Full System Simulation with QEMU: an Approach to Multi-view 3D GPU Design

Shye-Tzeng Shen, Shin-Ying Lee, and Chung-Ho Chen
Dept. of Electrical Engineering and Inst. of Computer & Communication Engineering
National Cheng Kung University
Tainan, Taiwan
{shen, sing}@casmail.ee.ncku.edu.tw, chchen@mail.ncku.edu.tw

***Abstract*—Hardware-and-software full system co-verification and co-simulation in the early stage of SoC development, i.e., before HDL code synthesis, is usually a big challenge for design engineers. In this paper, we propose a QEMU-based full system simulation framework to tackle the problem faced with the design of an embedded multi-view 3D GPU (graphic processing unit). Through the framework, we are able to extensively explore the multi-view GPU architecture, and at the same time software designers can develop and debug the associated device drivers and graphics applications by simulating the GPU design in full system operation. This approach greatly improves the ESL design process and shortens the development time for the complex multi-view GPU system.**

I. INTRODUCTION

When developing a complex SoC system such as a multi-view GPU, the verification task of the hardware architecture and software algorithm often takes most of the development time. It's a big challenge for engineers to develop the system without adequately efficient simulation and verification methodologies. Recently, using an electronic system level (ESL) design flow and transaction-level modeling (TLM) models has become a popular approach to perform hardware-and-software co-verification in the early development stage [1]. Nevertheless, it's still very slow and time-consuming to simulate the whole system, including device drivers, operating system (OS), and software applications (APs) by using current commercial ESL tools.

Specifically, the SystemC instruction set simulator (ISS) module usually expands an enormous amount of time in booting up the OS. A designer may take more than half an hour just waiting for the OS to come up before he or she can actually start the debugging task for the RTL modules.

An embedded multi-view GPU design is such a system which requires full system simulations of the GPU's SystemC and/or RTL codes with the application graphics benchmarks running on the operating system (Linux in this case). In this paper, we propose an efficient hybrid full system simulation platform to meet the ESL simulation need of the multi-view GPU system. The rest of this paper includes the following sections. Section II highlights the hybrid full system simulation framework. Section III gives the overview of the multi-view GPU simulation platform and Section IV presents the multi-view GPU architecture. Some evaluation results using the hybrid simulation system are presented in Section V. Finally, a brief conclusion is given in Section VI.

II. HYBRID FULL SYSTEM SIMULATION FRAMEWORK

Fig. 1 shows the hybrid full system simulation framework which consists of the QEMU virtual machine [2], the QEMU-SystemC patch [3], and a commercial ESL tool. The QEMU virtual machine is a high-speed functional simulator that executes the software programs such as the OS, device driver, and graphics APs for the target multi-view GPU design which resides in the commercial ESL tool environment. The QEMU-SystemC is a patch to QEMU to provide a SystemC wrapper for hardware simulation. The ESL tools we use include CoWare PA [4] and ModelSim [5], one for high-level SystemC simulation and the other for HDL simulation.

For the original QEMU-SystemC wrapper, it only provides a single slot for system-C hardware module, so that it is not suitable for simulating a complex SoC system and cannot involve to HDL code simulation. Since we look forward to simulating the full system with the elaborate GPU module not only for SystemC but also RTL models, a more flexible, compatible, and efficient bridge interface is needed. In our scenario, to form an ARM AMBA-based system [6], we have developed and improved the QEMU-SystemC bridge interface which connects the local bus in the QEMU side to the AHB bus of the ESL tool side both for SystemC and HDL simulator. Because the TCP/IP stack is a highly flexible and universal communication protocol, it's chosen to achieve the new bridge interface. By implementing the new bridge interface through TCP/IP sockets, the QEMU virtual machine and the commercial ESL simulator can run in different host machines. Consequently, the simulating tasks can be accelerated by distributing the computation more efficiently.

In this way, we have logically created a full system which

This work was sponsored and supported in part by the Himax Technologies Inc. and partly by the National Science Council of Taiwan under grand NSC 96-2221-E-006-192-MY3.

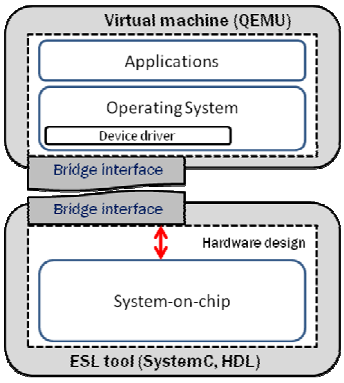


Figure 1. Structure of the QEMU-SystemC hybrid full system simulation platform

is able to simulate the processor, OS, device drivers, as well as graphics benchmarks (functionally simulated by the QEMU virtual machine) and at the same time simulate the multi-view GPU either in highly abstract SystemC modules or in synthesizable RTL modules. In this system, through the bridge interface, the QEMU virtual machine simply treats the design in the ESL tool as a general hardware device on its local bus system. Therefore, the programs executed on the QEMU virtual machine are allowed to access and control the hardware module via the load/store instructions of the processor directly.

With this hybrid simulation methodology, developers can emulate the processor and the software run upon it in a much higher speed and verify the interactions between the software programs and the hardware modules before a synthesizable HDL code is ready. The designer in this case only needs to wait less than one minute for the OS to boot up for full system simulations compared to half an hour. In addition, because the QEMU virtual machine has already provided a channel to connect with GNU GDB debugger, software engineers can start to design and debug system programs and APs in the early stage of system development projects. Since all of this happens as early as the project begins, this hybrid full system simulation platform can also help in smoothing the task of building an actual hardware prototype, such as an FPGA verification system, prior to chip fabrication.

III. Multi-View GPU Simulation Platform Overview

In this section, by using the previous hybrid full system simulation framework, we present the methodology of developing a multi-view 3D GPU built out of a tile-based rendering architecture [7]-[14]. Fig. 2 shows the data processing flow of a multi-view graphics system intended for use in auto-stereoscopic displays. In order to be used in cross platforms and different OSs, the multi-view GPU design follows the standard application programming interface (API) of the OpenGL ES (open graphics library for embedded system) [15].

The multi-view GPU processor consists of three pipelined processing units, that is, the geometry engine (GE), the

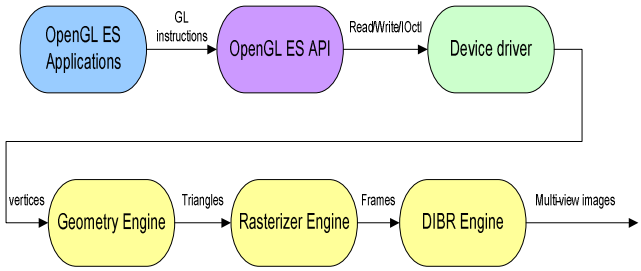


Figure 2. Data flow of the multi-view GPU

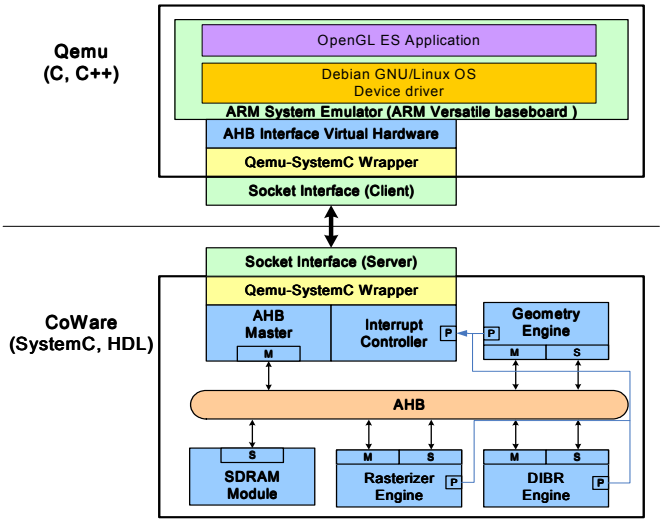


Figure 3. Full system simulation for the multi-view GPU scenario

rasterizing engine (RE), and the Depth Image-Based-Rendering (DIBR) unit which generates the multi-view graphics for a 3D auto-stereoscopic display [16]-[17]. Fig. 3 shows the system architecture of the design in the hybrid full system simulator. As it can be seen, the graphics applications and the simulated OS both run on the ARM-based virtual machine within the QEMU simulator. This part of the simulation is performed in terms of functional accuracy as our primary interest here is the functional verification of the system software, the implemented graphics APIs, and the GPU drivers.

Through the AHB interface virtual hardware, the drawing commands, for instance, from the graphics application can be issued to the GE, RE, and DIBR hardware modules in the CoWare ESL simulator. Note that with this environment, we can explore both the functional models of the multi-view GPU in the early stage of the development as well as the synthesizable RTL models when they are ready, all in the capacity of efficient full system simulations.

IV. Multi-View GPU Architecture

Fig. 4 presents the overall hardware architecture of the multi-view 3D GPU design. The DIBR module takes the

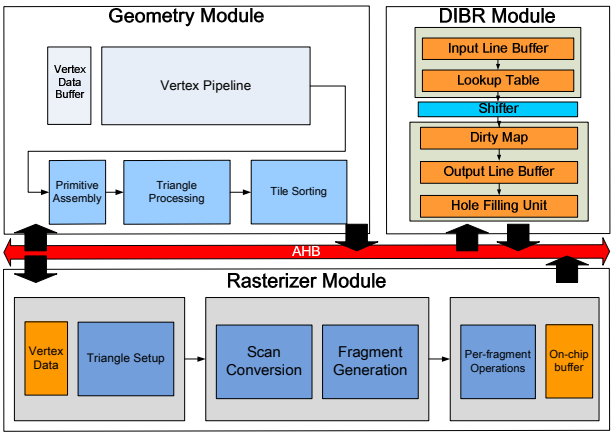


Figure 4. Multi-view GPU architecture

TABLE I. THE VERIFICATION ENVIRONMENT

Power MBX SDK	Microsoft Visual Studio 2005
SystemC Development	GCC 4.1.2
	SystemC v2.0.1
Full System Simulation	QEMU v0.9.1 with Linux kernel v2.6.18-6 for ARM Versatile-PB
	CoWare PA v2007.1.2
	ModelSim 6.3

TABLE II. MULTI-VIEW GPU EVALUATION PARAMETERS

Parameter	**Setting**
Tile Size	32x32
Resolution	640x480
Frequency	200MHz
No. of Views	2
Pixel Format	RGBD 8888
Bus Architecture	ARM AHB 2.0 with 32-bit width
Bus Arbitration Scheme	Fixed priority
Local Frame Buffer	2-cycle latency (SRAM)

output of the RE unit, including the color map and the depth map of the frame to generate the required multi-view images. We have investigated the following architectural issues to improve the performance and reduce the memory traffic for this multi-view GPU design:

- Profiling graphics benchmarks and their processing requirements due to various levels of details in the workload.
- Quantifying the effect of various tile-sizes on the memory traffic.
- Exploring various fixed point data formats for quality graphics
- Automatic adjustment of the shifted pixel number for different displays that have various pixel pitches in multi-view image synthesis.

V. EVALUATION RESULTS

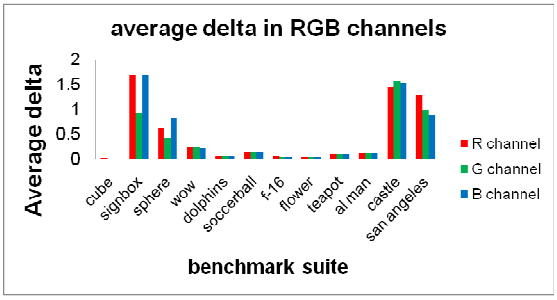
We have implemented the multi-view GPU module in both SystemC TLM and Verilog HDL model and evaluated in the hybrid full system simulation platform. The work includes porting the OpenGL ES library and designing the device drivers of the multi-view GPU module onto the ARM Linux system within the QEMU virtual machine. Table I shows the verification environment for our multi-view GPU design, including a reference suite of the OpenGL ES pipeline implementation for comparison. Table II shows the simulation parameters used to present the results in this paper. The local AHB bus uses fixed priority for bus arbitration although using other types of the arbitration policy is not excluded for evaluation.

$$average\_delta = \frac{\sum_{i=0,j=0}^{i,j=resolution} \left| C_{S_i} - C_{r_j} \right|}{resolution} \qquad \text{Equ.(1)}$$

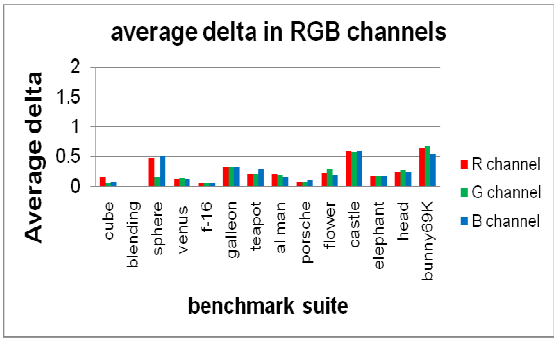
To quantify the fixed-point data format on graphics quality and our GPU optimization strategies which include abandoning small area of triangles for processing and various anti-aliasing schemes used, we define the average pixel data difference between the image generated by our graphics GPU and the image produced by the reference implementation as shown in Equ.(1). Specifically, Equ.(1) represents the average pixel errors between the images produced by our GPU scheme and by the PowerVR MBX SDK [18], where C_s is the pixel value of image generated by our GPU model and C_r is the pixel value of image from PowerVR.

Fig. 5(a) shows the average pixel errors in RGB channels respectively for the SystemC TLM design. Most of the errors are very small; only some of them have a diffence value of about 1.5. Note that 1.5 out of 255 in pixel value difference is quite small because this tiny error shift will not be recognized by human vision in general. Traditional PSNR metric used in video quality evaluation is not suitable for graphics since there is no compression here. Fig. 5(b) shows the average pixel errors of our Verilog HDL model which uses fixed-point data format and our SystemC model which uses floating-point data format. The purpose of this comparison is to evaluate the chosen fixed-point data operation on the quality of the generated graphics. The result indicates that the images generated by the fixed-point GPU have very little variations compared with their floating-point counterpart.

One of the critical factors in tile-based rendering is the tile size to be used since for embedded applications this first means the on-chip buffer cost and secondly it affects the external memory traffic coming from the GPU. Reducing the memory traffic directly reduces the power consumption. Fig. 6 shows the data traffic reduction ratio of the RE module for



(a) Average delta error of SystemC TLM model to reference design



(b) Average delta error of Verilog HDL model v.s SystemC TLM Model

Figure 5. Comparisons of average pixel values

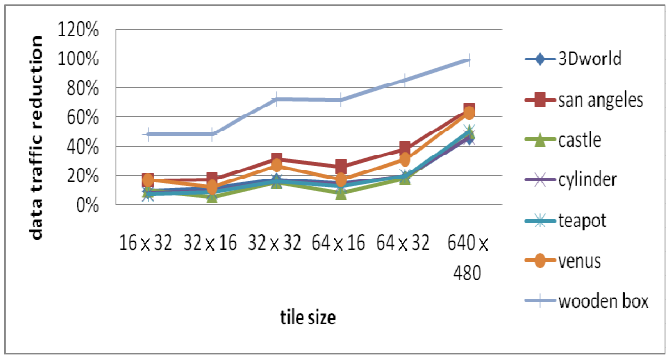


Figure 6. Data traffic reduction compared with 16x16 tile size

different tile size used compared with the traffic of the baseline tile size, 16x16. The 640x480 case is used for comparison; assuming perfect on-chip buffer size which equals the frame resolution. From the result, we note that when using a 64x32 tile, the reduction ratio only grows up around 5% than the size of 32x32 case, but using twice the buffer size. Hence, by trading-off the local frame buffer size and the amount of data traffic reduction, we select the tile size of 32x32 for our GPU design.

VI. CONCLUSIONS

We have presented a high performance full system simulation platform with the QEMU-SystemC virtual machine and used it to show the design of a multi-view 3D GPU. By means of this virtual platform, developers are able to perform hardware verification as well as software debugging efficiently in addition to high-level architecture explorations. This hybrid full system simulator allows the system engineers to evaluate and analyze the system for both the hardware architecture and software algorithm in the early stage of design either in SystemC TLM models or HDL models. Through the simulator, we have introduced the pixel value difference metric to quantify the graphics quality of the GPU design, when either referring to the reference implementation, or to the floating point version. The result shows that only small differences in pixel value are found for the fourteen graphics benchmarks that have various levels of details.

REFERENCES

[1] Brian Bailey, Grant Martin, and Andrew Piziali, "ESL design and verification: a prescription for electronic system level methodology," Morgan Kaufman/Elsevier, 2007.

[2] Fabrice Bellard, "QEMU, a fast and portable dynamic translator," USENIX Annual Technical Conference, 2005.

[3] Monton, Marius Portero, Antoni Moreno, Marc Martinez, Borja Carrabina, Jordi CEPHIS, "Mixed SW/SystemC SoC emulation framework," IEEE Symposium on Industrial Electronics (ISIE), 2007.

[4] CoWare Inc., "CoWare platform architect," http://www.coware.com.

[5] Mentor Graphics, "ModelSim-advanced simulation and debugging," http://www.model.com.

[6] ARM Ltd., "AMBA specifications," http://www.arm.com.

[7] H. Fuchs et al., "Pixel-planes 5: a heterogeneous multiprocessor graphics system using processor-enhanced memories," Proc. Ann. Conf. Computer graphics and interactive techniques (SIGGRAPH'89), pp. 79-88, July 1989.

[8] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs, "A sorting classification of parallel rendering," IEEE Computer Graphics and Applications, vol. 14, no. 4, pp. 23-32, July 1994.

[9] M. Cox and N. Bhandari, "Architectural implications of hardware-accelerated bucket rendering," Proc. on ACM SIGGMPH/ EURDGRAPHICS Workshop on Graphics Hardware, pp. 25-34, 1997.

[10] M. Chen, G. Stoll, H.Igehy, K. Proudfoot , and P. Hanmahan, "Simple models of the impact of overlap in bucket rendering" Proc. on ACM SIGGMPH/EURDGRAPHICS Workshop on Graphics Hardware, pp. 105-112, 1998.

[11] I. Antochi, B. Juurlink, S. Vassilidadis, and P. Liuha, "Memory bandwith requirements of tile-based rendering, " in Proc. of the Third and Fourth International Workshops SAMOS 2003 and SAMOS 2004 (LNCS 3133), pp. 323-332, July 2004.

[12] J. Hasselgren and T. Akenine-Möller, "An efficient multi-view rasterization architecture," Proc. on Eurographics Symposium Rendering, pp. 61-72, 2006.

[13] A. Kalaiah and T.K. Capin, "A unified graphics rendering pipeline for autostereoscopic rendering," IEEE 3DTV Conference, May 2007.

[14] T. Capin, K. Pulli, and T. Akenine-Möller, "The state of the art in mobile graphics research," IEEE Computer Graphics and Applications, vol. 28, no. 4, pp. 63-73, July/August 2008.

[15] A. Munshi and J. Leech, "OpenGL ES common/common-lite profile specification version 1.11.10," Khronos Group, April 2007.

[16] C. Fehn, "A 3D-TV approach using depth-image-base rendering," Proc. of Visualization, Imaging, and Image Processing, pp. 482-487, September 2003.

[17] C. Fehn, R. Barre, and S. Pastoor, "Interactive 3-DTV--concepts and key technologies," Proc. of the IEEE, vol. 94, No. 3, pp. 524 - 538, March 2006.

[18] Imagination Technologies Ltd., "PowerVR technology overview," 2004