# Design of Programmable Pixel Shader Computing Unit

**C. Kanagasabapathi, Siddamma, Siva S Yellampalli**

***Abstract*: With increased focus on improved Graphics, the gaming industries for computer and mobile devices requires more complex graphical algorithms, which in need of better computational resources. In general graphical computation requires more computational units, which will be major constraint in mobile devices as it demand for more power. In order to develop such efficient computational units, one need to focus on the Graphical pipeline to have better visual effects. In this paper design of programmable Pixel shader computing unit is discussed. In GPU, the Pixel shaders are used to process and manipulate the each pixel. Also the Pixel Shaders are used to provide final coloring to the processed data. Here the programmable Pixel shader computing unit design is discussed, which help the programmer to add his own code for achieving better visual effects or results. The possible Instructions of Pixel shader are designed using Verilog HDL. The SIMD(Single Instruction Multiple Data) concept is adopted for the designing of each instruction.***

***Index Terms*: GPU, Pixels, Pixel Shader, Graphical Rendering, SIMD.**

## I. INTRODUCTION

Present day computer-Systems and cell phone devices needs better quality of graphics for various applications such as gaming etc. The key innovation in this modern Graphics processing unit is the substitution of fixed-functional blocks by programmable one. Here the programmable block provides facility to the application developer to write his own written codes or programs. These applications are executed at very fast rate using SIMD technique. In modern systems the GPUs are widely used in addition to the traditional CPUs. The CPUs are intended mainly to perform mathematical-functions and some of basic-graphics related functions, such as Microsoft Power-Point and very low-resolution videos.

The GPUs are intended to perform complex calculation for 3-D Graphical rendering. Most commonly CPUs contains 4 to 8 cores with flexibility compared to GPUs, which contains thousands of cores. Currently the operating speeds of CPU's cores are about 2.3 GHz, and for GPU's are at 1GHz. Hence the GPUs are more powerful in graphical application compared to CPUs as in graphical applications more tasks are run in parallel. CPU's are most commonly used for sequential tasks, whereas GPU's are used for parallel tasks. The evolution of GPUs from basic one to the modern one is well explained in the paper [1]

**Revised Manuscript Received on July 22, 2019.**

**C. Kanagasabhapati**, VLSI design and Embedded Systems, UTL Technologies Ltd, Bangalore, Karnataka, India.

**Siddamma**,VLSI design and Embedded Systems, UTL Technologies Ltd, Bangalore, Karnataka, India.

**Dr. Siva S Yellampalli**, VLSI design and Embedded Systems, UTL Technologies Ltd, Bangalore, Karnataka, India.

The reference [2]-[4] provides the detail description about the design of Vertex Shader. To meet the high speed and low power graphics rendering, vertex shader is used and it gives improved performance per vertex operations. The Shader core is group of programmable and fixed functional blocks. Shader is a user-defined program which is used to carry out some mathematical operations through some stages of graphical rendering pipeline. There are many types of shaders, but most important ones are vertex shader and pixel shader [5].

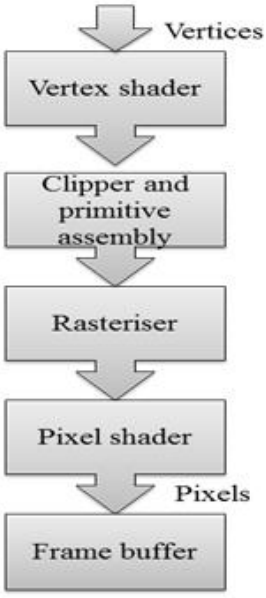


**Fig.1: Programmable Graphics Pipeline**

Graphics processing unit (GPU) are most widely used as a coprocessors in personal computers as well as in some server machines to accelerate 3D graphics operations.

Graphical rendering is the process of generating 2-D or 3-D images from the model by using computer programs. The above fig.1 shows the basic programmable graphics pipeline. The vertices are given sequentially as an input to the vertex shader. All the vertices manipulation algorithms are performed inside the Vertex shader. Then the vertex shader output are assembled and fed to the next stage of the graphics pipeline. The next stage in the rendering pipeline is Rasterizer stage, which converts or generates fragments, and these fragments are sent to the pixel shader. The pixel shader manipulates pixels and stored the results at the frame buffer [6].

The shader program may be simple one or complicated depends on the visual effect we are trying to

achieve. The most critical path in the shader core is the process time of a pixel. In this paper we have designed programmable pixel shader computing unit using Verilog HDL and results were simulated by CADENCE NcSim Simulator.

## II. PIXEL SHADERS

The architecture of Pixel Shader with its registers is shown in fig.2. Here the pixel shaders are used to manipulate the Pixels taken from the rasterizer stage in the graphical rendering pipeline.

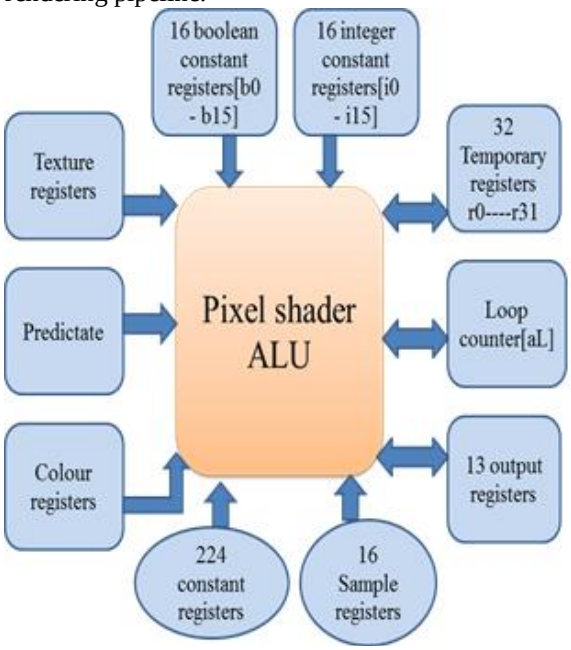


**Fig.2: Pixel Shader Block Diagram**

As we already explained the Vertex shader is used to manipulate Vertices of any images to create some visual effects like flapping flag or flowing clothes. These manipulated data fed further in the graphics pipeline. The last stage will be the pixel shader, which will be used to manipulate the pixels taken from the rasterizer stage. This Pixel shader block is also used to provide color to the manipulated pixels.

Here all the data will be done using large set of registers and not through any external memories as use of external memories will slow down the operation. In fig 2, 224 constant registers, 16-integer register, 16 Boolean register, 16 sample registers, 32 temporary registers, texture registers, address register, loop count register, predicate register, 12 color registers,13 output registers are used.

All the registers are either called by an external function (APIs) or through regular graphical program instruction. For example Defi, def, defb instructions are used for setting integer constant registers, constant registers, and Boolean constant register. Each Pixel shader registers are of the size of 128-bits, and this 128bit will be divided into 4-elements as X-Y-Z-W for processing. This registers are called as a quad vector, means they have a block of arranged as four elemental components X-Y-Z-W, each of 32-bit value as shown in fig.3.

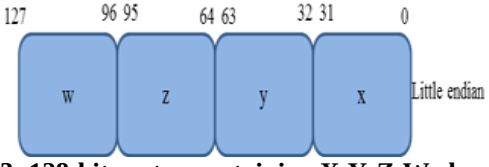


**Fig.3: 128-bit vector containing X-Y-Z-W elements.**

## III. IMPLEMNTATION

In 3-d graphics the Pixel shader ALU performs complex mathematical operations. This Pixel shader ALU takes Pixel data from the sample register and provide the possibility of creation of 64 combinations of inputs (16 sample registers and each has 4 elements).

The Pixel shader is used to provide color to the processed data. The instructions carried out by the Pixel shader ALU are mathematical operations such as branching operations, branchless operations and matrix multiplications. The mathematical operations include the addition, subtraction, cross product, dot product etc.

Here the process is based on SIMD, which means a Single Instruction Multiple Data is adopted. By giving the one instruction we can have multiple data [7].

The SIMD Consists of single CU (control unit) and multiple PEs (processing Elements).Control Unit is used to fetch the instruction from the main memory and then broadcasts the control signals to all the PEs.

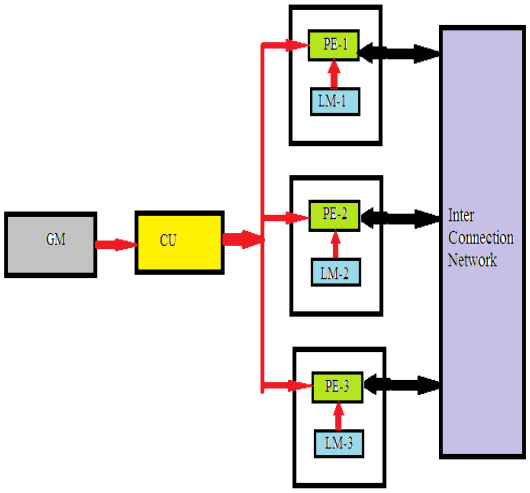


**Fig.4: SIMD**

All the Processing Elements are synchronously executing the same instruction but on the different sets of data. The execution process of pixel shader instruction using SIMD is as shown in the fig. 5.

Here the values of A and B added and then the result will be stored in the destination D.

$Dx = Ay + Bx;$
$Dy = Ay + By;$
$Dz = Ay + Bz:$
$Dw = Dw;$

The following instructions were implanted:

**Mathematical instructions** like add(addition), sub(subtraction), mu(multiplication), mad(multiply add), dp3(dot product of 3-vectors), dp4(dot product of 4-vectors), dp2add(2-d

dot product) and crs(cross product).

**Branching instructions** like if, else, end-if, loop, predicate, call, callnz, break, dsx(x rate of change), dsy(y rate of change).

**branchless instructions** like abs(absolute), max(maximum), min(minimum), slt(set if less than), sge(set if greater or equl), sgn(sign).

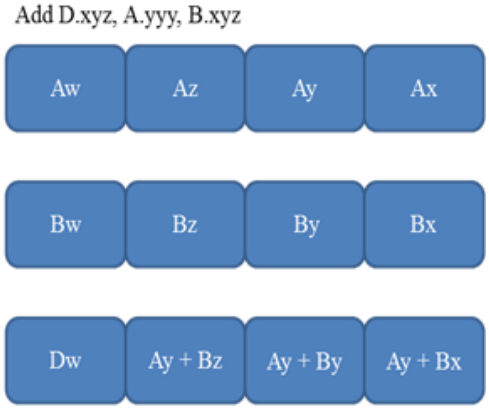


**Fig.5: SIMD operation**

**Special instructions** like log(Logarithmic value with base 2), pow (power), exp (exponential), nop ( no operation), lrp(linear Interpolation).

**Data conversion Instruction** like mov ( copying vector value to the register), mova ( copying a data from register to register).

**matrix multiplication instructions** m4x4(Applying 4X4 matrix to vectors), m4x3(Applying 4X3 matrix to vectors), m3x4(Applying 3X4 matrix to vectors), m3x3(Applying 3X3 matrix to vectors), m3x2(Applying 3X2 matrix to vectors).

The designs of hardware for all these possible instructions of Pixel shader were implemented using Verilog HDL. All these instructions are micro coded and implemented inside a single module. Decoder logic used to execute each of the instruction using a select line to select a particular instruction.

## IV. RESULTS

The following fig.6-10 shows the simulation results of pixel shader. The instructions are designed using Verilog HDL and the design is simulated by using Cadence simulator. The Fig. 6 shows the number of inputs to the pixel shader. Fig.7, 8, 9, 10 shows the output waveforms of Pixel shader ALU.

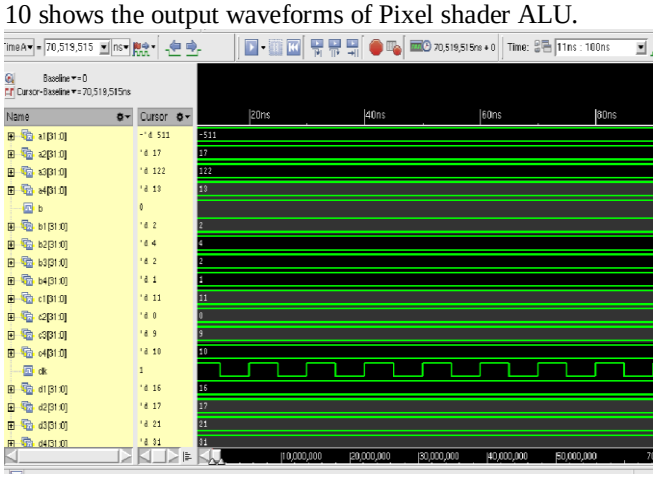

**Fig.6: pixel shader input**

The design of instructions are simulated for add, sub, crs, dp3, dp4, mul, mad, dp2add, dsx, dsy, irp, log, exp, pow, nop, abs, max, min, sge, sgn, slt, pred, if, else, end, call, callnz, loop, break, end-if, m3x2, m3x3, m3x4, m4x3, m4x4 and so on.

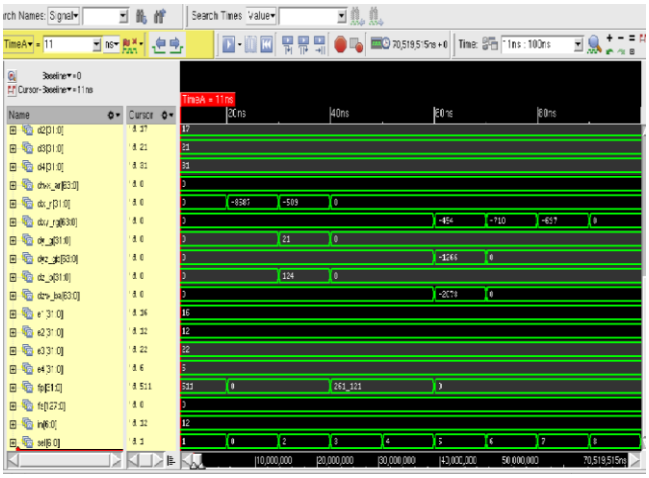

**Fig.7: pixel shader output**

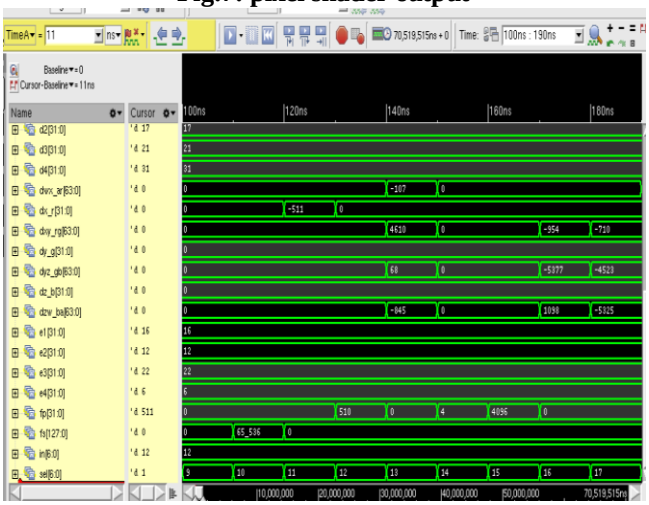

**Fig.8: pixel shader output**

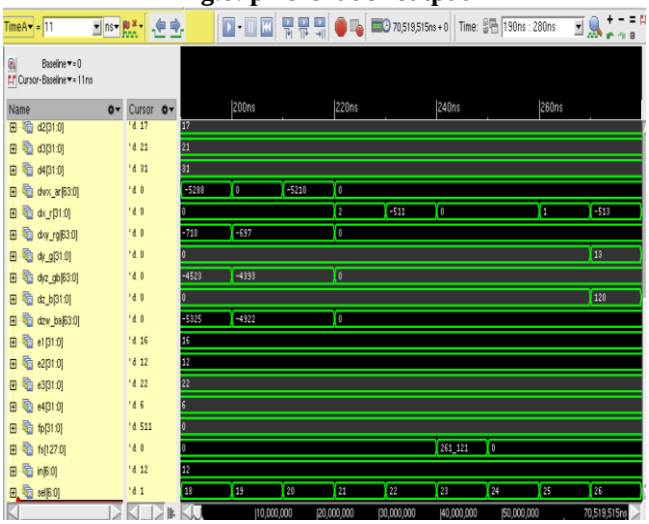

**Fig.9: pixel shader output**

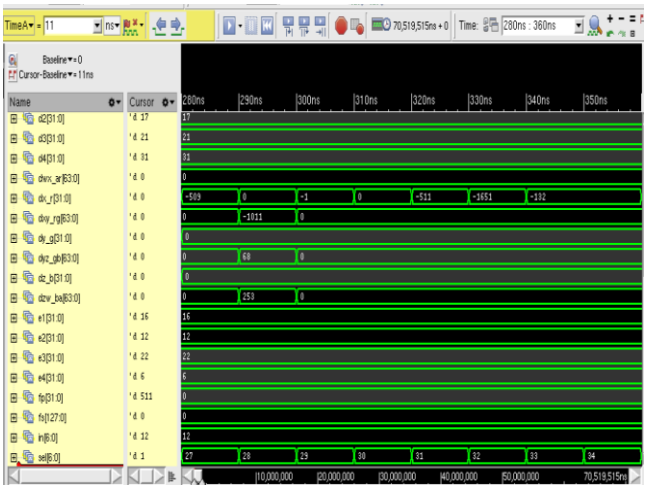

**Fig.10: pixel shader output**

## V. CONCLUSION

This paper provides the overall information about the design of programmable Pixel shader instruction, registers, and the SIMD concept. The design of programmable pixel shader computing unit is done by using Verilog HDL. Using these instructions the programmer can customize the application according to the user need. Because of this programming flexibility, better graphical solution can be obtained. Hence, the programmable shader will achieve a better efficiency compared to the fixed functional block Pixel Shader. Due to the concept single instruction and multiple data the execution will be done in parallel and faster. Here all the possible instructions of and Pixel shader are designed and the results are simulated using Cadence NCsim simulator.

## REFERENCES


1. J. Nickolls and W. J. Dally, "The GPU Computing Era," IEEE Mirco, pp. 56-69, Mar./Apr., 2010.
2. B.-G. Nam, H. Kim, and H.-J. Yoo, "A Low-Power Unified Arithmetic Unit for Programmable Handheld 3-D Graphics Systems," IEEE Journal of Solid-State Circuits, vol. 42, no. 8, pp. 1767-1778, Aug. 2007.
3. B.-G. Nam, H. Kim, and H.-J. Yoo, "Power and Area-Efficient Unified Computation of Vector and Elementary Functions for Handheld 3D Graphics Systems," IEEE Trans. Computers, vol. 57, no. 4, pp. 490-504, Apr. 2008.
4. B.-G. Nam and H.-J.Yoo "An Embedded Stream Processor Core Based on Logarithmic Arithmetic for a Low-Power 3-D Graphics SoC," IEEE Journal of Solid-State Circuits, vol. 44, no. 5, pp. 1554-1570, May 2009.
5. By Jason Zink, Matt Pettineo, Jack Hoxley, "Practical Rendering and Computation with Direct3D 11" 2011
6. Ed Angel (University of New Mexico), Dave Shreiner(ARM) , "Teaching a Shader-Based Introduction to Computer Graphics", IEEE Computer Graphics and Application. Revised 10 Feb. 2011.
7. Shen-Fu Hsiao, Po-Han Wu, Chia-Sheng Wen, and Li-Yao Chen, "Design of a Programmable Vertex Processor in OpenGL ES 2.0 Mobile Graphics Processing Units" 2013 International Symposium on VLSI Design, Automation, and Test (VLSI-DAT)2013.
8. James C. Leiterman, "Learn Vertex and Pixel Shader Programming with DirectX® 9" 2004.
9. Yu-Jung Chen, Chao-Hsien Hsu, Chung-Yao Hung, Chia-Ming Chang, Shan-YiChuang, Liang-Gee Chen, and Shao-Yi Chien, "A 130.3 mW 16-Core Mobile GPU With Power-Aware Pixel Approximation Techniques", IEEE Journal Of Solid-State Circuits, Vol.50, No.9, September 2015.
10. Ian Bratt, "The ARM® Mali-T880 Mobile GPU", 2015 IEEE Hot Chips 27 Symposium (HCS), 2015.
11. Kwang-Yeob Lee, Tae-Ryoung Park, Jae-Chang Kwak, Yong-Seo Koo, "A Design of Multi-threaded Shader Processor with Dual-Phase Pipeline Architecture", 2009 First International Conference on Advances in Multimedia 2009.


## AUTHORS PROFILE

**Mr. C. Kanagasabapathi** obtained his M.Tech from Visvesvaraya Technological University. He is currently with VTU Extension Centre, UTL Technologies Ltd. He worked on a broad range of research topics including Very Large Scale Integration (VLSI), High voltage Instrumentation, EMI & EMC and high performance computing. He has published multiple journal papers & IEEE Conference papers in these areas of research.

**Ms. Siddamma Biradar** obtained her M.Tech from Visvesvaraya Technological University. She is currently with 4semi Technology India pvt ltd. Her research interest is VLSI circuits and Digital design.

**Dr. Siva Yellampalli** obtained his MS & Ph.D from Louisiana State University. He is currently with VTU Extension Centre, UTL Technologies Ltd. He worked on a broad range of research topics including Very Large Scale Integration (VLSI), mixed signal circuits/systems development, micro electromechanical systems (MEMS), and integrated carbon nanotube based sensors. He has published a book in the area of mixed signal design, and edited two books on carbon nano tubes. He also published multiple journal papers & IEEE Conference papers in these areas of research.