these terms. The first term doesn't need to be converted into logarithmic domain since the first term of the Taylor series usually a constant or the input x and it can be directly fed into the *bias* port. Thus this can be implemented by programming the PMUL into 4-way 32b×6b BMUL and the PADD into a single 5-input summation tree.

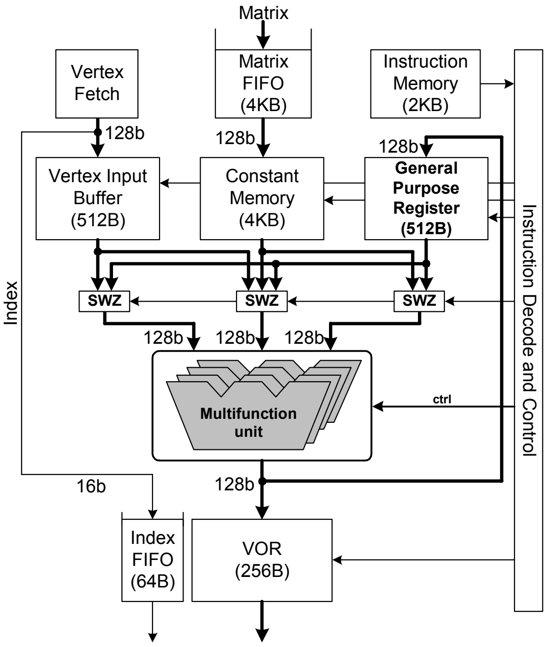

Figure 6: *Vertex shader*

5. Vertex Shader

This section describes the proposed vertex shader, as graphically shown in Figure 6. It is based on the multi-function unit proposed in section 4. It has 4-way 32-bit FLP vector register files including 512-byte 32-entry vertex input register (VIR), general purpose register file (GPR), 4KB 256-entry constant memory (CMEM), and 256-byte 16-entry vertex output register (VOR). The FLP operands are fetched from the GPR, VIR or CMEM and the result is written back to the GPR or VOR. The FLP instruction set of the vertex shader is shown in Table 1.

Mnemonic	Description	Type	Latency/ Throughput	Max. error (%)
ADD	Vector multiply	(V,V) -> V	1/1	-
MUL	Vector multiply	(V,V) -> V	3/1	3.09
MAD	Vector multiply-add	(V,V,V) -> V	5/1	0.21
DIV	Vector divide	(V,V) -> V	3/1	0.207
DSQ	Vector divide-by-sqrt	(V,V) -> V	3/1	0.199
DOT	Vector dot-product	(V,V) -> V	5/1	2.434
MAT	Matrix-vector multiply	(M,V) -> V	6/2	0.0825
POW	Power	S -> (S,S,S,S)	3/1	2.096
LOG	Logarithm	S -> (S,S,S,S)	3/1	0.403
SIN	Sine	S -> (S,S,S,S)	5/1	0.1175
COS	Cosine	S -> (S,S,S,S)	5/1	1.14
ATAN	Arctangent	S -> (S,S,S,S)	5/1	7.1

M: Matrix, V: Vector, S: Scalar

Table 1: *Floating-point instruction set of vertex shader*

The operand forwarding improves the throughput of the processor pipeline. In this processor, the operand forwarding in the logarithmic domain is also supported. As shown in Figure 7, in the case of consecutive FLP operations, the antilogarithmic and logarithmic conversions cancel each other and the intermediate logarithmic value of the previous FLP operation is forwarded directly into the logarithmic domain of the next FLP operation bypassing the antilogarithmic and logarithmic converters of two operations. This reduces the pipeline latency and the computation error by bypassing the repeated antilogarithmic and logarithmic conversions, which are the sources of the computation errors.

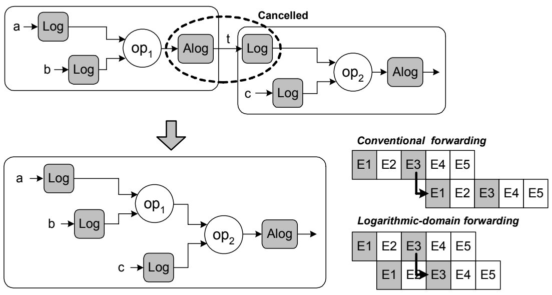

Figure 7: *Logarithmic-domain forwarding*

6. Rendering Engine

Figure 8 shows the block diagram of the rendering engine. It consists of a triangle setup engine (TSE), a rasterizer, and a pixel pipeline. All divisions required in these stages are implemented in the logarithmic domain for the power- and area-efficient design. The TSE and rasterizer computes SIMD interpolation i.e. $p_0 + (\Delta p/\Delta y) \times y$ to setup triangle parameters. The sequence of division and multiplication required for the interpolation are implemented as a sequence of subtraction and addition in the logarithmic domain as in (11).

$$p_0 + (\Delta p/\Delta y) \times y = p_0 + 2^{(\log_2 \Delta p - \log_2 \Delta y + \log_2 y)} \qquad (11)$$

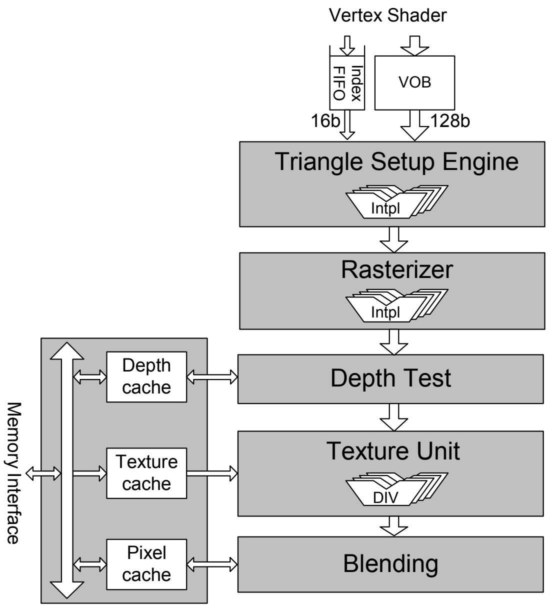

Figure 8: *Rendering engine*

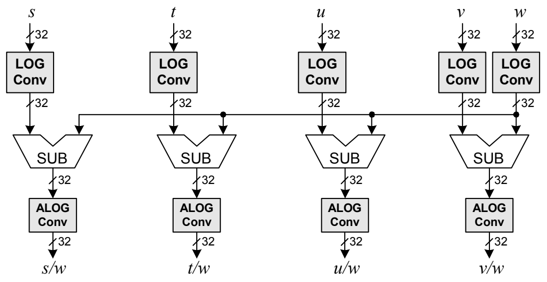

Figure 9: *SIMD logarithmic divider*

For the perspective correct texture address calculation, the texture unit requires a 4-way SIMD divider, $(s,t,u,v)/w$. This is also implemented as subtractions in the logarithmic domain according to (12), reducing total area and power consumption.

$$(s,t,u,v)/w = 2^{(\log_2 s, \log_2 t, \log_2 u, \log_2 v) - \log_2 w} \qquad (12)$$

The implementation of the division in logarithmic domain is shown in Figure 9.

7. Multiple-domain Power Management

In this GPU, triple power domains with separate frequencies and voltages are tuned by tracking the workloads. This GPU uses two levels of hierarchical power management scheme i.e. inter-frame and intra-frame level. Since 3D graphics scenes are displayed at a given frame rate, the RE should draw only finite amount of pixels within the time slot of a single frame. Therefore, at the inter-frame level, the target frequency and supply voltage for the RE is determined. In every completion of drawing a scene, the software library running on the RISC measures the time elapsed for the drawing and sets a new target frequency and supply voltage adaptively for the processing of next scene.

Since the objects in a scene are composed of a number of triangles and these triangles are again composed of a number of pixels, the workloads of the RISC, VS and RE, which operate per object, per triangle, and per pixel, respectively, can be completely different. Therefore, at the intra-frame level of power management, the RISC, VS and RE are divided into different power domains and their frequencies and voltages are separately controlled according to their workloads. Figure 10 illustrates the proposed power management scheme.

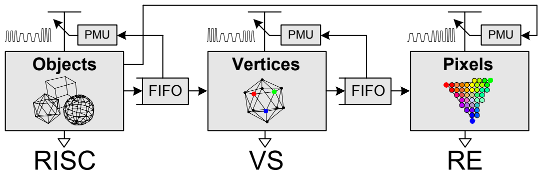

Figure 10: *Triple-domain DVFS*

The workloads of the RISC and the VS are obtained from measuring the occupation levels of the 16-entry matrix FIFO and 32-entry index FIFO, respectively. Each occupation level is compared with its reference level to give the error value, thus new target frequency.

8. Evaluation Results

8.1. Accuracy

The proposed GPU based on the logarithmic arithmetic scheme has been verified using 3D graphics software environment with FLP and HNS libraries. A test 3D graphics scene is shown in Figure 12 to show the reliability of the proposed GPU. The test model consists of 5,878 triangles and the screen size is QVGA i.e. 320×240. The test model was rendered with OpenGL TnL operation to show the effects of the mixture of various operations. The average TnL error between two scenes is 0.035%. The in-box shows a zoomed image for the accuracy comparison. The result shows that the computation error from the HNS is tolerable for the small screen of handheld systems.

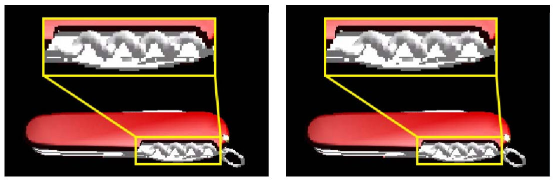

Figure 12: ***Scene comparison***

8.2. Implementation

The proposed GPU was implemented with 1.57Mtransistors using TSMC 0.18μm CMOS technology. The maximum operating frequency is 200MHz for the RISC processor and the vertex shader and 50MHz for the rendering engine while comsuming 153mW in total.

For a given performance level, each module doesn't have to operate at its full speed and its operating frequency and supply voltage can be lowered. The power consumption is measured to be 52.4mW at 60fps for the test model used in section 8.1. As shown in Figure 11, the droop of the FIFO entry level, i.e. high workload due to small triangle, increases the frequency and supply voltage of the vertex shader to speed up until the FIFO entry level returns to the reference point.

8.3. Performance

The performance is evaluated for the following full OpenGL TnL routine implemented using our instruction set. It includes the model-view, normal, and perspective transformations, normalizations of light, view, normal, and Blinn half vectors, and intensity calculations of diffuse and specular lightings for a single light source. After rescheduling of the code to avoid the dependencies, the required execution cycle for it on the proposed processor is 38 cycles. With the same process, the cycle counts for this routine are estimated for other works based on their reported cycle counts of the instruction set. Figure 13 shows the comparison of cycle counts and our work shows 19.1% cycle count reduction from the latest work [YCK*06]. Table 2 shows the comparison with other implementations and our work shows 2.47 times performance improvement, while reducing 50.5% power and 38.4% area overheads from the latest work [YCK*06].

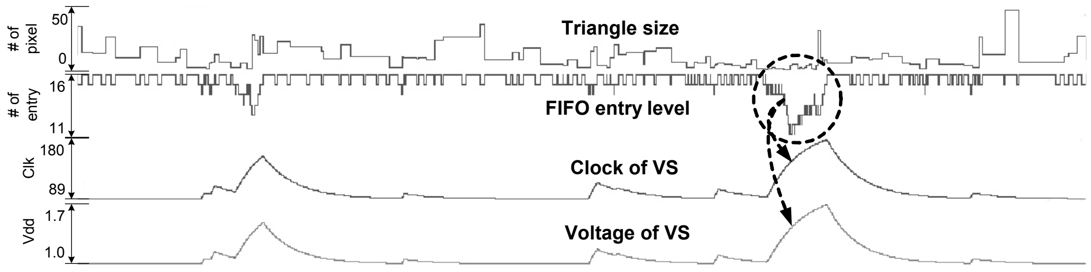

Figure 11: *DVFS operation diagram*

Geometry Transformation and OpenGL Lighting

```
//------------------------------ constant memory ------------------------------//
// c[0-3]  = modelview transformation matrix preconverted into LNS
// c[4-7]  = modelview inverse transpose preconverted into LNS
// c[8-11]  = modelview projection matrix preconverted into LNS
// c[20]   = light position
// c[21]   = precomputed amb_mat * amb_lit
// c[22]   = precomputed diff_mat * diff_lit
// c[23]   = precomputed spec_mat * spec_lit
// c[33]   = zero: (0.0,0.0,0.0,0.0)
//---------------------------------- input register -------------------------------//
// i0      = vertex coordinate
// i1      = vertex normal
// i2      = vertex specular exponent
//-------------------------------- output register -------------------------------//
// o0      = vertex coordinate in screen space
// o1      = vertex color

//--------------------- ModelView Transformation --------------------------//
// vertex transformation to eye space
MAT  t0, i0, c[0]
// normal transformation to eye space
MAT  t1, i1, c[4]

//----------------------- Perspective Transformation -----------------------//
// vertex transformation to clip space
MAT  t2, i0, c[8]

// 1/w for screen space mapping
DIV  o0.xyz, t2, t2.w

//------------------------------------- Lighting -----------------------------------//
// convert vertex coordinate into Euclidian space
DIV  t0.xyz, t0, t0.w

// compute normalized light direction
ADD  t3.xyz, -t0, c[20]
DP3  t4.xyz, t3, t3
DSQ  t3.xyz, t3, t4

// normalize normal vector
DP3  t5.xyz, t1, t1
DSQ  t1.xyz, t1, t5

// normalize view vector
DP3  t6.xyz, -t0, -t0
DSQ  t0.xyz, -t0, t6

// compute normalized Blinn half vector
ADD  t7.xyz, t3, t0
DP3  t8.xyz, t7, t7
DSQ  t7.xyz, t7, t8

// N.L
DP3  t9.x, t1, t3
MAX  t9.rgb, t9.x, c[33].x

// N.H
DP3  t10.x, t1, t7
MAX  t10.x, t10.x, c[33].x

// diffuse lighting
MAD  t13.rgb, t9, c[22], c[21]

// specular lighting
MOV  t14.x, i2.x
POW  t15.rgb, t10.x, t14.x
MAD  o1.rgb, t15, c[23], t13
```

Our programmable vertex shader can process various vertex shading kernels other than the OpenGL TnL. When processing other TnL kernels with the Oren-Nayar (O-N) [ON94] and the Cook-Torrance (C-T) [CT81] lighting models, our vertrex shader takes 73 and 54 cycles for the O-N and the C-T, respectively. The O-N model enhances the diffuse lighting model for rough faces and reveals the outstanding performance improvement from our vertex shader because the kernel includes several trigonometric function evaluations which are not directly supported in others [SWY04, AYH*04, KCY*06, YCK*06]. The C-T model improves the specular lighting used for metals and plastics and it also requires several power function evaluations. As shown in Figure 13, our processor shows 49.3% and 23.9% cycle count reduction for the O-N and the C-T kernels, respectively, from the latest work [YCK*06].

9. Conclusion

A handheld GPU using the logarithmic arithmetic

	Performance (Mvertices/s)	mW @ full speed	mW @ 60fps	Area (Kgates)	Frequency (MHz)	Process (μm)	Functions	Kvertices/s/MHz
[KKF*03]	0.185	38	-	360	30	0.18	GE+RE	6.17
[SWY04]	7.2	115	-	230	400	0.13	GE	18
[YCK*06]	2.13	157	106	375	100	0.18	GE	21.3
This work	5.26	153	52.4	393 (242 for GE)	200	0.18	GE+RE	26.3

Table 2: *Performance comparison of handheld GPUs*

and triple DVFS power domains is proposed. The floating-point vertex shader is implemented using the hybrid approach of floating-point and logarithmic number systems. It operates on the floating-point operands and uses the logarithmic arithmetic internally to reduce the arithmetic complexity. In this way, the proposed multifunction unit unifies the matrix, vector, and elementary functions into a single 4-way arithmetic unit. It implements all these functions by programming the proposed programmable multiplier and the programmable adder. The vertex shader with this arithmetic unit achieves single-cycle throughput with maximum 5-cycle latency for all the operations, except for the matrix-vector multiplication with 2-cycle throughput and 6-cycle latency. The rendering engine is also implemented with logarithmic dividers for power- and area-efficient design. The scene comparison result shows that the error from the logarithmic arithmetic is tolerable for the handheld systems with small screens. In addition, the GPU is divided into triple power domains with dynamic voltage and frequency scaling to minimize power consumption at a given performance level. Our GPU shows maximum 49.3% cycle count reduction and 2.47 times performance improvement while reducing 50.5% power and 38.4% area overheads from the latest work.

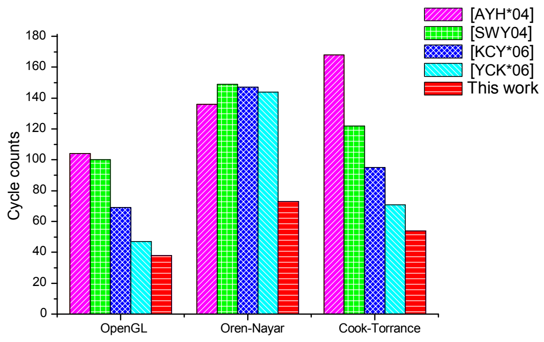

Figure 13: *Comparison of cycle counts for each kernel*

References

[AS03] Abed K., Sifred R.: CMOS VLSI implementation of a low-power logarithmic converter. *IEEE Transactions on Computers*, 52(11):1421-1433, Nov. 2003.

[AbS03] Abed K., Sifred R.: VLSI implementation of a low-power Antilogarithmic converter. *IEEE Transactions on Computers*, 52(9):1221-1228, Sept. 2003.

[AYH*04] Arakawa F., Yoshinaga T., Hayashi T., Kiyoshige Y., Okada T., Nishibori M., et al.: An embedded processor core for consumer appliances with 2.8GFLOPS and 36M Polygons/s FPU. In *IEEE Int. Solid-State Circuits Conference Digest of Technical Papers*, pp. 334-335, Feb. 2004.

[CCS*00] Coleman J.N., Chester E.I., Softley C.I., Kadlec J.: Arithmetic on the European Logarithmic Microprocessor. *IEEE Transactions on Computers* 49(7):702-715, July 2000.

[CT81] Cook R., Torrance L., Kenneth E.: A reflectance model for computer graphics. In *Proc. SIGGRAPH'81*, pp. 307-316, July 1981.

[KCY*06] Kim D., Chung K., Yu C., Kim C., Lee I., Bae J., et al.: An SoC with 1.3Gtexels/s 3-D graphics full pipeline for consumer applications. *Journal of Solid-State Circuits*, 41(1):71-84, Jan. 2006.

[Khr05] Khronos Group: OpenGL-ES 2.0. http://www.khronos.org, 2005.

[KKF*03] Kameyama M., Kato Y., Fujimoto H., Negishi H., Kodama Y., Inoue Y., Kawai H.: 3D graphics LSI core for mobile phone-Z3D. In *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003*, pp. 60-67, Aug. 2003.

[LKM01] Lindholm E., Kilgard M., Moreton H.: A user-programmable vertex engine. In *Proc. SIGGRAPH 2001*, pp. 149-158, Aug. 2001.

[LW91] Lai F., Wu C.: A hybrid number system processor with geometric and complex arithmetic capabilities. *IEEE Transactions on Computers*, 40(8):952-962, Aug. 1991.

[Mit62] Mitchell J.: Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers* 11:512-517, Aug. 1962.

[NKY06] Nam B., Kim H., Yoo H.: A low-power unified arithmetic unit for programmable handheld 3-D graphics systems. In *Proc. IEEE Custom Integrated Circuits Conference*, pp. 535-538, Sept. 2006.

[ON94] Oren M., Nayar S.: Generalization of Lambert's Reflectance Model. In *Proc. ACM SIGGRAPH'94*, pp. 239-246, 1994.

[SLS04] Sheaffer J., Luebke D., Skadron K.: A flexible simulation framework for graphics architectures. In *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware 2004,* pp. 85-94, Aug. 2004.

[SWY04] Sohn J., Woo R., Yoo H.: A programmable vertex shader with fixed-point SIMD datapath for low power wireless applications. In *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware 2004*, pp. 107-114, Aug. 2004.

[YCK*06] Yu C., Chung K., Kim D., Kim L.: A 120Mvertices/s multi-threaded VLIW vertex processor for mobile multimedia applications. In *IEEE Int. Solid-State Circuits Conference Digest of Technical Papers*, pp. 408-409, Feb. 2006.