SOC Bus Transaction Verification Using AMBA Protocol Checker

Kab Joo Lee, Si Hyun Kim, and Hyo Seon Hwang

Abstract— **This paper presents an ARM-based SOC bus transaction verification IP and the usage experiences in SOC designs. The verification IP is an AMBA AHB protocol checker, which captures legal AHB transactions in FSM-style signal sequence checking routines. This checker can be considered as a reusable verification IP since it does not change unless the bus protocol changes. Our AHB protocol checker is designed to be scalable to any number of AHB masters and reusable for various AMBA-based SOC designs. The keys to the scalability and the reusability are Object-Oriented Programming (OOP), virtual port, and bind operation. This paper describes how OOP, virtual port, and bind features are used to implement AHB protocol checker. Using the AHB protocol checker, an AHB simulation monitor is constructed. The monitor checks the legal bus arbitration and detects the first cycle of an AHB transaction. Then it calls AHB protocol checker to check the expected AHB signal sequences. We integrate the AHB bus monitor into Verilog simulation environment to replace time-consuming visual waveform inspection, and it allows us to find design bugs quickly.**

This paper also discusses AMBA AHB bus transaction coverage metrics and AHB transaction coverage analysis. Test programs for five AHB masters of an SOC, four channel DMAs and a host interface unit are executed and transaction coverage for DMA verification is collected during simulation. These coverage results can be used to determine the weak point of test programs in terms of the number of bus transactions occurred and guide to improve the quality of the test programs. Also, the coverage results can be used to obtain bus utilization statistics since the bus cycles occupied by each AHB master can be obtained.

Index Terms — SOC Verification, AERA, Bus transaction coverage.

I. Introduction

Designing Silicon-On-a-Chip (SOC) typically involves in design and integration of one or more microprocessors, DSP units, on-chip bus architecture, memory system, and peripheral blocks. Typical industrial processors have their own bus architectures and bus transaction protocols. During SOC design phase, on-chip bus architecture should be fully understood and precisely implemented. Also, most module level blocks have bus interface wrapper, either as master and/or slave, to be attached to bus.

Verifying bus transactions of an SOC design may require a great deal of visual simulation waveform inspection, which can be time-consuming. Fig1 shows an example of AMBA [1] AHB bus transactions. It is 4-beat incrementing burst, transfer size of byte. In order to verify the correctness of AHB bus transactions, it is necessary to check AHB signals for all bus transactions in a cycle-based manner. The motivation for our work is to capture legal bus transaction behavior in a form of

Manuscript received January 2, 2002; revised June 12, 2002.

Kab Joo Lee is with system LSI Divison Samsung Electronics Co., Ltd. San #24 Nongseo-Ri, Kiheung-Eup, Yongin-City, Kyunggi-Do, South Korea

(E-mail: flyingv@samsung.co.kr)

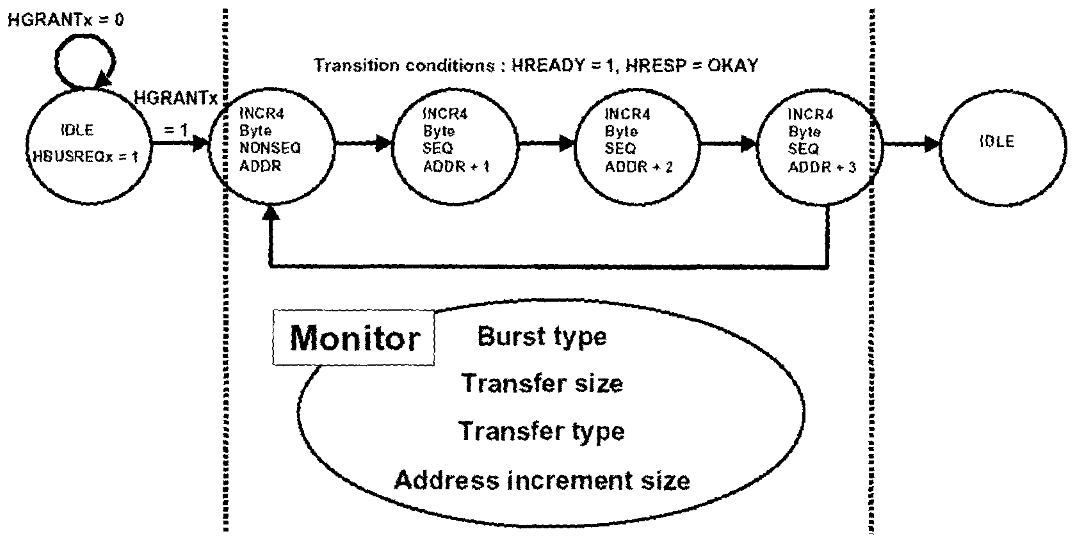


Fig. 1. 4-beat incrementing burst transfer diagram.

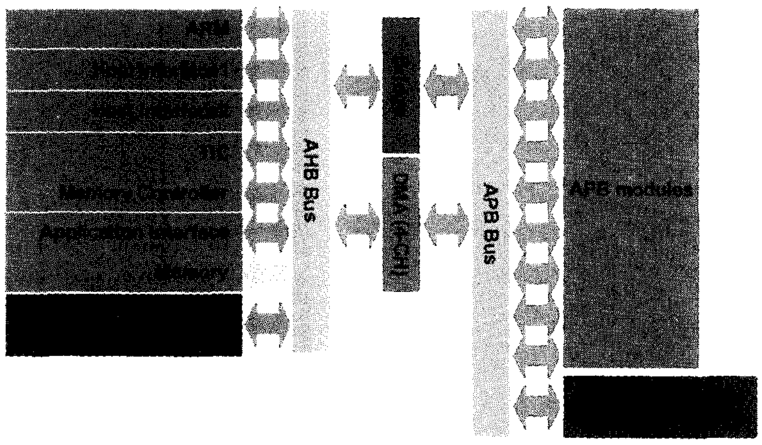


Fig. 2. DUV block diagram.

cycle-based checker so that we can greatly reduce the waveform inspection time.

In this paper, ARM-based SOC bus transaction verification using AMBA protocol checker is introduced. An AMBA AHB protocol checker is written using VERA Hardware Verification Language (HVL) [2] and integrated into Verilog simulation environment. This AHB protocol checker can be considered as a verification IP since it can be used for any AMBA-based design verification with little modifications.

This paper also introduces AMBA AHB transaction coverage metrics and analysis. The coverage metrics are also defined using VERA HVL. This coverage metric can be used to improve verification efficiency in two ways. First, it allows us to create more testbenches for less covered bus transactions. Secondly, it can provide bus utilization information of bus masters so that one can improve bus arbitration mechanism.

This paper is organized as follows. Section 2 describes Design Under Verification (DUV) which has two AMBA buses. Section 3 introduces AMBA AHB protocol checker and simulation monitor. Section 3 also defines bus transaction coverage metrics. Section 4 describes the methodology for AHB protocol checker integration into Verilog simulation environment. Section 5 discusses usage experiences, especially design bugs and transaction coverage analysis.

II. Design Under Verification

Design under verification (DUV) is shown in Fig2. DUV is an AMBA-based SOC for wireless network applications. It employs two AMBA buses, AHB and APB. A total of 8 AHB masters can request bus simultaneously and only one AHB master can start AHB transactions upon the reception of bus grant. AHB masters are ARM, Host Interface1, Host Interface2, TIC, and 4 DMA channels. AHB bus arbitration schemes are combination of fixed priority and rotated priority.

III. Amba AHB Protocol Checker-Verification IP

AHB protocol checker that we designed is basically a bus transaction checker. Legal AHB transactions are captured in FSM-style sequence checking routine. It is designed to be scalable to any number of AHB masters and reusable for various AMBA-based SOC designs. The keys to the scalability and the reusability are Object-Oriented Programming (OOP) [3] and virtual ports. This section describes how OOP and virtual port features are used to implement AHB protocol checker. This section also introduces an AHB simulation monitor that detects the start of AHB transaction and checks the legal AHB transaction in cycle-based fashion by calling the AHB protocol checker.

A. *AHB Checker Class and Simulation Monitor*

AHB protocol checker class is defined as shown in Fig. 3. The class includes local variables and methods. The variable type AHB_PORT_SIGNALS is a bind variable type, which will be used during checker object creation with a specific AHB master. The class method *new* is to create *AHB_checker* class object. When created, VERA accepts a bind variable input to create an *AHB_Checker* object specific to a particular AHB master. Using a bind variable during *AHB_Checker* creation allows us to create multiple number of *AHB_Checker* objects for multiple AHB masters. The class method *top* decides the bus transaction type based on HSIZE and HBURST and branches to the proper transaction verification task included in the class. Each task of four class tasks (only four transaction verification tasks are shown in Fig. 3) monitors AHB signals in cycle-based manner and checks if the AHB transactions are performed according to the AHB bus protocol. The details of these tasks are not shown in this paper. The AHB slave signals such as HREADY and HRESP are also monitored and used to check the valid AHB transactions.

```
class AHB_Checker
{
  bit                 [HBURST_WIDTH-1 : 0] burst;
  integer             i, addrIncr, addrBound,
                      transCount, burstLength;
  AHB_PORT_SIGNALS    master;

  task new            (AHB_PORT_SIGNALS busOwner);
  task top();
  task single_chk     (bit [HSIZE_WIDTH-1 : 0] size,
                       bit [HADDR_WIDTH-1 : 0] addrInit,
                       integer addrIncr );
  task incrX_chk      (bit [HSIZE_WIDTH-1 : 0] size,
                       bit [HADDR_WIDTH-1 : 0] addrInit,
                       integer addrIncr );
  task incrN_chk      (bit [HSIZE_WIDTH-1 : 0] size,
                       bit [HADDR_WIDTH-1 : 0] addrInit,
                       integer addrIncr );
  task wrapN_chk      (bit [HSIZE_WIDTH-1 : 0] size,
                       bit [HADDR_WIDTH-1 : 0] addrInit,
                       integer addrIncr );
}
```

Fig. 3. AHB protocol checker class.

This checker class and the tasks included in the class can be considered, as a verification IP since it is reusable for any AHB masters. The class object creation and the detection of the first cycle of AHB transaction are described in the VERA main test program. Note that in order to detect the first AHB cycle, one has to monitor and check the AHB arbitration. An AHB master can perform an AHB transaction only after it requests the bus to arbiter and gets bus grant signal from arbiter. Thus, using AHB protocol checker allows us to check bus request and grant logic mechanism.

Fig. 4. shows an example of AHB monitors (DMA monitor). As mentioned above, the DMA AHB simulation monitor first checks if DMA requests bus and gets granted properly. Then, based on the AHB slave condition (HREADY), it branches to the *top* task of AHB protocol checker, which detects first cycle of AHB transaction. One can construct an AHB monitor easily by modifying the example AHB monitor.

```
task
DMA0_AHB_monitor()
{
  integer  dma0ReadyWait = 0;
  string   dma0Name = "GDMA CHANNEL 0";

  while (1) {
    @ (posedge CLOCK);        coverage (OFF, DMA0_cov);
    @0, HBUSREQ_WAITv         DMA0.$HBUSREQ == 1'b1;
    @1, HGRANT_WAITv          DMA0.$HGRANT == 1'b1;

    if (DMA0.$HREADY == 1'b0) {
      @(posedge CLOCK);
       dma0ReadyWait++;
       if (dma0ReadyWait > HREADY_WAITv)
          error_report (dma0Name);
    } else {
      @0, HMASTER_WAITv       TOPv.hmaster_ahb == M_GDMA0v,
                              DMA0.$HTRANS == NONSEQv;
      dma0CovArg = ({DMA0.$HBURST, DMA0.$HSIZE});
      DMA0_AHB_Checker.top();
      coverage (ON, DMA0_cov);
      dma0ReadyWait = 0;
    }}}
```

Fig. 4. AHB simulation monitor example.

B. *Virtual Port and Binding*

Basically, AHB protocol checker is a class that embedded in a simulation monitor, which runs all the time during simulation. This checker needs to be designed scalable and reusable since there can be several AHB masters in an SOC design. Just copying the entire checker routines for each AHB master should not considered as a solution. This paper describes how virtual port and bind operation can be used for easy checker object creation and placement in target DUV. Conceptually, virtual port can be considered as signals that occur repeatedly in design. Good example of virtual port is AHB signals. All AHB modules need to interface the AMBA bus with AHB signals, which are common to all AHB modules. Thus, defining virtual port once, and binding each module's AHB signals with the corresponding physical port information can provide efficient object-oriented verification IP implementation.

Fig. 5. shows virtual port and bind operation for DMA channel 0 written in VERA HVL. In the Fig5, first, virtual ports are defined, which are AHB signals in AHB masters. Virtual ports defined in our AHB checker are HADDR, HTRANS, HSIZE, HBURST, HBUSREQ, HWRITE, HREADY, HRESP, and HGRANT. Then bind operation is performed. The virtual port variable, *AHB_VPORT_SIGNALS*, is used to define bind variables for AHB masters. A bind variable has physical port information (RTL signal) of a particular AHB master. A bind variable should be defined for each AHB master to be monitored during simulation.

```
port
AHB_VPORT_SIGNALS
{
   HADDR;
   HTRANS;
   HSIZE;
   HBURST;
   HBUSREQ;
   HWRITE;
   HREADY;
   HRESP;
   HGRANT;
}
```

```
bind
AHB_VPORT_SIGNALS DMA0_BIND
{
   HADDR      TOPv.haddr_dma0;
   HTRANS     TOPv.htrans_dma0;
   HSIZE      TOPv.hsize_dma0;
   HBURST     TOPv.hburst_dma0;
   HBUSREQ    TOPv.hbusreq_dma0;
   HWRITE     TOPv.hwrite_dma0;
   HREADY     TOPv.hready_dma0;
   HRESP      TOPv.hreap_dma0;
   HGRANT     TOPv.hgrant_dma0;
}
```

Fig. 5. AHB virtual ports and binding for DMA channel 0

C. *Coverage Definition*

AHB transaction coverage is defined based on two AHB signals, HBURST and HSIZE. Since HBURST is a 3-bit signal and HSIZE is also a 3-bit signal, we can define a total of 64 transaction coverage metrics. In our implementation, we define 24 transaction coverage metrics using the combinations of these two signals. Table1 shows the bus transaction metrics defined during verification. In this table, only 9 coverage metrics out of 24 are shown. In our actual coverage definitions, combinations of all types of HBURST and only 3 types of HSIZE (byte, halfword, word) are used.

These coverage metrics are captured in VERA coverage definition structure and a coverage object is created for each AHB master. Initially, all coverage counters are set to zeros. Whenever any of these coverage definitions occurs during simulation, the corresponding coverage counters are incremented.

IV. Amba AHB Protocol Checker Integration

The AHB checker and simulation monitor is written in VERA HVL. Thus, it is required to integrate VERA verification environment and Verilog environment. This

Table 1. AHB transaction coverage metrics examples.

Coverage name	HBURST/HSIZE	Coverage Description
SINGLE BYTE	000/000	Single transfer, transfer size of byte
SINGLE HWORD	000/001	Single transfer, transfer size of halfword
SINGLE WORD	000/010	Single transfer, transfer size of word
WRAP4 BYTE	010/000	4-beat wrapping burst, transfer size of byte
WRAP4 HWORD	010/001	4-beat wrapping burst, transfer size of halfword
WRAP4 WORD	010/010	4-beat wrapping burst, transfer size of word
INCR4 BYTE	011/000	4-beat incrementing burst, transfer size of byte
INCR4 HWORD	011/001	4-beat incrementing burst, transfer size of halfword
INCR4 WORD	011/010	4-beat incrementing burst, transfer size of halfword

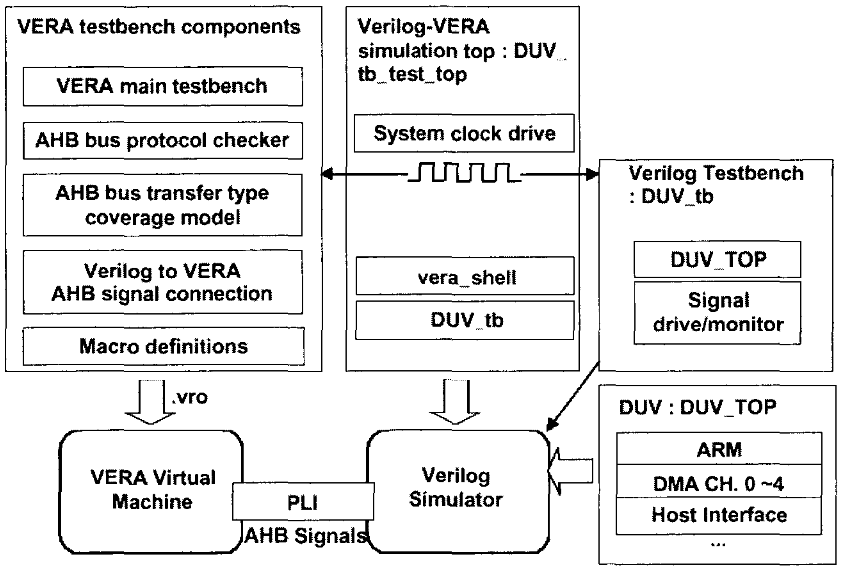


Fig. 6. Verilog-VERA simulation environment.

section describes AHB protocol checker/simulation monitor integration into Verilog simulation environment. The integration requires creation of Verilog-VERA simulation top file, definitions of VERA interface signals for monitoring AHB transactions, and VERA main program that creates AHB checker objects, AHB transaction coverage objects, and execution of VERA monitors with Verilog simulation.

A. *Simulation Environment*

Fig. 6. shows the Verilog-VERA simulation environment, which can be divided into two worlds. The Verilog environment includes DUT in Verilog HDL, Verilog-VERA simulation testbench which includes vera_shell for VERA interface, Verilog-VERA simulation top module, and Verilog simulator. The VERA environment includes VERA testbench components and VERA virtual

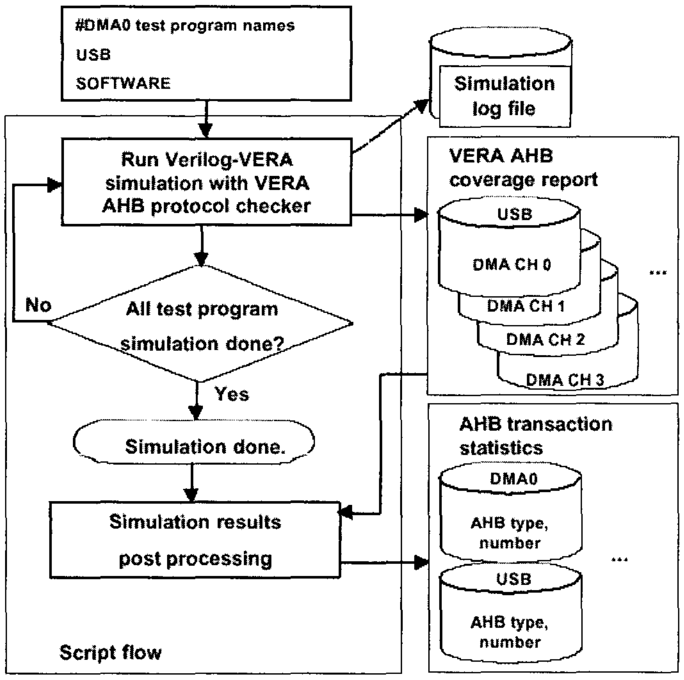


Fig. 7. Script and post-processing flow for AHB transaction coverage analysis.

machine. The VERA test bench components consist of VERA main program, AHB protocol checker, AHB coverage model, interface signal definition between Verilog and VERA (all AHB signals to be monitored are defined), and macro definitions. Verilog simulation and VERA checker are synchronized with the system clock generated in Verilog-VERA simulation top module. VERA and Verilog simulator communicates via Programming Language Interface (PLI).

VERA monitors, which are instantiated in VERA main program, are run during Verilog simulation and watch Verilog simulation termination condition. If simulation is terminated, VERA monitors are also terminated and VERA coverage reports are generated.

B. *Simulation and Post-processing Script Flow*

Fig. 7. shows simulation and post-processing flow for 4 channel DMA AHB transaction coverage analysis. The simulation script reads DMA test program name and runs the corresponding test program written in ARM assembly program. In other words, DMA operation is tested in chip level verification environment that includes ARM processor, AMBA bus controller, memory controller, and interrupt controller.

Each test program is run in Verilog-VERA simulation environment including AHB protocol checker and coverage metrics. After each program execution, VERA AHB transaction coverage reports are generated for each DMA channel, i.e. AHB monitors are attached to all four DMA channels since a test program may utilize several DMA channels. After execution of all DMA channel test programs, simulation results post processing is executed. This step reads AHB coverage reports and produces AHB transaction statistics, the total AHB transactions obtained for DMA channels. Scripts for post processing are written in perl langauge.

V. AHB Transaction Verification Results

This section describes design bugs that are found using the protocol checker, and analyzes AHB transaction coverage results obtained from four channel DMA verification.

A. *Design Bug*

The AHB monitor was attached to Host Interface1 which has a newly-designed AHB wrapper. For some cases, Host Interface1 did not give IDLE signal (HTRANS) at the end of AHB transactions, and this bug

Table. 2. DMA AHB master transaction coverage.

Test Program	DMA Channel			
	DMA0	DMA1	DMA2	DMA3
T1	0	0	0	0

T2	*S_8(60)*
	S_16(30)
	S_32(14)
		S_8(60)
		S_16(30)
		S_32(14)
			S_8(60)
			S_16(30)
			S_32(14)
				S_8(60)
				S_16(30)
				S_32(14)

T3	*S_8(15)*
	S_16(8)
	S_32(7)
		S_8(15)
		S_16(8)
		S_32(7)
			S_8(15)
			S_16(8)
			S_32(7)
				S_8(15)
				S_16(8)
				S_32(7)

T4	*S_32(1)*
	I4_8(15)
	I4_16(8)
	I4_32(7)
		S_32(1)
		I4_8(15)
		I4_16(8)
		I4_32(7)
			S_32(1)
			I4_8(15)
			I4_16(8)
			I4_32(7)
				S_32(1)
				I4_8(15)
				I4_16(8)
				I4_32(7)

T5	*S_8(4)*
	S_16(4)
	S_32(5)
	I4_8(7)
	I4_16(6)
	I4_32(4)
		S_8(4)
		S_16(4)
		S_32(5)
		I4_8(7)
		I4_16(6)
		I4_32(4)
			S_8(4)
			S_16(4)
			S_32(5)
			I4_8(7)
			I4_16(6)
			I4_32(4)
				S_8(4)
				S_16(4)
				S_32(5)
				I4_8(7)
				I4_16(6)
				I4_32(4)

T6	*S_8(8)*
	S_8(8)
		S_8(8)
		S_8(8)
			S_8(8)
			S_8(8)
				S_8(8)
				S_8(8)

T7	*S_32(12)*
	S_32(10)
		S_32(9)
		S_32(9)
			S_32(9)
			S_32(11)
				S_32(9)
				S_32(9)

T8	*S_32(2)*	S_32(2)	S_32(2)	S_32(2)
T9	*S_32(45)*	*S_32(32)*	*S_32(25)*	*S_32(24)*

was detected using the AHB protocol checker. Since protocol checker flags verification error at the time of protocol violation, one can easily use error information to find the cause of verification error.

B. *AHB Transaction Coverage Analysis*

A total of 9 test programs (T1-T9) were executed for four channel DMA master transaction verification as shown in Table 2. A total of 8 (T1-T8) test programs were used for DMA channels 1, 2, and 3, separately. The italic letters in the table represent the coverage reports obtained during DMA channel 0 verification. For instance, *S_8(60)* represents that single transfer, transfer size of byte, occurred 60 times. The letters with underlines represent the coverage reports obtained during DMA channel 1 verification. The letters in the boxes are associated with DMA channel 2 verification. The letters in the dark region are associated with DMA channel 3 verification.

Table 2 shows the results of coverage report during 4 channel DMA verification with AHB protocol checker. The types of AHB transactions supported by DMA are S_8 (single byte transfer), S_16 (single half-word transfer), S_32 (single word transfer), I4_8 (four beat increment byte transfer), I4_16 (four beat increment half-word transfer), and I4_32 (four beat increment word transfer). The numbers in the parenthesis represent the number of DMA master AHB transactions occurred during simulation. Thus, S_8(60) means that single byte transfer occurred sixty times during simulation.

The initial observation of this table indicates that DMA test programs covered all AHB transaction types supported by DMA. Since DMA operation is identical for each channel, the coverage numbers looks identical also. Note that test programs T6 and T7 are written to use two DMA channels and the coverage reports show two DMA channels are actually used. All of these observations allow us to verify the intents of the test programs, which is one of the important aspects of functional coverage analysis.

Note that test program T1 did not produce any AHB master bus transactions since T1 only reads and write DMA registers. It only requires DMA to be a AHB slave.

Transaction coverage can be also used to estimate bus utilization of test program since the number of bus transactions implies how much each master occupied bus.

VI. CONCLUSIONS

This paper presented SOC bus transaction verification and coverage analysis using VERA AHB protocol checker. The protocol checker can be considered as a verification IP so that it can be used for various AMBA-based SOC designs. Since it monitors AHB signals and detects illegal AHB transactions, it can replace time-consuming visual inspection for bus transaction sequence verification. Note that implementing this type of verification routines requires knowledge regarding specific bus protocol and verification system.

This paper also presented AHB transaction coverage analysis. four channel DMAs were verified in chip level verification environment and transaction coverages were collected during verification. These coverage results were used to determine the weak point of test program in terms of the number of transactions occurred and guide to improve the quality of the test programs. Also, the coverage results can be used to obtain bus usage statistics since the actual bus cycles used by each DMA channel can be obtained using the coverage numbers.

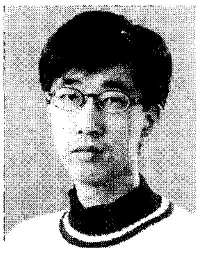
One disadvantage of using AHB protocol checker is imulation overhead caused by running Verilog simulation with AHB monitors. The simulation overhead actually depends on how many monitors are integrated in a design and how a test program is designed. However, we believe that the benefits using the checker may easily overwhelm the disadvantages. Our protocol checker did not implement SPLIT, RETRY protocol since the AHB master in our design did not support these types of bus transactions.

REFERENCES

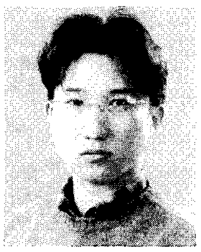
[1] AMBA Specification (Rev 2.0), ARM Ltd., 1999

[2] Vera Verification System User's Manual, Synopsys Inc., 1999

[3] Bruce Eckel, C++ Inside & Out, McGraw-Hill, California, 1993



Kab Joo Lee received B.S. in Electronic Engineering from Hanyang University, Ansan, Korea, 1989. He also received his Master of Engineering in Computer Engineering from Cornell University, Ithaca, NY, in 1992, and Ph.D. in Computer Engineering from Texas A&M University, College Station, TX, in 1997. Since June 1997, he has been a senior engineer at Samsung Electronics Co., where he is now working on development of wireless LAN ASICs and software. During 1997 – 2000, he has worked on development of ASIC verification methodologies including static timing analysis, formal equivalence checking, and RTL verification. During this period, his duties included ASIC design library development and design flow integration for Samsung ASIC products. His research interests include wireless LAN MAC protocols, architecture, and implementations, as well as SOC verification.



Si-Hyun Kim was born in Busan, Korea in 1971. He received B.S. and M.S. degrees in Electrical Engineering and Computer Science from the Ritsumeikan University, Kyoto, Japan, in 1996, and 1998, respectively. Since March 1998, he has been an engineer in system design technology lab, Samsung Electronics Co., Korea, where he is now working on system level verification IP development. He has been working in the areas of functional verification methodologies using formal verification, dynamic simulation using Vera, and functional verification IP development.

Hyoseon Hwang was born in 1974 in Pusan, Korea. She received B.S. degree in Computer Engineering from Pusan National University, Korea in 1998. In 1998, she joined the ASIC division of Samsung Electronics Co., Korea, and involved in design verification using formal equivalence checking. Since 2001, she has been working for development of wireless LAN hardware system. Her interests include system level design and verification for wireless systems.