An Implementation Method of Cycle Accurate Simulator for the Design of a Pipelined DSP

Hyeong Bae Park, Ju Sung Park, Tae Hoon Kim, and Hua Jun Chi

Abstract—**In this paper, we introduce an implementtation method of the CBS (Cycle Base Simulator), which describes the operation of a DSP (Digital Signal Processor) at a pipeline cycle level. The CBS is coded with C++, and is verified by comparing the results from the CBS and HDL simulation of the DSP with the various test vectors and application programs. The CBS shows the data about the internal registers, status flags, data bus, address bus, input and output pin of the DSP, and also the control signals at each pipeline cycle. The developed CBS can be used in evaluating the performance of the target DSP before the RTL(Register Transfer Level) coding as well as a reference for the RTL level design.**

Index Terms—**Cycle accurate simulator, co-simulation, pipelined DSP**

I. INTRODUCTION

Today's remarkable development of the semiconductor fabrication technology and CAD (Computer Aided Design) tools makes it possible to design million gates chip in a short time. The chip complexity becomes larger, but the design period becomes shorter. Under this situation, it is very important to make the model for evaluation of the performance of the target chip before the main design start. And also, it would be good we could have a reference for quick debugging at RTL coding step, which has the detailed information about internal data and control signals of the target chip. There are many abstraction levels in modeling microprocessors and DSP's depending on the usage of the model, such as instruction accurate simulation model, event driven simulation model, cycle accurate simulation model[1~3]. There is a trade-off between simulation speed and accuracy of the models.

The various models may be helpful in designing a microprocessor or a DSP, but we have to invest a lot of time and manpower for developing the models themselves. Thus it is very important to choose the proper model for designing the DSP efficiently. In this respect, it would be good the model have a good simulation speed, lots of information about the internal registers, status flags, data bus, address bus, input and output pin of the DSP, and also the control signals at each pipeline cycle. The model that has such properties is called as the pipeline cycle accurate simulator.

In this paper, we represent an implementation method of a cycle accurate simulator using C++ language for 24bit DSP. The previous papers didn't show the detailed techniques and procedures for implementing the CBS and the control signal information, which is very useful for designing DSP at RTL level[4~7]. This paper introduces the detailed implementation method and procedure, which are easily adopted to developing another CBS's for other processors. Another the important goal of CBS is being used as a reference in RTL design verification step, at this step the information about control signals are useful. This paper shows how to build co-simulation environment to easily compare the result from CBS and that of HDL (Hardware Description Language) simulation[4~6, 8].

In section 2, we describe design flow of CBS,

Manuscript received Sep 9, 2006; revised Nov. 14, 2006.
Dept. of Electronics Engineering Pusan National Univ., Busan, Korea
E-mail : juspark@pusan.ac.kr

followed in section 3, the procedure and technique of designing cycle accurate model for 24bit DSP CBS is introduced. In section 4, the integration of CBS model and RTL model on the same platform will be introduced. In the rest of section, we give results and conclusions.

II. Design Flow

The design procedure of CBS and what have to be done at each step are shown in Fig. 1 The clear and explicit analysis about architecture, instructions, and pipeline of the target DSP is carried out before the detailed code coding begins.

Pipeline consisting of 6 levels is implemented by combination of one or more functional blocks. At the pipeline implementation step, the thorough understanding about what functional block is involved by each pipeline stage and how each pipeline stage communicates with the functional blocks for implementing all instructions is necessary. We makes the I/O ports of the functional block of CBS exactly matched to those of the RTL level code for the purpose of using CBS as a reference at RTL code debugging step. Then operation of the functional blocks is described as behavioral level. After the functional blocks are designed, 6 pipeline stages are designed according to the objects that should be done at each stage. The decoder block that generates control signals depending on the instruction is designed at the step of implementing instruction and verification, and verified. The designed CBS is verified through the various test vectors and application programs. The integration of CBS and HDL simulation tool and the GUI design have been carried out for convenient use at the step.

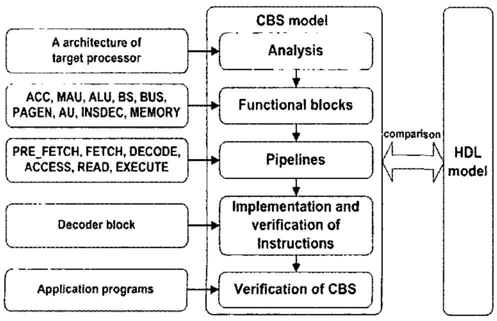

CBS model
A architecture of target processor
Analysis
ACC, MAU, ALU, BS, BUS, PAGEN, AU, INSDEC, MEMORY
Functional blocks
PRE_FETCH, FETCH, DECODE, ACCESS, READ, EXECUTE
Pipelines
comparison
HDL model
Decoder block
Implementation and verification of Instructions
Application programs
Verification of CBS

Fig. 1. Design flow of CBS.

III. Design of Cycle Accurate Model

In this section, we introduce the details of implementing the cycle accurate simulation model. Since target DSP has 6 pipeline stages, the CBS is divided to 6 operations such as: PRE_FETCH, FETCH, DECODE, ACCESS, READ, EXECUTION. The functional blocks that are designed at the previous step are placed the proper pipeline stages according to their function as shown in Fig. 2 The architecture of the CBS is shown in Fig. 2.

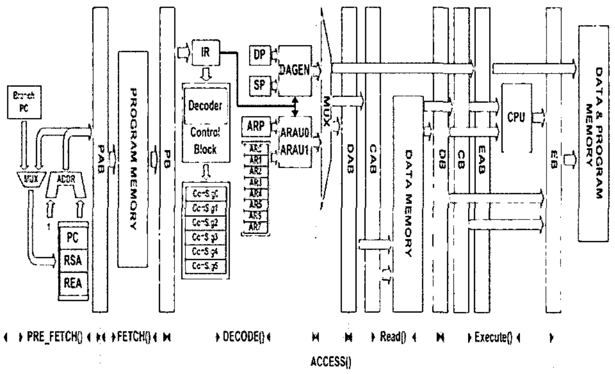


Fig. 2. The architecture of CBS.

1. Functional Block

In the case of designing processor using HDL, each functional block is independent module, and each module has connected through wire. Fig. 3 indicates the method for designing functional blocks. Each functional block is defined by CLASS type variable to preserve modularity. Internal variables consist of 3 types which indicate inputs, outputs, and status values. Internal variables can be accessed only through GET() and SET() member functions. Variables are defined as PROTECT type, and functions are defined as PUBLIC type. Functional blocks are composed of 3 internal functions that include ControlSignal() function for control signal, Execution() function for its operation, and Status() for handling operational results.

It is shown in Fig. 4 that how the functional blocks are executed in accordance with pipeline stage, through an example, MAU unit. The necessary control signals for operation of MAU is obtained by calling MAU. SetConSig( ) function and MAU operation is carried

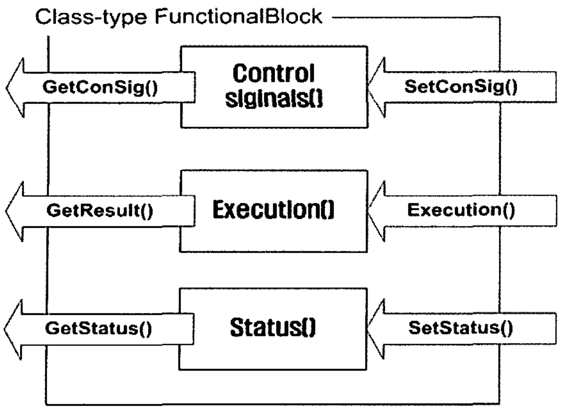


Fig. 3. Concept of implementing the functional blocks.

```
MAU.SetConSig(  CONSIG[CurPos].E.pMau.AddSub,
                CONSIG[CurPos].E.pMau.Mux,
                CONSIG[CurPos].E.pMau.PlusMinus,
                CONSIG[CurPos].E.pMau.SelSrc,
                CONSIG[CurPos].E.pMau.XMSignCon,
                CONSIG[CurPos].E.pMau.XMux,
                CONSIG[CurPos].E.pMau.YMSignCon,
                CONSIG[CurPos].E.pMau.YMux);

MAU.ExeMAU(     TREG.GetReg(),
                BUS.GetDB(),
                BUS.GetPB(),
                BUS.GetCB(),
                ACC.GetACCA(),
                ACC.GetACCB(),
                ST1FRCT);
```

Fig. 4. Concept of the operation of a functional block.

out by calling MAU.ExeMAU( ) function. Finally, MAU.GetResult() function is executed to access the internal status values.

2. Pipeline

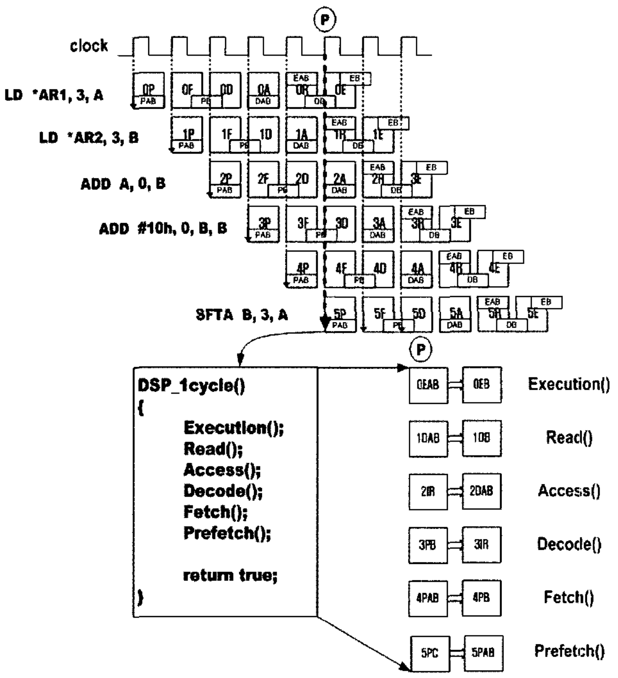
Real hardware operates in parallel, but the simulation programs coded with C or C++ are executed serially on computers. The operation of pipeline is carried out in reverse order of data flow of the hardware on the CBS program, properly modeling the operation of the target DSP[7, 9]. The pipeline functions, which carry out 6 pipeline operations, are implemented with CLASS variable type. Entire pipeline operation is taken into account when one of 6 pipeline stage is designed. Source code for implementing EXECUTE pipeline is shown as an example in Fig. 5. EXECUTE stage operations consist of CPU operation and storing results in memory. Each functional block is called in reverse order as shown in Fig. 5.

```
BOOL CDSP::ExeExecute()
{
    // CPU operation
    BS.SetConSig();
    BS.ExeBS();
    ALU.SetConSig();
    ALU.ExeALU();
    MAU.SetConSig();
    MAU.ExeMAU();
    ACC.SetConSig();
    ACC.ExeACC();

    // Store result and update-status register
    BUS.ExeEB();
    DATAMEM.ExeWriteMEM();
    StatusRegisterUpdate();
    return true;
}
```

Fig. 5. An example of implementing pipeline stage, EXECUTE.

In the case of the H/W, 1 cycle execution means that the 6 pipelines operation is carried out concurrently. But CBS must call 6 pipeline functions for 1 cycle execution in the reverse order: EXECUTE() → READ() → ACCESS() → DECODE() → FETCH() → PRE-FETCH() as shown in Fig. 6 for operating like the H/W[3, 6]. Each pipeline function is called at every 1 cycle. All of the control signals is generated in decoder blocks at DECODE stage. The hazard problems are solved by generating control signals if next instruction is need to be stalled or in the case of hazard.



```
DSP_1cycle()
{
    Execution();
    Read();
    Access();
    Decode();
    Fetch();
    Prefetch();

    return true;
}
```

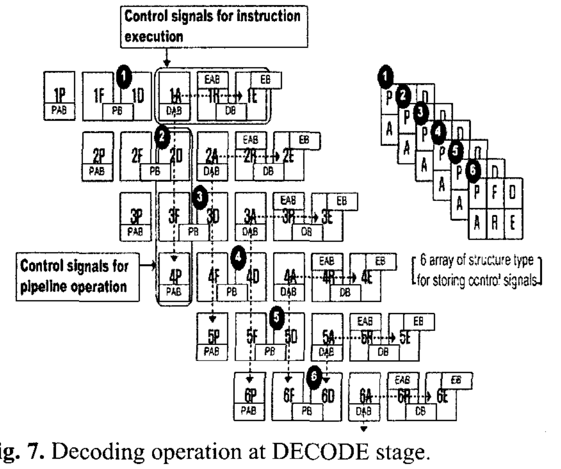
Fig. 6. Pipeline implementation sequence of CBS.

3. Decoder

Before the decoder block is designed, it is necessary to define what control signals are required for implementing all instructions at all functional blocks. The decoder block is designed to generate the control signals to be supplied to functional blocks for implementing the instructions of DSP. The instructions of target DSP have various operation modes, repeat mode, and hazard mode. The instructions are grouped into 9 kinds of TYPE depending on whether they are repeatable or not, and instruction cycles. The instructions have 15 STATE's that determine the pipeline operation of next instructions. The TYPE and STATE of instruction is determined at DECODE stage. The decoder block generates control signals according to TYPE and STATE.

The decoder block reads the instructions from IR(Instruction Register) and translates the instruction and then generate control signals. For storing control signals generated from decoder block, STRUCTURE type variable that has 6 arrays is prepared for 6 pipeline stages. Thus 6 instructions have independent control signals for parallel operation. Control signals are stored for next pipeline operation. Each pipeline is executed using stored control signals for appropriate operation of functional blocks. All of the control signals are generated and stored in arrays at DECODE stage.

In Fig. 7, ❶❷❸❹❺❻ indicates decoding timing in CBS. While the control signals for ACCESS, READ, EXECUTE of number 1 instruction are generated, and DECODE, FETCH, PRE_FETCH steps are carried out for instruction 2, 3, 4, respectively. If an instruction needs exceptional pipeline operation like stall or hazard, the decoder block generates exception control signals by using STATE-MACHINE that combines DECODE, FETCH, PRE_FETCH of the next three instructions.


Control signals for instruction execution
Control signals for pipeline operation
6 array of structure type for storing control signals
PAB
PB
DAB
DB
EAB
EB
Fig. 7. Decoding operation at DECODE stage.

4. Operation of Cycle Accurate Model

The operation of CBS for an example program CBS that includes repeat mode is shown in Fig. 8. For exceptional pipeline control, appropriate pipeline control signals are generated in DECODE stage. The repeat mode of "ADD *AR1+, A" instruction means that ADD operation is executed repeatedly for the value of RC(Repeat Counter) register without increasing PC(Program Counter). RC register is set to the value of repeat times at the first, then decreases as ADD operation has been finished. A control signal is generated to make PC stop increasing during the repeat period. The state of pipeline returns to the normal mode after the repeat operation has been done, then CBS carries out the next "ADD A, 0, B" instruction.

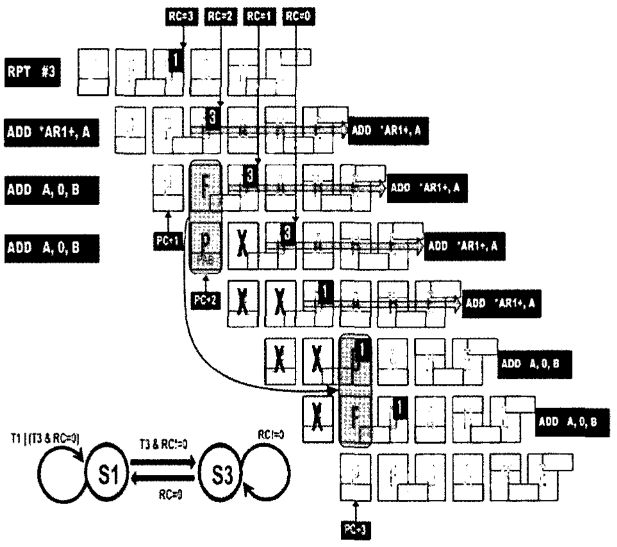


Fig. 8. Pipeline sequences of repeat mode.

IV. Set-up Co-Simulation Environment

We had two simulation models, CBS and RTL, for the same DSP, but they are different from each other in the points of the language used and the abstraction level for

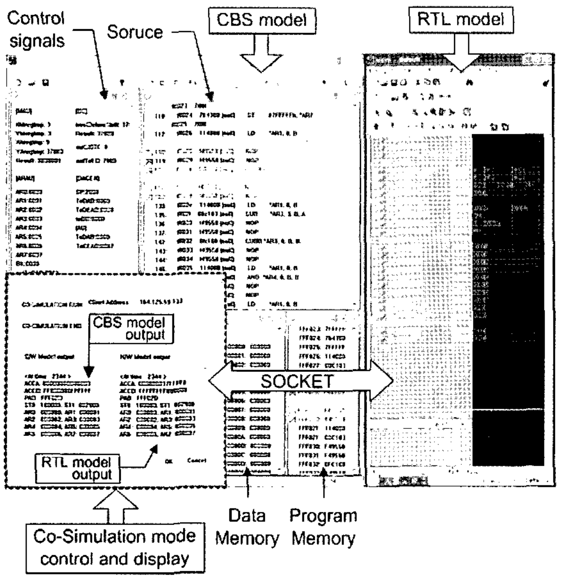


Fig. 9. Co-simulation of RTL model and CBS model.

logic operation, and the platform on which the program is running. It is necessary to be seen the simulation results from CBS and RTL level simulator on the same screen for efficient RTL code debugging. We make the CBS synchronized to RTL level simulator at the rate of pipeline cycle because RTL level simulation is slower than CBS. Two simulation is executed the same test vector at the same time, then the results, which consist of all data from storage elements and buses in DSP and control signals, are compared at pipeline cycle level. If there is a mismatch between two simulations, the simulation is hold and error message is generated. In co-simulation, socket protocol is used for communication between two models. To get internal values of RTL model and send the values through socket, PLI(Program Language Interface) is used[10, 11]. The operation of co-simulation is shown in Fig. 9.

V. Debugging and Results

CBS was verified with the various test vectors that consist of individual instruction set, combination of instruction set, and application programs. We have confirmed that results from CBS and HDL are exactly the same through running the prepared test vectors. We can save the debugging time at the RTL level coding of target DSP by using the developed CBS as a reference.

The running times of CBS and HDL RTL simulation for three applications are compared in Table 1. The

Table 1. Simulation speed.

Application Program	RTL SIMULATION (cycle/seconds)	CBS SIMULATION (cycle /seconds)	RATIO CBS/RTL
FFT	1,408	59,165	42
ADPCM	1,664	41,601	25
AAC DECODER	664	23,934	36

ratios of two simulation running time are different from application to application, but CBS is faster than HDL RTL as much as 25 to 42 times. We could evaluate the performance of target DSP in short time before RTL level design get stated, because the CBS simulates the DSP operation very fast compared to HDL RTL simulation.

VI. Conclusions

In this paper, we introduce an implementation method of CBS for a 24 bit DSP. The CBS gives us the information at the pipeline cycle that consists of all data of storage elements and buses, and control signals of functional blocks in DSP. The developed CBS is faster as 25 to 42 times than RTL simulation. We could evaluate the performance of target DSP in short time by using CBS before RTL level design get stated. An also, we could save RTL level design and debugging time of DSP that takes most time of design work by using CBS as a reference model for RTL design. The concept and technique we used in developing CBS for a 24 bit DSP can be adapted and expanded to developing CBS of other DSP's and processors.

Acknowledgments

This work was supported for two years by Pusan National University Research Grant.

References

[1] P. Bose, T. M. Conte, T. M. Austin, "Challenges in processor modeling and validation," *Micro, IEEE*, Volume 19, Issue 3, pp. 9-14, May-June 1999.

[2] A. Nohl, G. Braun, O. Schliebusch, R. Leupers, H. Meyr, A. Hoffmann, "A Universal Technique for Fast and Flexible Instruction-Set Architecture

Simulation," *Design Automation Conference 39th*, pp. 22-27, June 2002.

[3] A. Hoffmann, T. Kogel, A. Nohl, G. Braun, O. Schliebusch, O. Wahlen, A. Wieferink, H. Meyr, "A Novel Methodology for the Design of Application-Specific Instruction-Set Processors (ASIPs) Using a Machine Description Language," *Computer-Aided Design of Integrated Circuits and Systems*, pp. 1338-1354, Nov. 2001.

[4] Guillermo Maturana, James L. Ball, Jeffery Gee, "Incas: A Cycle Accurate Model of UltraSPARC^{TM}," *Computer Design IEEE*, pp. 130-135, Oct. 1995.

[5] Lisa Guerra, Joachim Fitzner, Dipankar Talukdar "Cycle and Phase Accurate DSP Modeling and Integration for HW/SW Co-Verification," *Design Automation Conference IEEE*, pp. 364-369, June 1999.

[6] R. Voith, "The PowerPC^{TM} 603 C++ Verilog^{TM} Interface Model," *Digest of papers-Spring compCon 94*, IEEE Computer Society Press, pp. 337-340, March 1994.

[7] Moon Gyung Kim, Byung In Moon, Sang Jun An, "Implementation of a Cycle-based Simulator for the Design of a Processor Core", *AP-ASIC '99 IEEE*, pp. 108-111, Aug. 1999.

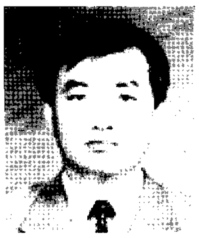
[8] Luc Séméria, Abhijit Ghosh, "Methodology for Hardware/Software Co-verification in C/C++," *Design Automation Conference*, pp. 405- 408, Jan. 2000.

[9] Mayan Moudgill, "Techniques for Implementing Fast Processor Simulators," *Simulation Symposium 31st Annual*, pp. 83-90, April 1998.

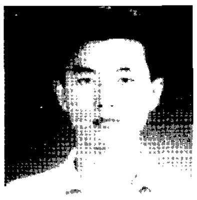
[10] Stuart Sutherland, "The Verilog PLI handbook : a user's guide and comprehensive reference on the Verilog programming language interface," Kluwer Academic Publishers, 1993.

[11] Mittra Swapnajit, "Principles of Verilog PLI," Kluwer Academic Publishers, 1999.

Hyeong Bae Park received the B.S degree in Telecommunication Engineering from Dongseo University and M.S. degree in electrical engineering from Pusan National University, Busan, Korea, in 2004, 2006, respectively. His research interests include High level modeling of processor, high-performance processor, and on-chip debug architecture.



Ju Sung Park was born in Junju, Korea, in 1953. He received the B.S degree in electronics engineering from Pusan National University, Pusan, Korea, in 1976, the M.S. degree in electrical engineering from KAIST, Seoul, Korea, in 1978, and the Ph.D. degree in electrical engineering from University of Florida, Gainsville, in 1989. From 1978 to 1991, he was with the ETRI, Taejun, Korea, where he work as a Principal Research Engineer and as the Manager and Director of the IC Design Group. While at ETRI, he designed several bipolar analog ICs and was in charge of developing VCR ICs, CMOS 8-bit microprocessors, and telecommunication chips. In 1991, he joined the Electronics Department, Pusan National University where he is now a Professor of Electrical Engineering. His current research interests are microprocessor and digital signal processing core design, digital audio algorithm implementation by hardware, and software co-design.



Tae Hoon Kim received the B.S., M.S., and Ph.D. degrees in Electronic Engineering from Pusan National Univ., Korea, in 1995, 1997, and 2002 respectively. He is currently in the course of Post-Doc. at the same University. From 2001 to 2006, he served as a manager in VOISO semiconductor corp.
His current research interests are DSP design, digital audio signal processing and coding, audio algorithm implementation by hardware and software.

Hua Jun Chi received the B.S degree in Computer Science Engineering from YUST(Yanbian University of Science & Technology), Yanji, China, in 2006. His research interests include audio algorithm (AAC, AC-3, MP3) coding and design of high performance core.