SYSTEMC MODELING FOR 3D GRAPHICS HARDWARE ACCELERATION

Tse-Chen Yeh, Ruei-Ting Gu, Hung-Yu Chen and Ing-Jer Huang

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan

ABSTRACT

Due to the growing complexity of system architecture current today, the system modeling is utilized in the early design under the time-to-market. To establish a 3D graphics acceleration platform, we need to explore the system architecture and performance issues. In this paper, we use SystemC to model our hardware models including accelerator, SDRAM memory, and graphics engine (GE) for investigating hardware and software implementations. And the experimental results give the open issues of performance tuning and architecture exploration on 3D graphics acceleration on embedded system.

1. INTRODUCTION

3D graphics applications become popular on mobile devices and other embedded systems. To build the system consists of the HW/SW co-development, and the time-to-market is critical in the conventional design flow. RTL simulation slow down the hardware development and the software utility must to wait the usable hardware prototype. On the stage of system integration, we will face the performance tuning and architecture adaptation, however, it is impossible for porting huge software using RTL simulation.

Our goal is to apply 3D graphics on digital television (DTV) system in real time. In order to establish the acceleration platform in time, we use system modeling to speedup the simulation time and explore the performance issues of whole system. On the other hand, the abstraction level which we called TLM (Transaction Level Model) can reduce the simulation time by omitting the model details of hardware implementation. We choose SystemC for modeling language and the CoWare™ Platform Architect for the analysis toolkit on performance tuning.

The organization of this paper is as follows. In Section 2, the related works are presented, which includes the SystemC modeling and basic knowledge on 3D graphics pipeline. The specification analysis is described in Section3. In Section 4, we present the modeling flow and implementation details. And we compare the hardware and software simulation results in Section 5. Finally, the future works and open issues for our research are summarized in Section 6.

2. RELATED WORKS

2.1 System modeling using SystemC

SystemC dominates the properties of language on system modeling. In practice, SystemC created into the form of class library in C++, it can be integrated with software function block as simulation. The kernel of SystemC is a process scheduler to handle the concurrency mechanism which is the significant feature of hardware behaviour [1][2].

TLM is the medium layer on hardware modeling, which higher than register transfer level (RTL), but lower than algorithmic level, the classifications of TLM are presented by [3]. By omitting the pin-accurate details, TLM economizes large simulation time which separate hardware model into computation and communication. S. Pasricha et al. extended the TLM approach to model a SoC platform for architecture exploration [4].

In terms of Platform Architect utilizes the model library for the platform building based on ARM cores, the library consist of Embedded Processor Model (EPM) and Transactional Bus Simulator (TBS) [5-6]. The EPM embedded the ARM instruction set simulator (ISS) for run-time software simulation. Besides, the TBS is built into a hierarchical channel with arbitration module and address decoder. Inside the TBS also contain the analysis library which can record the transaction information for bus performance analysis.

2.2 3D graphics pipeline

OpenGL ES specification provided by the Khronos Group specifies the 3D graphics pipeline implemented on embedded system which shows in Fig. 1 [7]. This spec. defines the order of primitive processing and API which interfaces between the users and OS/driver.

![Fig. 1: 3D graphics pipeline in OpenGL ES 1.1](image-url)
Generally, we separate the 3D graphics pipeline into two processing functionalities, one is the geometry processing, the other is the rendering processing. The former is responsible for coordinate transformation and lighting, and the later processes the rasterization, texture mapping and other rendering effects. In [8], Inho Lee et al. proposed Hardware Description Macro (HDM) based on C++ to model the 3D graphics platform adapting OpenGL, which modified the order of 3D pipeline to improve the software performance. In 2001, PowerVR™ and ARM™ proposed MBX™ core for 3D graphics acceleration [9], which is one of the hardware solution with high performance. In this paper, we target the low cost solution and utilize TLM to model our 3D graphics hardware platform and explore trade-off on HW/SW implementation.

### 3. SYSTEM SPECIFICATION

#### 3.1 Specification analysis

In the beginning of our design flow, we surveyed several the-state-of-the-art products to specify our 3D graphics hardware specification. Due to the different technology utilization form our targets, we formalized the technology into .13µm for finding Pareto-optimal point showed in Table 1. To suit for embedded system requirements, the cost must be down. We press the gate count as small as possible under the performance constraint. In terms of performance, we permit lower performance than MBX core to earn the smaller gate count. Notice about the falanx™ Mali serials, Mali55 and Mali110 only include the geometry processing, like the MaliGP only contains the rendering functionality. Therefore, we specified the performance of our 3D graphics accelerator reaches 1M triangle per second with no exceeded 400K gate count.

#### 3.2 System architecture

For the integration consideration, we employ AMBA bus to be the system bus with 32 bit width, thus our accelerator can be integrated completely by wrapping the AMBA bus interface (BI). Moreover, we target ARM926EJ-S to be the core of the 3D graphics platform. For the low cost, the memory utilized SDRAM for external memory, which stores all primitives during 3D graphics processing. Frame buffer also allocated in external memory for Graphics Engine (GE) access. Graphics Engine is used to read frame buffer allocated in SDRAM, and to draw fragments on the display. From the system block diagram showed in Fig. 2, we have notice that the performance bottleneck will fall on the memory bandwidth, because the ARM9 core, 3D graphics accelerator and the GE will content the bus utilization.

### 4. SYSTEM MODELING

#### 4.1 TLM API

For more accurate analysis, we model 3D graphics platform via Platform Architect. EPM provided by Model Library can handle the software simulation by loading the program, and TBS record the transaction information for each transfer. As section 2.1 mentioned, TLM separate the hardware model into computation and communication. In order to integrate hardware model with the EPM and TBS for simulation, the communication block of our hardware model needs to conform to the TLM API pre-defined by Platform Architect.

TLM API defines several port classes used to connect with AMBA TBS, one is the initiator port used to start a transaction, and the other is the target port which can only receive the transaction. The initiator port use getTransaction() to obtain the transaction object, and exploit the sendTransaction() to transfer after setting the attributes. Through setting the attributes in a transaction, the TBS will record these transfers for statistics. Hence, we have to modify our SystemC model for building the platform by going through the three steps:

1. Change the port class which connect to AMBA TBS.
2. Replace the original sensitive list of the SystemC process/thread by the event finder defined by TLM API.

Table 1: The-state-of-the-art specifications of 3D graphics hardware acceleration

<table>
<thead>
<tr>
<th>Company</th>
<th>Chip Name</th>
<th>Clock frequency</th>
<th>Gate count</th>
<th>Performance (triangle/sec)</th>
<th>Power efficiency (mW/MHz)</th>
<th>Display resolution</th>
<th>Technology normalize</th>
</tr>
</thead>
<tbody>
<tr>
<td>falanx™</td>
<td>Mali55</td>
<td>200MHz</td>
<td>190K</td>
<td>1M</td>
<td>0.4</td>
<td>320×240</td>
<td>.13µm</td>
</tr>
<tr>
<td></td>
<td>Mali110</td>
<td>200MHz</td>
<td>230K</td>
<td>5M</td>
<td>0.5</td>
<td>320×240</td>
<td>.13µm</td>
</tr>
<tr>
<td></td>
<td>MaliGP</td>
<td>150MHz</td>
<td>150K</td>
<td>5M</td>
<td>0.3</td>
<td>320×240</td>
<td>.13µm</td>
</tr>
<tr>
<td>TAKUMI™</td>
<td>TAKUMI plus</td>
<td>40MHz</td>
<td>160K</td>
<td>0.27M</td>
<td>0.7</td>
<td>512×512</td>
<td>.13µm</td>
</tr>
<tr>
<td>Faraday™</td>
<td>FG300</td>
<td>115MHz</td>
<td>600K</td>
<td>4.31M</td>
<td>0.5</td>
<td>512×512</td>
<td>.13µm</td>
</tr>
<tr>
<td>PowerVR™</td>
<td>MBX R-S</td>
<td>120MHz</td>
<td>365K</td>
<td>1.5M</td>
<td>0.57</td>
<td>320×240</td>
<td>.13µm</td>
</tr>
<tr>
<td></td>
<td>MBX HR-S</td>
<td>120MHz</td>
<td>870K</td>
<td>3.75M</td>
<td>1.13</td>
<td>320×240</td>
<td>.13µm</td>
</tr>
<tr>
<td>Proposed HW spec.</td>
<td></td>
<td>200MHz</td>
<td>400K</td>
<td>1M</td>
<td>1.1</td>
<td>640×480</td>
<td>.13µm</td>
</tr>
</tbody>
</table>
(3) Package the data which will send to the AMBA bus into a transaction.

To illustrate the modeling step, the code segments of initiator port are showed in Fig. 3. Notice that the bold type expressions are defined by TLM API of Platform Architect. Fig. 3 (a) shows the SC_METHOD process sensitive to the event of sending address by calling getSendAddrTrfEventFinder(), which will obtain a pointer derived from sc_event_finder [2]. In Fig. 3 (b), the code segment with prefix of set is used to specify the attributes of a transaction arranged by getTransaction() and sendTransaction(). In this transaction, the read/write address has been set start from 0x0, and the data width is 32 bit. The pre-defined tlmWriteAtAddress indicates that the transaction is a write access, and the tlmSingle means the transaction is under single transfer mode. Finally, the 0x3E will be the write data by calling setWriteData().

![Code segment using burst mode transfer exploiting TLM API.](image)

4.2 Hardware modeling

To build 3D graphics platform, we need to model the hardware model surrounded by dotted line in Fig. 2. We separate the 3D graphics pipeline into geometry module and rendering module that will be assisted for individual performance estimation. Also, we need to establish the SDRAM model combined with memory controller, because built-in memory model is based on the SRAM latency. Additionally the GE has to be modelled into a bus master, which will access the frame buffer allocated in SDRAM in period. By the display speed limitation, the data buffer is consumed by 25MHz in contrast of the system clock rate 200MHz.

To economize the memory bandwidth, we enforce the 3D graphics processing data into 32 bit package at first. Further, the most significant is to utilize the burst transfer. To model the burst mode via API, we modify the group attribute of the transaction illustrated in Fig. 4. We must to give a counter, e.g. burst_cnt, to accumulate the burst length, and we will increase the address by 4 for successive access because of our bus width. In the example, io_mst is an initiator port class, the burst length is 8 beat, and tlmIncremental implies the transaction is a successive access.

```cpp
void Ctrl::Tranx() {
    //--- Get, setup, and send transaction
    dout.getTransaction();
    dout.Transaction->setAddress(0);
    dout.Transaction->setGroup(AMBA::tlmSingle);
    dout.Transaction->setAccessType(AMBA::tlmWriteAtAddress);
    dout.Transaction->setWriteData(0x3E);
    dout.sendTransaction();
}
```

4. EXPERIMENTAL RESULTS

5.1 Software simulation

For comparisons of the HW/SW performance, we build a simple 3D graphics pipeline by pure software, which renders 100 cubes for 1 frame, and each cube has 12 triangles constructed by 36 vertexes. Our simple pipeline includes viewing transform, lighting and perspective transform in geometry processing, and tile divider and tile-based rendering in rasterization block. Also notice that rendering block adopts the tile-based rasterization for preserving the bus bandwidth, and appending with SDRAM latency. The software simulation results expressed in Fig. 5 and Table 2. To compare with our objective performance, the software implementation can only reach 87.43 triangles per second. To fill with the huge gap on performance, both geometry and rendering processing need to be
accelerated by hardware implementation. Even the geometry processing only occupies 6.8% execution time.

<table>
<thead>
<tr>
<th>Software Function</th>
<th>Execution time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame buffer clear</td>
<td>0.0887</td>
</tr>
<tr>
<td>Geometry</td>
<td></td>
</tr>
<tr>
<td>Viewing transform</td>
<td>0.1234</td>
</tr>
<tr>
<td>Lighting</td>
<td>0.6178</td>
</tr>
<tr>
<td>Perspective transform</td>
<td>0.1938</td>
</tr>
<tr>
<td>Tile divider</td>
<td>0.0384</td>
</tr>
<tr>
<td>Rendering</td>
<td>12.6558</td>
</tr>
<tr>
<td>Total execution time</td>
<td>14.6589</td>
</tr>
</tbody>
</table>

5.2 Hardware modeling simulation

We exploit the same software benchmark as mentioned in section 5.1 but drawing 4 frames. We also replace the geometry and rendering function by load/store instructions at the hardware memory-map address. The simulation results show in Fig. 6 by 100,000 cycle sample rate and the simulation time spent approximately 15–20 minutes. Each frame consists of frame buffer clear, tile divider, geometry and rendering processing. And the GE always occupied 10% bus performance according to the cyclical access. From the result, we discover that frame buffer clear and tile divider which implemented by software becomes the performance bottleneck after accelerating geometry and rendering processing into hardware. During the frame buffer clear, the SDRAM will be accessed in sequential, so the SDRAM bandwidth has the better performance remain 52%. In contrary, the tile divider access the non sequential address, the performance will down to 37% in average due to the internal latency of row and bank changes in SDRAM.

<table>
<thead>
<tr>
<th>3D graphics pipeline</th>
<th>Execution time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame buffer clear (SW)</td>
<td>0.0999</td>
</tr>
<tr>
<td>Geometry module (HW)</td>
<td>0.0007</td>
</tr>
<tr>
<td>Tile divider (SW)</td>
<td>0.0536</td>
</tr>
<tr>
<td>Rendering module (HW)</td>
<td>0.0021</td>
</tr>
<tr>
<td>One frame</td>
<td>0.1553</td>
</tr>
</tbody>
</table>

On the performance issue, we also take a look at the execution time of the simulation showed in Table 3. To compare with the pure software solution showing in Table 2, frame buffer clear and tile divider have increase 11.5% and 39.7% performance penalty by the GE integrated. On the part of geometry processing gain 1280 fold and the rendering block promote 6098 times on execution time. The system performance has promoted 88.38 times, which can render 7727.6 triangles per second.

6. CONCLUSION

To summary, we have use SystemC modeling for the architecture exploration and investigating the performance issue on 3D graphics hardware acceleration. The experimental results reveal the performance gap which cannot obviously notice using the software profiling. In the future, the performance tuning on system architecture, hardware pipelining, frame buffer clear, hardware implementation of tile divider are all open issues on our 3D graphics acceleration research.

REFERENCES