

A Systematic Transaction-level Modeling and Verification

Junhyung Um, Woo-cheol Kwon, Hoon-Sang Jin, Kyu-Myung Choi, Jeong-Taek Kong, Soo-Kwan Eo
CAE Center, SoC R&E, System LSI Division
Samsung Electronics
Suwon, Korea
Junhyung.um@samsung.com

Taewhan Kim
School of Electrical Engineering and Computer Science
Seoul National University
Seoul, Korea
tkim@ssl.snu.ac.kr

Abstract— Design flow by transaction-level (TL) prototyping has received a great deal of attention as a solution of system-on-chip challenges that cannot be addressed by traditional design methodologies. This work is to address the issue of systematic TL modeling and verification methods. Specifically, (1) A systematic design modeling flow with a set of well-defined transaction protocols and TL model verification method are proposed; (2) We show comprehensive details on architecture (bus, memory, basic blocks) exploration results obtained by using our TL modeling and verification methodology for movie capture application, which interwinds Camera I/F, two DMAs, LCD buffer, SDRAM, and MPEEG HW accelerator.

I. INTRODUCTION

Recently, SoC (system-on-chip) design has faced new challenges that cannot be addressed by traditional design methodologies. Among them, the pressure of reducing the time-to-market in the presence of exponentially increasing design complexity has become critical in SoC design. New research activities around this issue have introduced the concepts of design reuse and platform-based design methodology [1-3]. Virtual platform [4,5] (or highly abstracted system model) based design methodology, which captures the concept of the platform-based design approach, has recently been gaining interest and recognition.

When we have been trying to make the platform-based design as a part of SoC designing process, there have been several issues which make the adaptation of the new design flow difficult. In particular, since the success of the project mainly depends on the early availability of fast and accurate system verification environment, we should create TL model as early as possible. [5-9] suggest several methods to adapt transaction-level modeling for architecture exploration with high-simulation speed. However, the previous works have not address the issue of systematic IP component modeling, which is targeted to practical large-scaled SoC applications.

In this paper, we proposed a systematic modeling process to achieve early availability and reusability of TL system prototype. The proposed framework has been successfully applied to a contemporary mobile design to illustrate how our approach has benefits the real world SoC design process. We prototyped embedded SW on this platform running more than 1000x faster than RTL designs, decided the optimal architecture of entire system by measuring the expected performance thanks to the pre-developed SW, which eventually reduced the design re-spin due to the performance issues. Furthermore, SW can be concurrently optimized while the architecture exploration is performed thanks to the cycle-accuracy guaranteed by the developed virtual platform.

II. BUS FUNCTIONAL TL MODEL

Let us begin with an example to show how the separated design approach, that is, to fix bus subsystem and then design IP according to the fixed bus protocol, limits the scope of IP optimization. Fig. 1 shows our target systems and movie capture operation flow which should be run on the system.

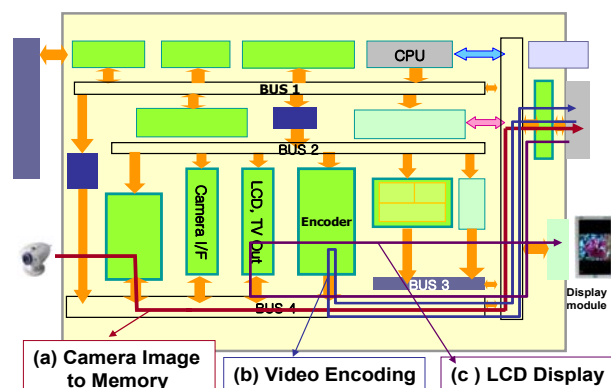


Fig. 1: Movie capture operation

Fig. 1(a) indicates a data flow - camera image to memory path. Camera I/F has two DMAs, preview and codec DMA. Using preview DMA, camera I/F sends images to LCD frame buffer. At the same time, another image data is sent to the memory and saved at the MPEG encoding source image area by codec DMA. For video encoding, the data in SDRAM is moved to MPEG HW accelerator, saved to SDRAM again after encoded by HW. Meanwhile, LCD display keeps running to display frame data in SDRAM received by preview DMA.

We initially decided to use AHB [10] bus system, and designed each IPs according to the bus protocol. However, after integration, performance was measured in terms of clock speed, and we found that the given requirement for the movie capture scenario is not satisfied. To fix the performance violation, we decided to change the bus system into AXI [11]. Since all the components were already implemented targeting AHB bus subsystem, these could not be integrated directly. We implemented AHBtoAXI bridges and re-integrated the system as shown in Fig. 2.

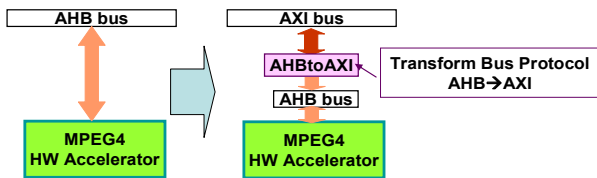


Fig. 2: Bus protocol translation

We performed performance analysis again, and Table 1 summarizes the results. As shown in the table, even though AXI bus system was applied with bridges, performance gain was not achieved contrary to our expectation, but even degraded about 19% compared to AHB bus system. It contrasts with the significant performance gain achieved by redesigning bus interface to fully utilize advanced features of AXI. This performance degradation was mainly caused from the semantic gap between the two different bus protocols. For example, in the above case, multiple threading, which is one of the most advanced features of AXI system could not be fully utilized because IPs were already designed based on AHB system. In addition, all the “INCR” transactions were translated into numerous numbers of “SINGLE” transactions by AHBtoAXI components. These heavily downgraded the system performance.

Table 1: Result of Performance Analysis

	AHB	AXI with AHB2AXI	AXI Interface
Execution Cycles(Kcycles)	2880	3440	2012
Performance Gain	-	+19%	-23%

This example strongly implies that both of the tasks of IP optimization and bus system exploration should be taken into account in an integrated fashion to fully exploit the advanced features of state of the art bus system. Also, to reduce the time-to-market pressure in the presence of exponentially increasing design complexity, to maximize reusability is seriously required. There are lots of requirements for a hardware macro to be fully reusable. Among them, the *generality* and *standard-based interface* are the main features. For the *generality* issue, the macro should be designed to solve a general problem – this often means the macro must be easily configurable to fit different applications. In addition, *standard-based interfaces* are also required – Unique or custom interfaces should be used only if no standard based interface exists. To support these features, the interface of macro should be designed as highly modularized and bus protocol independent not only in RTL but also in TL. However, in conventional SystemC modeling approaches, models are refined from a high-level functional specification down to a cycle-accurate model. Because bus interfaces and channels are refined from the abstracted form in this procedure, and such a modeling scheme cannot generate protocol-free bus interface.

The key factors that we consider in our IP design methodology are as follows: (1) for the engine part, TL model and RTL model should be reused regardless of the bus system change; (2) quantitative bus system exploration should be performed without any modification of TL interface; (3) Throughout the architecture exploration, optimal interface structure is decided, and RT level interface is obtained by interface synthesis. Without the above IP design flow, for example, if we designed the IP using multi-threading feature which is not supported by AHB bus protocol, we cannot use the IP for AHB bus system, which restrict the usage area. In the above IP design flow, engine part (TL and RTL domain) remains same, and only the interface part is explored, optimized, and synthesized. In the sequel, we explain our TL-interface design.

Bus protocol in a design specification is usually described at the signal level, so it is necessary to map signal-level protocol into transaction level protocol, which is performed by transaction-level ports (typically, transaction level ports are implemented as variables or functions).

The behaviors of each transaction-level port are modeled according to each burstX transaction scheme. Fig. 3 illustrates “SINGLE” transaction in AHB protocol. The procedure of sending “HBUSREQ” (bus request signal) and receiving “HGRANT” (bus grant signal) which is performed by RTL master can be represented as the following. TL-master check bus grant by “checkforgrant()” function call and if the master cannot access bus grant, it sends bus

request by “requestAccess” function call. Moreover, the procedure of sending an address, checking hready, and data read can be executed by “read(addr, *data, *ctrl)” function call. In this procedure, the TL-model checks “hready” by the return value of “read()” function.

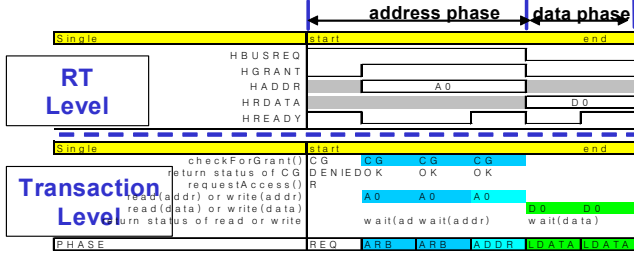


Fig. 3: SINGLE transaction in AHB protocol

As illustrated in the previous example, even for a comparatively simple bus protocol, if we define bus protocol in a specific bus dependent level, each of the data transaction requires complex description. As a result, engine part is strongly related to the specific bus protocol and this prohibits bus system exploration seriously.

Contrary to this, our TL bus protocol is defined as generic as possible. To support this, only the size information of address, length, and burst is explicitly described but all the other details are implicitly implemented in the separated library. Fig. 4 shows an example. As shown in the right hand side, IP designer uses only “read/write (addr, Ctrl)” function only. Therefore, we can design IP regardless of specific bus protocol, and this also enables easy adoption to bus subsystem changes.

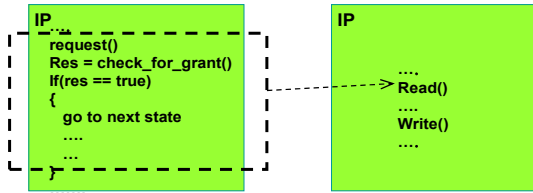


Fig. 4: Simple TL bus protocol

III. STRUCTURAL TL MODEL CREATION FLOW

In this section, we describe our structural TL model creation flow, which is performed in three steps. The main idea is to simplify the TL protocol to satisfy the requirements of *fast speed simulation* and *accurate modeling*. In each step, we check the accuracy of target IP by own developed simulator which measure $Q(X)$ value. The proposed TL model creation flow is performed in three steps.

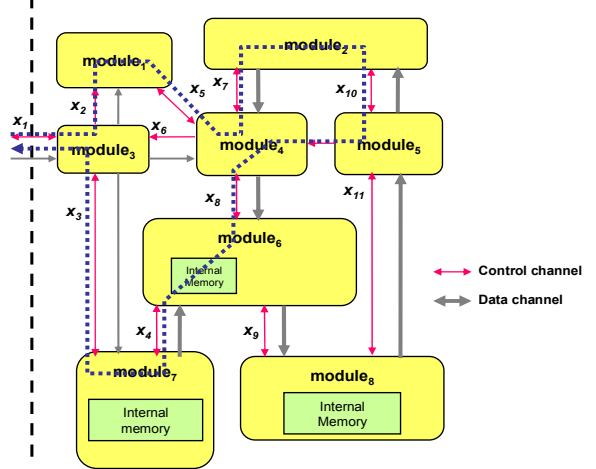


Fig. 5: Example showing the concept of structural TL model description

A. Step1 : Module structure design

We define the structure of target IP. Let *control channel* denote the signal which describes control and timing dependency between submodules, and *data channel* denote the path on which data information between the modules flows. In Fig. 5, control channels are represented by red lines, and data channels are represented by grey lines. Then, we also define *quality measure function* $Q(X)$ to control and justify the quality of the structure of TL model X . We compute the value of $D(x_i)$, which is the difference of signal arrival times of x_i between RTL (if it is top-down approach, it can be the spec) and the corresponding structural TL model, where $x_i, i=1, \dots, n$, indicates *control channel*. Suppose we have control flow streams $str_j = (x_{j1}, x_{j2}, \dots), j=1, \dots, n$, in IP X . For example, the control signal flow indicated by the dashed line in Fig. 5 corresponds to the stream $(x_1, x_2, x_5, x_7, x_{10}, x_{12}, x_8, x_4, x_3)$. We compute $CA(X, str_j) = (D(x_1) + D(x_2) + D(x_3) + \dots + D(x_m))/m$ and use the value of $CA(X, str_j)$ as the amount of the error in timing accuracy for the control flow stream str_j of the structure TL model. Then, the timing accuracy of the entire TL model for IP X is bounded by the quantity:

$$Q(X) = \max\{CA(X, str_1), CA(X, str_2), \dots, CA(X, str_n)\}$$

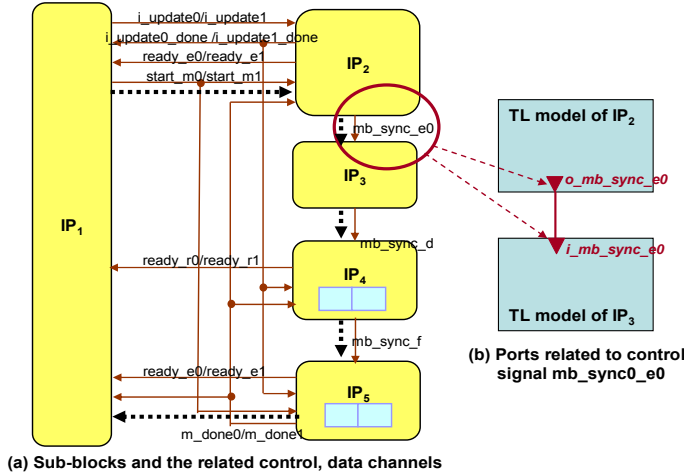


Fig. 6: Example of design spec. to be modeled

B. Step2 : Transaction protocol design

In this step, we model the transaction protocol, and check the feasibility of the design. We explain the creation of transaction protocols with an example. Suppose we want to transform the design in Fig. 6(a) into an executable TL model. Each of *control channels*, represented by arrows, is implemented with two ports, where one of them sends the corresponding signal-level protocol and the other receives the protocol. Fig. 8(b) shows the two transaction-level ports *i_mb_sync_e0* and *o_mb_sync_e0* corresponding to the control channel *mb_sync_e0*.

Then, we implement *data channel* (denoted by the dotted arrows in Fig. 6). Data to be transformed is abstracted by high-level transaction protocol, and transferred by the function call between the corresponding transaction-level ports. Note that we do not implement any additional port for data channel but reuse the transaction-level ports for control channel.

The functional behavior of each submodule will be described in each component while control and data interface are totally delegated to a dedicated routine called “interface handler” which is prepared for various communication scheme (for example, push type, pull type with defined delay, and pull type with undefined delay) with a unified programming interface. This enables easy adaptation to possible changes in the environment, and to achieve the enhancement of productivity and reusability.

C. Step3 : Internal FSM design

Finally, we model the internal behavior of each model. In this procedure, we assign the delay parameter to all function part and the size parameter to all the internal memory so that we can simulate the performance of different architectures easily. Then, we integrate the engine part which is described by pure C functions to the constructed frame. In this step, we should be very careful to consider the co-simulation with RTL design. The main idea is to separate state machine and registered variable parts from combinatorial logic parts that are connected to the RTL domain.

IV. MODEL VERIFICATION

A. Testbench reuse

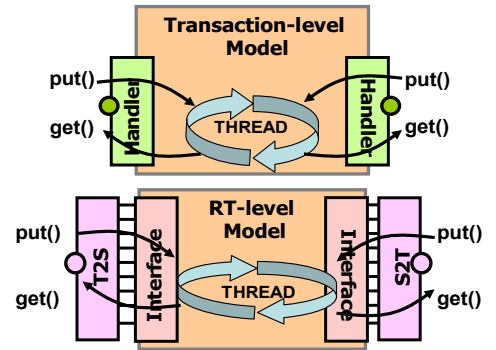


Fig. 7: The use of unified interface for testbench reuse

The most important consideration in verifying design is the reusability of testbench. It is necessary to create an environment in which the testbench used in the TL can be used in the RTL, or the testbench used in the RTL can be used in the TL. Furthermore, the environment should support simultaneous multiple IP verification. In our modeling method, we separate the functional and communication parts of each IP model as shown in Fig. 7. The main functional behavior is described in the core block while communication is totally delegated to a dedicated routine called “communication handler,” which is prepared for various communication schemes with a unified programming interface. Therefore, all the IPs use the unified bus interface scheme, and this enables an easy adaptation to possible changes in the environment, which include a system bus or even a whole new simulation environment. Fig. 7 shows the generation of an RTL verification environment to allow the reuse of the testbench used in TL, in which two additional submodules at the interface are needed; One (S2T) is to transform a set of signals to a corresponding transaction, and the other (T2S) is to transform a transaction to a set of signals.

B. Cycle accuracy test

To generate a legacy IP, it is essential to confirm the cycle-by-cycle accuracy of the IP. Particularly, in bus subsystem an accurate IP modeling is required for performance analysis. In our method, we first obtain a trace from the program of RTL testbench generation, and then update the TL IP module to accept the trace as input. Then, the cycle accuracy test is performed by comparing the outputs produced from the simulation of TL and RTL DUTs.

C. IP test in a system

Once the test of IP itself is completed, we complete the testing of IPs in the context of execution of the subsystem that contains the IPs. The subsystem may consist of a CPU, a basic bus system, including functional memory and a bus, and an interrupt controller. In our method, we mainly focused on the verification of the interrupt controlling scheme, SFR behavior, and DMA operations between the IP and memory. We simply implement a software to run on the CPU and check the above behavior using the software.

V. EXPERIMENTATIONS

We applied the proposed methodology to a contemporary mobile design. We created an TL prototype for the design and used it to find an optimal system architects with the selection of the most suitable bus system (among AHB, AXI, and OCP). Moreover, we decided an optimal interface architecture based on a quantitative analysis and an exploration of IP itself. Performance was measured in terms of clock speed, and the cycle accuracy of our platform was measured by comparing the simulation results against that of the RTL.

Bus system exploration for mobile application processor: In this experimentation, we are interested in finding the most suitable bus system for the mobile application processor shown in Fig. 1. Here is the detailed setting of our experimentation. Because our target is to know the feasibility of application scenarios for each bus system, we assumed worse situation than real application scenario, that is, all the IPs are fully pipelined by a frame unit and activated by video sync signal at the same time. Actually, such kind of behavior can be easily implemented in software. The interval length between vsync, as shown in Fig. 8, is determined by the given clock frequency. If video IP, including H.264, MPEG4, and post processor, which is activated at vsync, finishes the activation for one frame before the next vsync generation, the constraint for the IP, 30 frames per second, of the targeted application is satisfied. On the other hand, for IPs which have interface to the outside, such as LCD controller and camera interface, we check whether there exists a case such that FIFO is empty due to the heavy bus traffic. Fig. 8 shows an instance of

simulation when AHB bus subsystem is used. MPEG4 finishes one frame encoding before the next vsync generation, and it indicates that the encoding application can be performed in the current architecture.

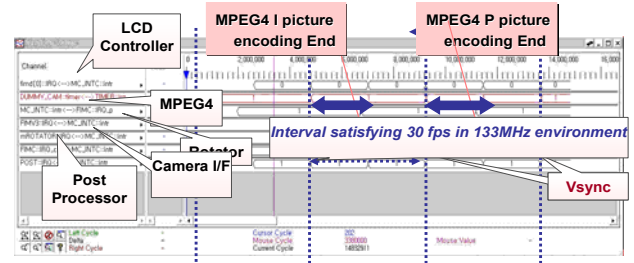


Fig. 8: A simulation snapshot

We explored various bus systems without any modification of TL IP. Among the operations, since MPEG4 encoding takes the longest execution time, we decide the feasibility based on the MPEG4 encoding time. In Fig. 9, we can see that architecture-b shows a better performance. However, because the performance constraint of the target system is also satisfied by the AHB system, which requires less circuit area and power consumption, we decide to use AHB bus system for this mobile application processor.

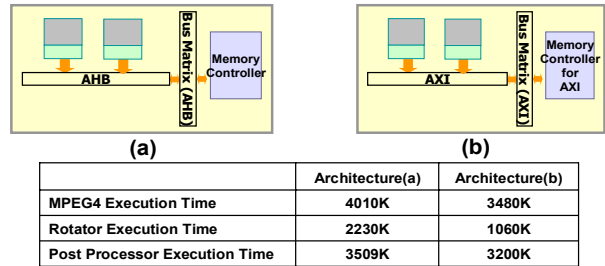


Fig. 9: Architecture exploration results

Interface exploration: We perform interface optimization focusing on transaction queue size. Our choice of transaction queue size is rather random, however, other design parameters can be evaluated easily. The testbench consists of one AXI master connected to masters PL300/PL340 to test the memory subsystem, and the master generates random transactions relentlessly without idle cycles. Regarding to the memory, we configured memory model as 16Mbx4x2 Mobile SDRAM. It is composed of two chips which include four 16 Mb banks. Bit width between the memory and interconnect was set to 16-bit. Timing parameters are summarized in Table 3, extracted from Mobile SDRAM K4S56163LC-RG/S75:133Mhz data sheet.

chip 1 addr	0x20000000 ~ 0x207FFFFFFF
chip 2 addr	0x30000000 ~ 0x307FFFFFFF
# of banks per each chip	4 banks
bus bandwidth	32 bits
memory bandwidth	16 bits

TRAS	5 clks
TRC	7 clks
TRCD	3 clks
TRP	3 clks
CL	3 clks
TRRD	2 clks
TRW	1 clks
TWTR	1 clks
Refresh cycle	655 clks

Table 3: Memory configurations and delay parameters

Data transfers between the memory and the master are counted during the simulation for 1M cycles. To measure the performance, we introduced memory utilization defined as:

$$\text{Memory_utilization} = \# \text{ of data transfer} / \text{elapsed cycles} \times 100 (\%)$$

Note that memory utilization will be 100% for ideal zero delay memory where data transfer happens at every cycle. Then, we performed experimentation for each memory burst length with different queue size varying from 1 to 8. Data transfer was counted between the memory and bus after 1M cycles. As shown in Table 4, with the increase of the number of active transactions, we can see memory utilization improvement clearly, and the improvement is saturated beyond queue size of 4.

Burst Length	Queue size : 1	Queue size : 2	Queue size : 3	Queue size : 4
Single	14.3%	21.3%	22.1%	22.4%
4	35.9%	62.0%	64.7%	64.9%
8	42.1%	74.7%	77.3%	77.8%

Table 4: Exploration results of an interface parameter

VI. CONCLUSIONS

We presented a systematic design of IPs and verification methodology for creating a virtual platform based design. The proposed method tried to solve two practical issues: (1) systematic TL modeling of IPs and (2) TL system model

verification. In the mean time, the proposed method was successfully modeled a movie capture application, and from experiments, we confirmed that the used virtual design was able to find an optimal system architecture, including bus subsystem and other component parameters, with more than 95% accuracy against RTL design.

Acknowledgment: This work is supported by Samsung Electronics, and the work of T. Kim is supported by the Ministry of Science and Technology / Korea Science and Engineering Foundation through the Advanced Information Technology Research Center (AITrc).

REFERENCES

- [1] K. Keutzer, et al., "System-level design: orthogonalization of concerns and platform-based design", *IEEE Transaction on Computer-Aided Design*, vol. 19, no. 12, December 2000
- [2] A. Sangiovanni-Vincentelli, et al., "Benefits and challenges for platform-based design", *Proc. of Design Automation Conference*, pp. 409-414, 2004
- [3] G. Smith, "Platform based design: Does it answer the entire SoC challenge?", *Proc. of Design Automation Conference*, pp. 409-414, 2004
- [4] S. Brini, et al., "A flexible virtual platform for computational and communication architecture exploration of DMT VDLS modems", *Proc. of Design, Automation and Test in Europe*, pp. 164-169, 2003
- [5] J. Notbauer, et al., "Verification and management of a multimillion-gate embedded core design", in *Proc. of Design, Automation and Test in Europe*, pp. 425-428, 1999
- [6] L. Cai and D. Gajski, "Transaction level modeling: an overview", *Proc. of Hardware/Software Codesign and System Synthesis*, 2003
- [7] I. Moussa, et al., "Exploring SW performance using SoC transaction-level modeling", *Proc. of Design Automation Conference*, pp. 120-125, 2003
- [8] A.K. Deb, et al., "System design for DSP applications in transaction level modeling paradigm", *Proc. of Design Automation Conference*, pp. 466-471, 2004
- [9] R. Jindal and K. Jain, "Verification of transaction-level SystemC models using RTL testbenches," *Proc. of Formal Methods and Models for Co-Design*, 2003
- [10] AHB CLI Specification www.arm.com/armtech/ahbcli
- [11] AMBA AXI Specification, [http:// www.arm.com/armtech/AXI](http://www.arm.com/armtech/AXI)