A Practical Approach for Bus Architecture Optimization at Transaction Level

Osamu Ogawa† Sylvain Bayon de Noyer‡ Pascal Chauvet‡ Katsuya Shinohara†
Yoshiharu Watanabe† Hiroshi Niizuma† Takayuki Sasaki† Yuji Takai†

†Semiconductor Company
Matsushita Electric Industrial Co., Ltd.
Nagaokakyo City, Kyoto Japan 617-8520
e-mail : {osamu-o, k_sinoha, y-nabe, niizuma, sasaki, takai}@scd.mei.co.jp

‡CoWare, Inc.
San Jose, CA 95131, USA
e-mail : {sylvain, pascal}@CoWare.com

Abstract

For multimedia applications, the System LSI design trend is to integrate an increasing number of applications running on a single chip. Traditional architectures have reached their limit in terms of performance. New architectures must be explored to fulfill the system application needs. Complex bus structures have been introduced. These bus architectures open a much larger exploration space than traditional hardware-software partitioning trade-offs. We have been researching methods to leverage these new architectural elements. We also introduce a design environment to apply practical and efficient methods in today's design flow. Two key technologies are supporting our method and environment: Automatic bus architecture synthesis for easy configuration of bus architecture and transaction level of abstraction for communication for improvement of simulation performance. In this paper, we show the design method, an overview of the design environment and its usefulness through experimental results.

1 Introduction

Consumer market trends are driving the integration of an increasing number of applications and features in a single product. Progress in semiconductor technology enables fabrication of larger chips increasingly. These two factors are pushing system LSI to the next generation of complexity. Furthermore, in the field of wireless, various performance-demanding multi-media applications such as voice, audio and moving pictures, are squeezed onto a single, low-power chip. The design trend has been to distribute applications on several processing units. Various CPU cores and DSPs share the processing load required by numerous applications. A resulting effect is the increasing amount of data exchanged on the system. Communication becomes the performance bottleneck. Traditional architectures are reaching their performance limits.

A number of new bus specifications have emerged to palliate this problem, introducing interesting elements to enable the optimization of bus architecture for performance. AMBA, for example, introduces multi-layer bus structures and interconnect matrix [10]. This opens a new space of bus architecture exploration: topology, connections flexibility, and freedom in configuring bus behaviors such as arbitration. However, relying on prior design experiences and experiments for the design of such complex bus architecture is not practical anymore. It would cause architectural errors and re-spins. Those troubles may result in slipping the projects, or even cancellation of the projects.

Several design methods have been proposed to address the distribution of applications onto such heterogeneous processing platforms. Some approaches are based on the system simulation ([1], [2], [3], [4], [5], [6]): The overall system is modeled in a high-level language for specification functional and performance validation, then decisions of hardware/software partitioning and bus architecture are based on quantitative results from simulation. These approaches enable the accurate evaluation of system architecture performance, but the development term of complex bus architecture modeling and the simulation performance of large system are critical issues. Others are based on the synthesis algorithm of bus communication architecture ([7], [8], [9]). However, we have recently seen a dramatic increase in the range of possible hardware bus architecture configurations, and their influence on optimum performance with a certain class of application. This is not adequately addressed by these approaches.

Thus, we propose a practical design method for bus architecture as the performance bottleneck to be optimized. Our method is based on the system simulation. We adopt two key technologies that ease the architecting of complex bus and improve significantly simulation performance: Automatic bus architecture synthesis and transaction level of abstraction for communication. Our goal is to support the method with a design environment to be practically applied today to the system LSI design process, and hence to improve time to market and quality.

This paper is organized as follows: Section 2 describes our design method. Section 3 describes the supporting design environment, and will detail key technology aspects: bus generation, and transaction abstraction level. Section 4 will present our case study demonstrating the usefulness of our design method and environment through experimental results. Section 5 gives the conclusions.

2 Bus Architecture Optimization Method

The primary technical objective will be to guarantee bus and memory bandwidth to handle the various interrelated data flows generated by distributed applications.

From a design process viewpoint, architectural decisions will have to take place at an early stage; this helps to remove loops of redesign caused by discovering architectural correction requirements late in the design cycle.

The expected outcome will be at first the "confidence that the architecture" can handle the applications and data flow with acceptable performance. Reliable analysis results should validate the architectural choice, thereby removing system architects intuition from the decision-making process.

Also, a series of recommendations can be delivered to system application engineers using the given platform architecture. For example: recommendation for an optimum dataflow scheduling on the defined architecture. With such information, software architects will be able to establish strategies for the embedded applications integrations.

The space available for the system architect to explore the architecture contains several dimensions. The most obvious will be the bus architecture topology itself with various possibilities of a number of buses, bus interconnections and bus master and bus slave device connections to the architecture.

And also, more interesting is the dataflow handling and scheduling within the bus architecture itself: Arbitration configuration, buffer insertion.

Several challenges will need to be addressed by the method and environment:

- Full system model is required as all applications have complex dependency and data flows interferences. Modeling a subset will not provide useful results. Furthermore, at early design stages, not all applications may be available.
- Accurate results are required. Real-time systems necessitate maintaining an efficient data-flow scheduling, to ensure a process is not served too late. The architect needs to track the worst-case scenario in the system, which cannot be done without a high level of accuracy, especially if the bus protocol is used versus packet-based communication.
- Long simulations are necessary to obtain valuable numbers. For example, video applications might need a large number of frames, which requires running for several million cycles. The goal is to cover as many different scenarios as possible.

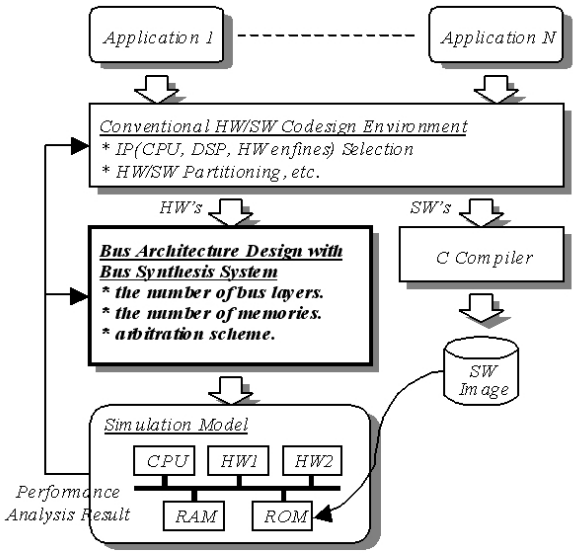

Figure 1: A Hardware/Software Codesign Method based on Optimization of Bus Architecture

The proposed optimization method is based on an exploration loop of a few tasks. The method as presented in Figure 1, follows these steps:

(1) Hardware/software partitioning of each application and IP is decided in the conventional approach.

(2) Designers create an architecture model for simulation by connecting the hardware model. CPU/DSP simulators and some common peripherals such as timer, interrupt controller, memory model and so on have been prepared as reusable libraries.
Even when modeled at high level of abstraction, hardware elements have an acceptable accuracy level for behavior and communication and are sensitive to clock signals.
Designers will model the application-specific hardware. Depending on the needs, models could contain real behavior described at a high level or performance models generating expected data flow.
Bus structural elements are predetermined by the chosen bus family: in our case AMBA Multi-Layer. Bus generators interfaced to the architecting tool will allow efficient bus network configuration.
Tentative clock frequency for buses, peripherals and cores are configured.
Software parts are compiled to binary for the target core.

(3) Designers validate the performance of the architecture based on quantitative results from executing the software on the architecture model. Targeted analysis tools will be used for bus and memory analysis in relation to behavior execution: utilization of bus and memory, time to service, latency, contention and so on.

The process will loop back to (2) or (3) and explore various architecture alternatives until the optimum architecture is determined.

As seen earlier, a number of elements will be involved in order to apply the most practical and efficient method.

Our design environment provides a solution for each of these requirements: simulation performance, model accuracy, ease of platform creation and architecture configuration, ease of modeling hardware, analysis tools targeted to the configured architecture and libraries of CPU/DSP models and common peripherals.

3 Implementing the Design Environment

When defining the design environment, our goal was clearly to address all the different technical challenges listed in the previous chapter.

Speed: C. simulation is a must when simulation speed needs to be improved. An innovative solution has been adopted to model the communications in the system: Transaction Level Modeling (TLM).

Accuracy: The platform will be described in a cycle accurate fashion. Both behavior and communication will follow the same rule. Cycle accurate CPU ISS (Instruction Set Simulator) is preferable to a statistical CPU model (the level of accuracy would not be satisfying enough).

Flexibility: A bus architecture synthesis system supporting a multi-layer bus in our design environment enables designers to change the architecture more easily and can generate simulation models at the transaction level with high speed and high accuracy.

3.1 Overview of Bus Architecture Synthesis

The modification of complex, multi-layer bus architecture models by hand is not practical for exploration of multiple architectures. It also increases the amount of work necessary for the verification of any change in the bus architecture. Therefore, we implemented a bus synthesis function into our design environment in order for designers to practice the design flow shown in Figure 1 more easily.

The inputs to the bus synthesis system are as follows:

(1) Connection information between hardware components

Designers create a net list with hardware components and bus components connected at one junction as shown in Figure 2(a). The bus network topologies are not decided at this time.

(2) Bus network topology

The designers make a bus network topology using bus components prepared as libraries and choose an arbitration scheme from library to each bus component with multiple masters (refer to Figure 2(b)). Full AMBA 2.0 specification including complex bus activities such as split and retried transfers has been enabled thru AHB, APB and interconnect matrix generators.

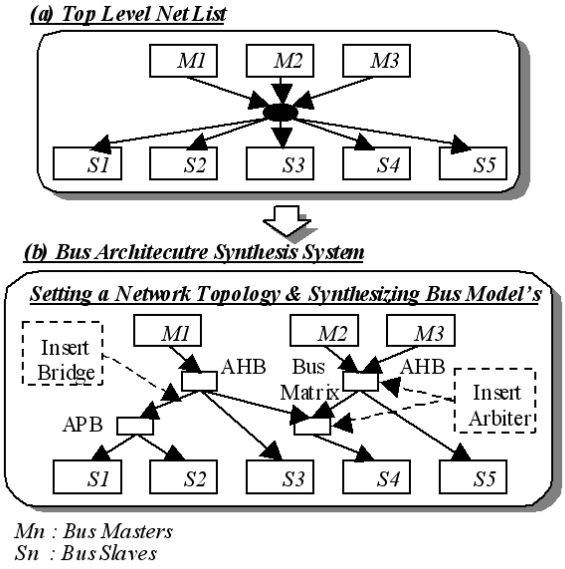

Figure 2: Overview of Bus Architecture Synthesis

(3) Memory mapping information for each bus master

The bus synthesis system generates a CoWareC description for each bus component with appropriate address decoders and multiplexers and bus bridges between bus components. Arbiters are inserted automatically for bus components with multiple masters (any node with more than two inputs as shown in Figure 2(b)). Our design environment has prepared a frame using C API's to implement a user-defined arbitration scheme, so any arbitration scheme can be implemented. The designer can also use the CoWare N2C analysis system (e.g. bus utilization, bus contention and bus arbitration) since the description to link to the analysis system can be added to each bus component model.

The bus synthesis function enables designers to build various architectures that have different numbers of bus layers easily in a short amount of time. The use of the analysis system enables the designer to perform architecture exploration more effectively.

3.2 Abstraction Level of Bus Model

The communications are becoming more and more important in the system. Modeling the communications at the transactional level is a way to minimize, during a simulation, the amount of information processed by the simulator. Instead of driving all the signals of the bus (RTL-like approach) an initiator block and a target block will only exchange what is really necessary (payload). Because every initiator and target block are connected to the bus abstract based on the way they communicate, the bus is the one implementing the details of the protocol. The bus model is generated by the bus architecture synthesis.

In general, bus accesses are done in the following order: bus request, bus grant, bus read or write access, bus release. The challenge is to maintain the cycle accuracy of the four protocol events above. Communication of our bus model at the transaction level groups several factors such as address, data and control signals in protocol in only one entity: a transaction.

We explain the communication mechanism of our bus model at the transaction level using Figure 3.

(1) First of all, the bus model receives a bus request signal from a bus initiator and sends a bus grant signal to the selected bus initiator after arbitration. Since the arbiter is a cycle accurate model, the accuracy from the bus request timing to the bus grant timing can be maintained.

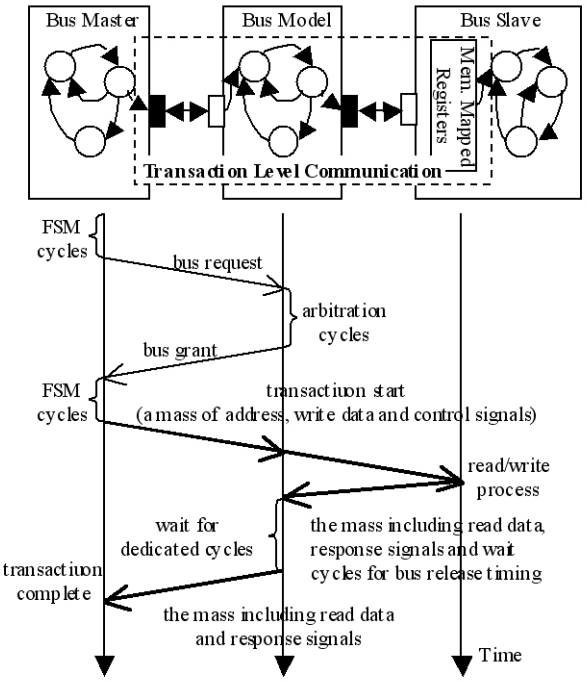

Figure 3: Abstraction Level of Bus Model

(2) The selected bus master sends a transaction containing address, data and control signals to the bus model. The cycle accuracy from the bus grant timing to transaction occurrence timing depends on the cycle accuracy of the initiator model.

(3) The bus model decodes the address and forwards the transaction to the appropriate bus target along with the timing specification of the bus.

(4) After receiving the transaction and completing the process of the access, read data and response attributes are set into the same transaction. The targets also set the number of wait cycles necessary and then return the transaction to the bus model without cycle delay.

(5) The bus processes the wait states defined in the transaction. Then, the bus is released to complete the access.

As stated above, our bus model provides a frame to maintain cycle accuracy of communications. Since behavior is also cycle accurate, the whole system becomes cycle accurate. Also, the validation of cycle accuracy and function of CoWareC bus model at transaction level generated by the bus synthesis system were done by confirming the waveforms.

4 Experimental Result

We experimented on a test situation to validate our method for architecture optimization and measure the effectiveness of the design environment.

4.1 Architecture Exploration

The example system executes a graphics calculation and sends the data to the LCD. As shown in Figure 4, the platform consists of ARM, GE (Graphics Engine) and DCU (Display Control Unit) as bus master, and two or three memories and some peripherals as bus slaves. GE executes the graphics calculation, and DCU reads the graphics data from memory and sends it to LCD. The ARM is modeled using a cycle accurate ISS. The software (Dhrystone Version 2.1[11]) is executed on ARM. By using this example, we attempted to explore the performance of multiple architectures for:

- Optimum number of bus layers.
- Optimum arbitration scheme.
- Optimum number of memories from the viewpoint of the hardware engines strategy.
- Optimum memory location from the viewpoint of software strategy.

As shown in Figure 4, we built four bus architectures with different numbers of bus layers and different numbers of memories using the bus synthesis system.

Architecture N (N=1,2,3,4) uses RAM1 shared by GE and DCU as a work memory, and Architecture N' (N'=2,3,4) uses Work RAM for ARM's exclusive use as a work memory. As for Architecture M (M=2,3,4) and N', we tried out two arbitration schemes: One is a fixed priority and another is a user-defined TDMA (Time Division Multiple Access). In the case of the fixed priority scheme, we set priority for each bus component as DCU had the highest priority and ARM had the lowest priority as shown in Figure 4. And then, Architecture 3/3' and Architecture 4/4' have RAM2 for GE and DCU to maximize the operation rate of hardware engines in unit time. We programmed the software as GE and DCU processed data of one frame size and executed it. Figure 5 shows a distribution graph of the operation rate of ARM and GE/DCU as the validation results for each architecture. As for the operation rate of ARM, we

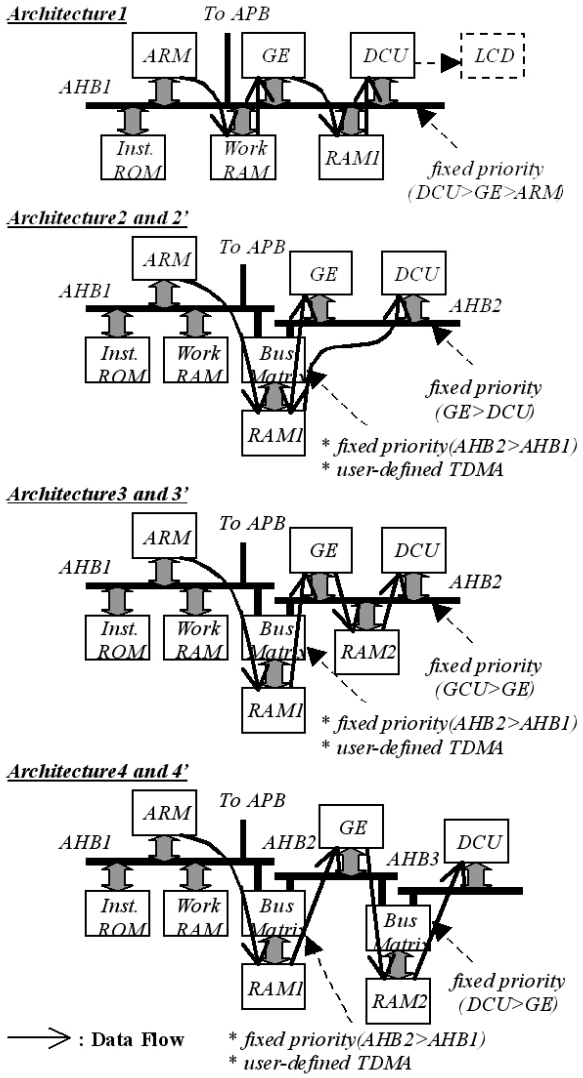

Figure 4: Evaluation Architectures

calculated it using the number of iterations of Dhrystone. As for the operation rate of GE/DCU, we calculated the simulation cycle counts in Table 2 which GE and DCU required in order to process data of one frame size. In this way, we could find out the quantitative performance of thirteen different architectures for one of the candidates of hardware/software partitioning by changing the number of bus layers, arbitration scheme, memory location as a software strategy or adding a memory for GE and DCU. By the way, architectures which maximized the operation rate of both ARM and GE/DCU were Architecture 3, 3', Architecture 4 with fixed priority arbitration scheme and Architecture 4' with user-defined TDMA arbitration scheme. Also, these are shown by the dotted line in Figure 5.

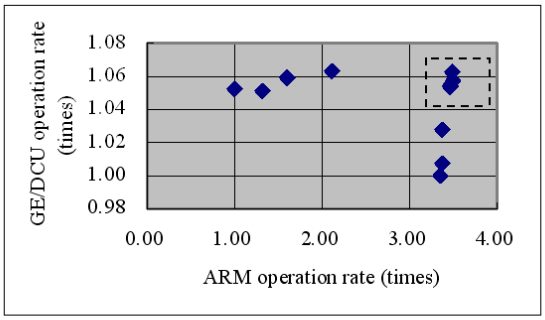

Figure 5: A Distribution Graph of Process Rate of ARM and GE/DCU

We got these results in only one day including examination of these results, provided that it took another five days including verification term to develop the performance models for GE and DCU. In the design of large and complex system LSIs, it is very important and very effective to validate the performance of the architecture based on accurate quantitative results early in the design process in order to minimize time consuming feedbacks from late design stages. And then, our design environment can use analysis functions on CoWare N2C. We think these functions will enable designers to be more effective in their jobs.

4.2 Bus Architecture Synthesis

As shown in Table 1, the bus synthesis system required only one or two minutes on a PC (Intel Pentium IV 2.4GHz, RDRAM 512MB, LINUX) to generate an architecture. Modifications done manually require a full day, including verification time, and furthermore the designer does not have to verify the bus models because they have been verified enough. Thus, designers can devote themselves to validate the performance of various architectures because they can be free from the modifications of architectures by hand.

Table1: Simulation Build Time

		Bus Synthesis	Sim. Build	Total
Arch. 1	F	8.54 sec	59.21 sec	67.75 sec
Arch. 2	F	12.55 sec	64.50 sec	76.60 sec
	T	12.42 sec	64.81 sec	77.23 sec
Arch. 3	F	13.70 sec	67.18 sec	80.88 sec
	T	12.71 sec	59.67 sec	72.38 sec
Arch. 4	F	18.90 sec	69.08 sec	87.98 sec
	T	18.64 sec	72.98 sec	91.62 sec

* F: use fixed priority arbitration scheme
* T: use user-defined TDMA arbitration scheme

4.3 Simulation Performance

We measured the simulation performance of each architecture model with one of the test programs for the architecture exploration. We used a PC (Intel Pentium IV 2.4GHz, RDRAM 512MB, LINUX) for this measurement. We show the results in Table 2. In Table 2, Simulation Cycle Count is the number of cycles that GE and DCU require to process data of one frame size, and Simulation Time is CPU time, which the simulation requires. We achieved approximately 155–230 kilocycles per second and could execute the test program in several seconds. This simulation performance at the transaction level is about thirty times faster than that in CoWareC at RTL and enables validation of architecture performance using many software programs executed on the ISS. Thus, we think that this simulation performance is fast enough to lead designers to more effective architecture exploration.

Table2: Simulation Performance

		Simulation Cycle Counts	Simulation Time	Cycle Per Second
Arch. 1	F	877744	3.74 sec	234691
Arch. 2	F	872128	4.32 sec	201881
	T	923724	4.90 sec	188515
Arch. 2'	F	878763	3.99 sec	220241
	T	916860	5.90 sec	155400
Arch. 3	F	869407	4.64 sec	186372
	T	873640	4.71 sec	185486
Arch. 3'	F	876658	4.37 sec	200608
	T	876483	4.56 sec	192211
Arch. 4	F	869020	5.43 sec	160041
	T	868946	5.13 sec	169385
Arch. 4'	F	898722	5.27 sec	170535
	T	876086	5.00 sec	175217

* F: use fixed priority arbitration scheme
* T: use user-defined TDMA arbitration scheme

5 Conclusion

We proposed a design method of optimization of multi-layer bus architecture early in the design process before RTL implementation. We also developed a design environment to practice our design method easily on CoWare N2C for LINUX and validated the performance of thirteen architectures by testing the method with an example system on it. In the test, we tried to not only explore an optimum topology of multi-layer bus architecture but we also explored a way to maximize the performance of each architecture. We changed the memory location as a software strategy or we added a memory for maximizing the operation rate of hardware engines in time unit. It took only one day for the test. It took only about ten minutes to generate seven complex architecture models with a multi-layer bus at the transaction level and to build the simulation. Simulation performance of the architecture models achieved about 155-230 kilocycles per second and the software could be executed on an ARM ISS fast enough. In this way, we confirmed the effect of our design method and usefulness of our design environment with the bus architecture synthesis system.

Acknowledgements

We would like to thank all members of this project at CoWare, Inc. and Matsushita Electric Industrial Co., Ltd. Thanks are also due to reviewers and the editor for their valuable comments.

References

[1] T. Kambe, A. Yamada, K. Nishida, K. Okada, M. Ohnishi, A. Kay, P. Boca, V. Zammit and T. Nomura, "A C-based Synthesis System, Bach, and its Application," in Proc. of ASP-DAC, p151-155, 2001.

[2] P. Chou, R. Ortega and G. Borriello, "The Chinook Hardware/Software Co-Synthesis System," in Proc. of ISSS, 1995.

[3] K. Van Rompaey, D. Verkest, I. Bolsens and H. De Man, "CoWare – A Design Environment for Heterogeneous Hardware/Software Systems," in Proc. of EuroDAC, 1996.

[4] F. Balarin et al., "Hardware-Software Co-Design of Embedded Systems, The POLIS Approach," Kluwer Academic Publishers, 1997.

[5] R. Gupta and G. De Michelli, "Hardware-Software Cosynthesis for Digital Systems," in IEEE Design and Test of Computers, 1993.

[6] Osterling, T. Benner, R. Ernst, D. Herrmann, T. Scholz and W. Ye, "The Cosyma System," in "Hardware/Software Co-Design: Principles and Practice," Kluwer Academic Publishers, 1997.

[7] L. Freund, D. Dupont, M. Israel and F. Rousseau, "Interface Optimization During Hardware-Software Partitioning," in Proc. of Codesign Workshop, 1997.

[8] J. M. Daveau, G. F. Marchioro, T. Ben-Ismail and A. A. Jerraya, "Protocol Selection and Interface Generation for HW-SW Codesign," IEEE Trans. VLISI Systems, vol.5, no.1, 1997.

[9] T. Yen and W. Wolf, "Communication synthesis for distributed embedded systems," in Proc. ICCAD, Nov. 1995.

[10] http:/www.arm.com/armtech/AHB?OpenDocument

[11] http:/www.atmarkit.co.jp/icd/root/12/5785812.html