World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:5, No:6, 2011

A Virtual Simulation Environment for a Design & Verification of a GPGPU

Kwang Y. Lee, Tae R. Park, Jae C. Kwak, and Yong S. Koo

***Abstract*—**When a small H/W IP is designed, we can develop an appropriate verification environment by observing the simulated signal waves, or using the serial test vectors for the fixed output. In the case of design and verification of a massive parallel processor with multiple IPs, it's difficult to make a verification system with existing common verification environment, and to verify each partial IP. A TestDrive verification environment can build easy and reliable verification system that can produce highly intuitive results by applying Modelsim and SystemVerilog's DPI. It shows many advantages, for example a high-level design of a GPGPU processor design can be migrate to FPGA board immediately.

***Keywords*—**Virtual Simulation, Verification, IP Design, GPGPU

I. Introduction

ESTABLISHING a verification environment provides a debugging mechanism for correct results from a processor and also a foundation for sharing a developing project to participated developing engineers.

To develop a processor, the design of processor IP must be accompanied with developing of a compiler and its applications. Proper commands should be defined for the compiler. These commands depend on the implementation of a processor. The application can be developed based on the design of a compiler and a processor. These applications are necessary to verify the design of a processor.

A parallel development is needed for the efficiency. In order to develop a system in parallel, independent development platform must be constructed, so that each component can be developed without consideration of a whole system. In order words, a processor, a complier, and its applications should be developed at the same time. For the parallel development, unrelated independent jobs needs to be designated as a pre-proceed step. The design of a processor does not start from the design of HDL. The design of a compiler and its applications do not consider the design of the processor at the beginning stage.

Fig. 1 shows the progress steps of IP development for the design of a processor as a parallel step. For the completion of a processor, the steps are divided into a preparation of parallel steps and an independent developing platform.

Kwang Yeob Lee is with the Computer Engineering Department, Seokyeong University, Seoul, S. Korea (e-mail:kylee@ skuniv.ac.kr).
Tae Ryoung Park is with the Computer Engineering Department, Seokyeong University, Seoul, S. Korea (e-mail:trpark@ skuniv.ac.kr).
Jae Chang Kwak is with the Computer Science Department, Seokyeong University, Seoul, S. Korea (e-mail:jckwak@ skuniv.ac.kr).
Yong Seo Koo is with the Electronics Engineering Department, Dankook University, Gyeonggi-do, S. Korea (e-mail:yskoo@ dankook.ac.kr).

Upper part developers design overall structure of a processor, and construct the system using C language. The constructed C modules are distributed to lower part developers. Lower part developers implement HDL with the assigned C modules and repeat verifications and designs with the system through DPI(Direct Programming Interface) of System Verilog[1-2]. They complete a processor by collecting the sources of constructed HDLs.

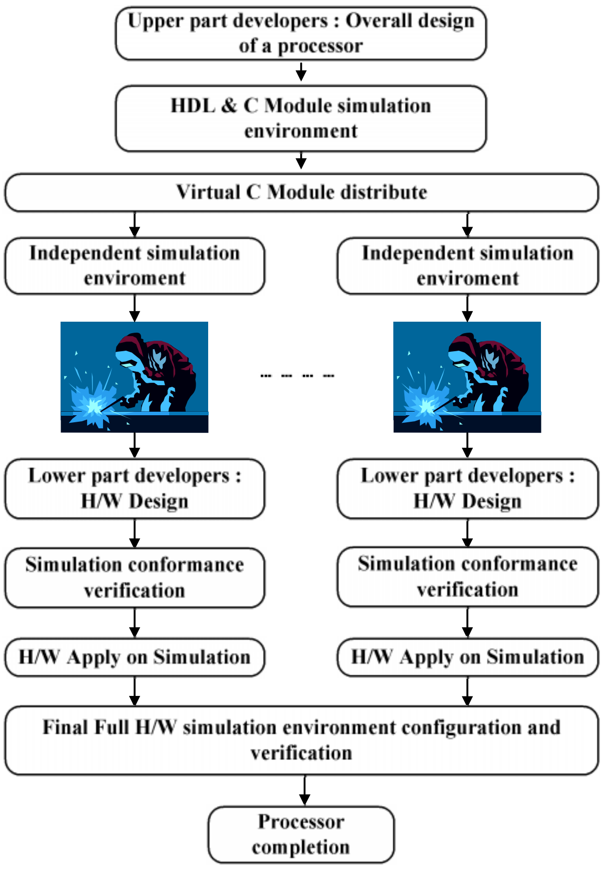
Upper part developers : Overall design of a processor
HDL & C Module simulation environment
Virtual C Module distribute
Independent simulation enviroment
Independent simulation enviroment
Lower part developers : H/W Design
Lower part developers : H/W Design
Simulation conformance verification
Simulation conformance verification
H/W Apply on Simulation
H/W Apply on Simulation
Final Full H/W simulation environment configuration and verification
Processor completion

Fig. 1 Verification steps for the efficient design

This mechanism takes a time to construct an initial environment, but a rapid and stable development is possible by the parallel process. We implement a Test Drive System[3] to construct these parallel developments of GPGPU(General Purpose computing on Graphics Processing Unit)[4].

II. C Implementation thru Interpreting Assembler

Executable example programs are needed for verification of a processor. For these example programs, a simplified assembler is required to implement machine instructions for function commands. Since the processor is at the developing stage, instruction fields are not yet set up. Whenever the definitions of instructions are modified, the assembler should be changed. At the worst case, the assembler is needed to be rewritten completely.

In practice, detailed parts of the design of GPGPU are kept changed. Development of general assembler should be postponed until the completion of processor architecture. But a certain type of simplified assembler is still needed for the verification of the processor. This assembler must be independent to the design of GPGPU.

As a solution, we implement the assembler as an interpreter. Syntax of assembler and Conversion rules to machine language are separated into different files, so that the development of assembler and design of processor can be preceded independently.

```
// equation
#function GP_SET_ERR
#function GP_SET_EQUAL        @ "="  # : @GP_SET_ADD #= @F GP_SET_EQL_ADD
#function GP_SET_ADD          @ "+=" # : @GP_SET_SUB #+
#function GP_SET_SUB          @ "-=" # : @GP_SET_MUL #-
#function GP_SET_MUL          @ "*=" # : @GP_SET_DIV #*
#function GP_SET_DIV          @ "/=" # : @GP_SET_AND #/
#function GP_SET_AND          @ "&=" # : @GP_SET_OR  #&
#function GP_SET_OR           @ "|=" # : @GP_SET_XOR #|
#function GP_SET_XOR          @ "^=" # : @GP_SET_NOT #^
#function GP_SET_NOT          @ "~=" # : @GP_SET_ERR #~
#function put                 #V @F GP_SET_EQUAL ;
```

With this interpreting assembler, a simple program can be written by writing a script of the assembler. But it is difficult to use branch and repeat instructions. C implementation is considered to write programs easily. A general compiler is implemented into two programs, object generation program that translates C codes into intermediate codes and linker program that converts the intermediate codes into binary codes. Since we use an interpreting method, C syntax is implemented by the interpreting assembler. The interpreting assembler converts C codes into syntax of the assembler.

```
// Function declaration
#function _FUNC_SET_CHECK       @Y _TEMP @F# @? 0 * \

#function void                  @F _FUNC_SET_CHECK @Y _FUNC_NAME #= @T _FUNC_NAME ( \
                                @ "void" # ) @L { @R{
#function {                     @R{
#function _FUNC_MAIN_END        @ "_FE goto.end;"
#function _FUNC_END             @Y _FUNC_NAME @? main @FUNC_MAIN_END \
                                @ "_FE addr(d[127].xyz,r[127],addr.x),mov(xyz,r[127].yzw)/
                                Thread(JMP.dir);"
#function _FE                   @R}
#function _FUNC_LOOP_END        @ "continue;_FE "
#function _FUNC_ELSE            @L @ "if" @FUNC_ELSE_IF @L { @Y PC @+ 1 @A @R} \
                                @R{ @Y PC @ 1 @A @ "goto };"
#function _ELIF                 @F _if_BODY
#function _FUNC_ELSE_IF         @Y PC @+ 1 @A @R} @R{ @Y PC @ 1 @A @ "goto };_ELIF"
#function }                     @Y _TEMP @F# @? 1 @FUNC_END @Y _TEMP @F% \
                                @? 1 @FUNC_LOOP_END @L @ "else" @FUNC_ELSE @R}
```

III. INTEGRATED VERIFICATION SYSTEM FOR TEST DRIVE

For rapid and stable processor development, designs should be proceeded in parallel and independently. A verification system is needed to provide a development platform and communication channels to developers.

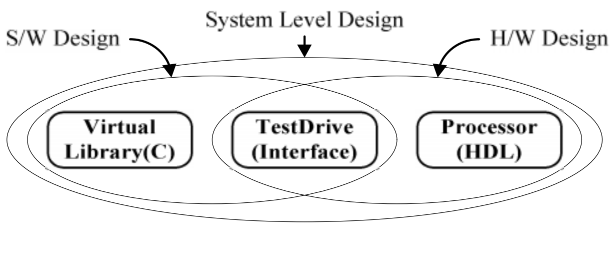

Fig.2 The role of test drive system

Fig.2 shows the structure of system level design to verify a group of software and hardware development together. In order to make an interface between two different languages, C and Verilog HDL, DPI(Direct Programming Interface) of SystemVerilog is used. DPI implements C functions into DLL (Dynamic Library Link) file. DPI provides the method for HDL to call these functions.But there are incompatibilities at the hardware composition in the syntax of SystemVerilog(*.sv) and Verilog(*.v). We introduce a source wrapping method to debug HDL.

```
// " System v"
...
`ifdef      _DEBUG_TESTDRIVE_
            DEBUG_Processor        processor_inst(A);
`else
            MTSP_Processor         processor_inst(A);
`endif
...
module DEBUG_Processor(A)
...
import "DPI-C" function int Debug_Print(***);
...
            MTSP_Processor         processor_inst(A);
            Debug_Print(processor_inst.***);
...
endmodule
module MTSP_Processor(A)
...
endmodule
```

As shown at the above example, we implement SystemVerilog source for debugging between the calling part and the implementation part, so that data extraction is possible for the debugging. In the hardware composition, the composition into FPGA is possible by removing debugging codes through the elimination of the definition of ‘_DEBUG_TESTDRIVE_’.

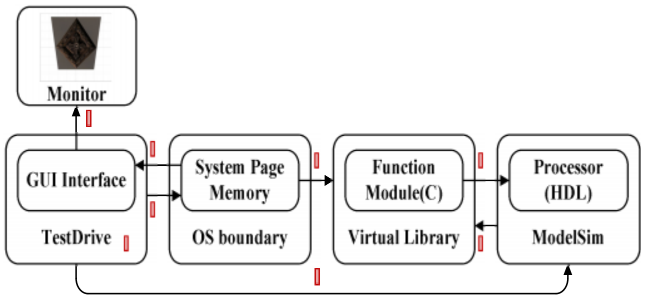

Fig. 3 Verification steps using test drive

The verification of test drive includes 8 steps as shown in Fig. 3.

Each step is as follows:

. A project is created. The size of system memory and the structure of output display are set at this step.
. When test drive opens the project, the system memory is generated as specified at the previous step. This memory structure has the same size of FPGA. It has an independent address space from the host PC.
. At the simulation, an application is called and the verification of emulation, simulation, and FPGA is executed selectively on the system driver as setting at test drive. The HDL module, which is implemented by multi-core with bus, is derived by Simulator(ModelSim).
. The called HDL module calls Virtual Library DLL to substitute unimplemented part by debugging related functions or HDL into the function.
. Virtual Library shares the system memory generated at the test drive.
. The results of HDL are stored at the shared system memory.
. The stored result can read by the test drive.
. Simulation results can be checked through the display.

The above verification steps provide an environment to drive every aspect of development processes. All developers can exchange their feedbacks through this verification system.

Since the processes from multi-core to bus interface can be verified at the simulation, FPGA is applied directly without any further verifications, as shown is Fig. 4.

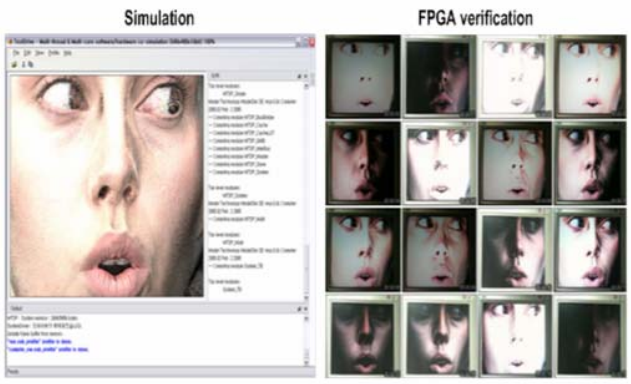

Fig. 4 Simulation using test drive and verification of FPGA

IV. Verification of GPGPU Multicore Processor

A. Verification of multi-core simulation

The verification system of test drive is used to verify multi-core. A number of contents for the proposed GPGPU are implemented by the modified C language. The performance and efficiency are measured by the simulation. Simulation environment includes a 64-bits bus with DIMM memory structure and a 200MHz clock speed[5].

Verification of multi-core efficiency

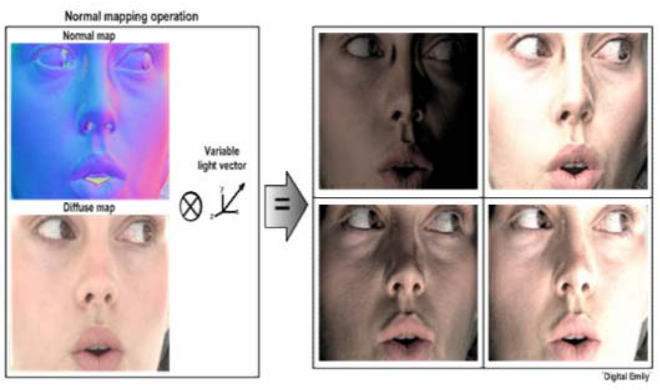

Fig. 5 Normal mapping computation

Fig. 5 shows the implementation of normal mapping from 'Digital Emily'source in the movie'Matrix'.

The above simulation uses dual-phases and memory pre-fetching for the performance optimization[6].

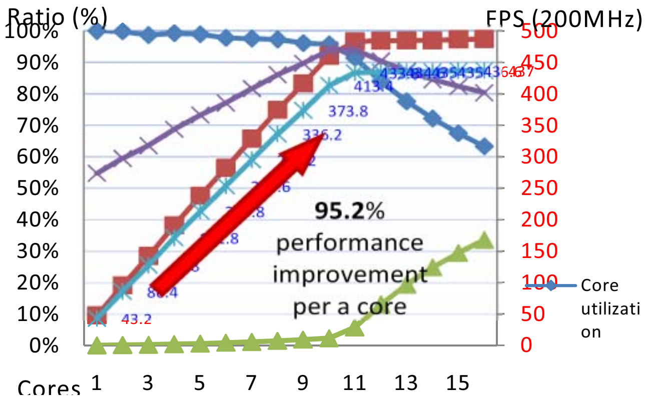

Fig. 6 Performance of normal mapping(all functions is included)

Fig. 6 shows the performance of the normal mapping program based on the number of core. Since the bandwidth of the system bus is 64-bits, the performance doesn't increase linearly by the number of core. But each core shows 95.2% performance improvement, which indicates excellent core efficiency. All processors show 95% efficiency until the limit of bus usage is reached. The utilization of the processor and bus can be improved over 90%.

Performance verification of memory pre-fetch and dual-phase

Fig. 7 shows the overall efficiency is lowered below 90% without using memory pre-fetch. At the below 10 cores configuration, the performance reduction per core is about 1%, which twice bigger than with memory pre-fetch. This reduction is caused by the wait time for the memory access. This program shows 16% improvement of overall performance.

Memory pre-fetch is a technique to implement a Memory Latency Hiding. In the multi-thread environment[7], when a thread requests to access the memory, other threads also can request the memory access at the same time. Therefore one memory operation could take a very long clock cycles, even thousands of them. It is called an idle time by a memory operation. There are two methods to utilize this idle time. One of them is a context switching to activate another thread for the performance improvement as a hardware solution. This solution is necessary for current GP-GPUs to improve the performance.

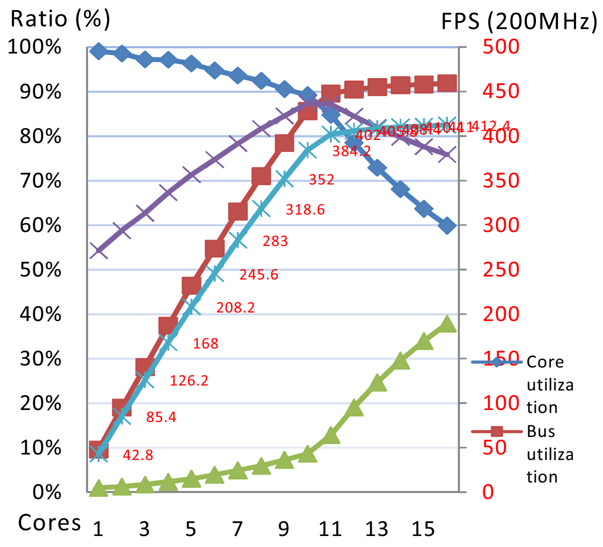

Fig. 7 Performance of normal mapping (without memory pre-fetch)

B. *Verification of multi-core execution*

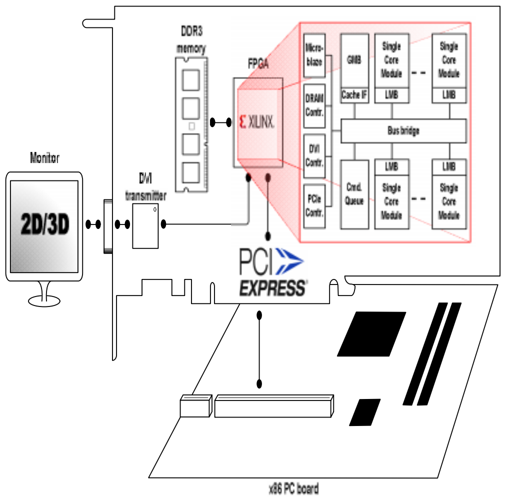

Fig. 8 the configuration of FPGA verification environment

As shown in Fig. 8, Virtex-6 board ML605 of Xilinx is used to verify real executions. The proposed GPGPU has seven cores. Input data from PCI Express Bus at Window environment can be checked at the monitor, connected to DVI port. The results are compared to the results of simulation of test drive, in order to verify real executions.

TABLE I
MULTI-CORE REGISTERS FOR FPGA VERIFICATION

Module	Count
Core	7
Instruction registers	4096 words
GPRs(Generanl Purpose Registers)	128 vectors
LMB(Local Memory Block)	2048 double vectors
GMB(Global Memory Block)	4096 double vectors
LUT	1024 words
SCs (Scratch Counters)	2 vectors
GSC (Global Scratch Counter)	1 vector

Since the optimal number of registers has not been decided, the size of registers is designed at the maximum number at this moment, as shown in Table 2. When the optimal number of registers is decided through implementation and verification of many applications , the size of register will be reduced. Implemented examples for this report use 42 of GPRs, 1060 of LMB, 1 of SCs, and 1 of GSC. GMB and LUT are not used.

•

V. CONCLUSION

The proposed multi-core GPGPU is designed for both single tread and multi-thread operations. It has a flexibility of general CPU and a high performance of GP-GPU. It shows the similar performance with the existing GPU.

The existing GPGPU executes a large number of SIMD array as a single instruction. It is called SIMT(Single Instruction Multiple Threads) structure in nVidia. With this structure, when hundreds of processors are integrated on a chip, the space for the instruction register is reduced. Therefore more processors can be integrated on a chip, so that it gains more performance.

For the verification, we built our own TestDrive system and it can do software emulation, hardware simulation and FPGA real verification by one single program on the same platform. The FPGA verification system had shown a normal operation with a GPGPU that is composed of 7 cores at 100MHz operation.

REFERENCES

[1] http://www.systemverilog.org
[2] http://www.doulos.com/knowhow/sysverilog/tutorial/dpi
[3] Hyungki Jeong, Kwang Yeob Lee and Jae Chang Kwak,"Test-Drive System for a Design & Verification of a GP-GPU Processor," 2010 SoC Conference, The institute of Electronics Engineering of Korea, April 2010, pp.40-43
[4] GPGPU, General Purpose Computation Using Graphics Hardware, http://www.gpgpu.org

[5] Hyungki Jeong, Kwang Yeob Lee and Jae Chang Kwak,"A Multi-thread Processor Architecture with Dual Phase Variable-Length Instructions," ITC-CSCC2008, July 2008, pp.209-212.

[6] Kwang Yeob Lee, Tae Ryoung Park, Jae Chang Kwak, Yong Seo Koo,"A Design of Multi-threaded Shader Processor with Dual-Phase Pipeline Architecture," The First international Conference on Advances in Multimedia MMEDIA 2009, 20-25 July 2009 colmar, France, pp 121-124.

[7] S. Ryoo, C. I. Rodrigues, S.S. Stone, S.S. Baghsorkhi, S.Z. Ueng, J.A. Stratton, and W.W. Hwu,"Program optimization space pruning for a multithreaded GPU," in Proceedings of the 2008 International Symposium on Code Generation and Optimization, April 2008, pp. 195-204.