對以智財單元為基系統晶片設計之驗證與測試技術開發研究
子計畫一:與組織探索階段互動之系統階層驗證技術
System-Level Verification Interacting with Architecture Exploration

計畫編號：NSC 89－2215－E－009－119－
執行期限：89 年 8 月 1 日至 90 年 7 月 31 日

主持人：董蘭榮　國立交通大學電機與控制工程學系
共同主持人：沈文仁　國立交通大學電子研究所
計畫參與人員：
魏廷勳　國立交通大學電機與控制工程學系
林國鼎　國立交通大學電機與控制工程學系
吳俊銘　國立交通大學電機與控制工程學系
彭仁俊　國立交通大學電機與控制工程學系
李彥霖　國立交通大學電機與控制工程學系
蔡仁傑　國立交通大學電機與控制工程學系
Email: lennon@cn.nctu.edu.tw

一、中文摘要

系統晶片設計涵蓋很廣的設計空間。設計者通常需要考量許多可能的系統組織包括選擇演算法則、挑選組織元件、建構候選組織。設計如此的複雜系統誠屬不易，而要設計出能完全符合要求、正確無誤的系統更為困難。設計上的失誤必須要儘早排除，否則在後續階段才發現的失誤將造成耗費耗時的再設計周期。因此，設計者必須面對兩項課題，其一是實現設計程序本身、另一是建立正確的設計結果。其中，設計的正確性將為本計劃的主軸。此子計劃第一年之工作為提出組織元素的成本模型、發展成本評估核心公式、定義效能模型資料結構、發展基礎效能模型。

關鍵詞：效能模型、系統階層驗證、系統晶片

Abstract

The System-On-Chip (SOC) design encompasses a large design space. Typically, the designer explores the possible architectures, selecting algorithms, choosing architectural elements, and constructing candidate architectures. Designing such a complex system is hard; designing such a system which will work correctly is even harder. Design errors should be removed as early as possible; otherwise, errors detected at the later stages will result a costly, time-consuming redesign cycles. Thus, the designer should face two distinct tasks in SOC design; carrying out design process itself and establishing the correctness of a design. Design correctness is the main theme of this project. In the first year, the tasks of this project are: proposing cost models of architecture elements, developing cost estimation engine, defining the data structure of performance modeling, developing the fundamental performance models

Keywords: performance modeling, system-level verification, system-on-chip

二、緣由與目的

Intellectual Property (IP) reuse is becoming essential in system-on-a-chip design [1][2][3]. We have developed a front-end system level verification environment, that assists designers in exploring architectures based on a reusable

IP library. The library is composed of generic components that can be tuned to embrace a large number of third-party and in-house IP specifications. Using the library, the environment maps algorithms onto architectures, allowing for hardware/software partitioning and interrupt-based task scheduling. The environment then generates a time-accurate prototype of the candidate architecture, capturing performance measures such as processor utilization, memory size, bus utilization, and interrupt overhead. Based on these measures, the environment helps designers perform architecture exploration and make early decisions about IP selection.

Very few tools exist at the front-end, architectural level of IP-based system design. Traditionally, designers develop algorithms, select IP components, and construct architectures sequentially within separate design environments. A number of difficulties can arise from the lack of a unified design environment: algorithm development may fail to consider implementation details; IP selection may not be the most suitable for certain architectures; and architecture synthesis cannot easily experiment with alternative algorithms. The gap between algorithm development and implementation makes correct and speedy development very difficult. In order to make good early decisions on architecture selection, designers usually need to overcome the integration problems of CAD tools. Therefore, our primary goal has been to provide a front-end, integrated architecture exploration environment allowing designers to describe their target systems entirely and estimate performance measures at a very early stage in the overall design process.

The proposed verification methodology partitions the design process into three phases: cost estimation, performance modeling and hybrid co-simulation. The first phase estimates the architecture costs in terms of power, area, and delay. By interacting with architectural allocation, the cost estimation verifies that the architecture adhere to system requirements. Following the first phase, performance modeling then simulates candidate architecture at the performance level of abstraction. Models at this level of abstraction do not concern actual data in the system, but rather the flow of data through the system. Hence, the second phase takes the advantage of simplifying the complexity of the simulation and modeling. Using the simulation results in terms of time-critical constraints, the designer is able to verify the candidate architectures and, if necessary, refine the design. Once finishing the first two phases, hybrid co-simulation starts to perform the simulation of heterogeneous architectures which contain hardware and software instances. This phase, therefore, verifies the behavior of the SOC design. Up to this point, the system-level verification has been done and establishes the correctness of a design at the system-level.

三、結果與討論

In the verification environment, we use three axes, {algorithm, attribution, structure}, to represent architectures. The embedded architectures can be defined in a three-dimensional representation space. The algorithm axis denotes the procedure and tasks for the solution of a given application. The attribution axis defines the attributes of architectural components such as type of processors, communication protocol, size of memory, and so on. The structure axis represents topology, arrangement of components, and connections. Using this three-dimensional architecture representation, designers are able to perform different architectural tradeoffs, simultaneously considering algorithm selection, component specification, and architecture allocation within an integrated environment.

After describing an architectural solution in the three-dimensional representation, the system-level verification generates a simulation model which consists of VHDL models from a reusable IP library. We completely separate the functional simulation from the performance simulation, in order to reduce the complexity and the time required for performance estimation. The aspects considered in our modeling

technology are: sizes of processing data, time-related parameters, and the sources and destinations of communication tasks.

Each node of the simulation model consists of three constituent entities: task model, memory model, and communication module. These three entities model computation, data storage, and communication elements, respectively. The task model represents the computational and local data-access-related latencies for local processing. The latter includes latencies resulting from requests sent by other nodes that communicate through the interconnection. In the system-level verification, both hardware and software instances are modeled as task models; that is, hardware and software instances are modeled at the same level of abstraction. Thus, designers can use either software or hardware instances to implement function tasks [2]. This modeling strategy, called *hardware-software neutralization*, makes our hardware-software tradeoffs efficient and handy.

The memory model models memory I/O, caching and data access in terms of interface bandwidth, data block sizes, associated latencies and so on. The memory model is used to estimate the usage of memory space and simulate data exchange in the target system. The use of the memory model is optional. Simple nodes, such as DMA, controller, and display, might not need the Memory Model.

The communication module represents the communication protocol and configurations at a high level of abstraction, using a set of standard signals. The model is a time-faithful model that accurately estimates the communication and data-movement overhead for complex communication protocols. Using the standard signals, the model directly communicates with the task and memory models in a plug-and-play fashion. This feature leads to a high degree of modularity; that is, designers can seamlessly link task and memory models to communication modules. The plug-and-play feature benefits architectural tradeoffs. For instance, the algorithmic tradeoffs can be done by plugging in different task models; the tradeoffs of communication subsystems can be done by replacing communication modules.

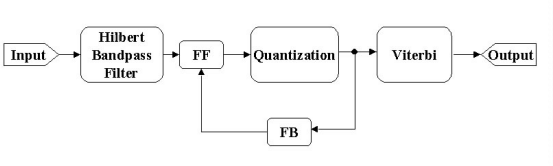

Fig.1: A simplified ADSL algorithm

Now we use a simplified ADSL application to demonstrate the architecture exploration process and show the capabilities of the verification environment. Given the algorithm in **Fig.1**, architecture exploration will involve: (1) architecture specification, (2) timing verification, and (3) performance estimation.

To start with, we define the structure of the candidate architecture in terms of interconnect topology, communication protocol, and node connections. Then, we use the data flow graph tool to describe algorithm in terms of task assignment, execution time, data dependency, and data volume. Finally, we use different user-interface forms to set architectural attributes for task models, memory models and communication modules. The fields of forms vary with the type of component. At this point, the system-level verification is ready to generate the simulation model.

After architecture specification, we generate and compile the simulation model. The model is created in VHDL. Designers can use a third-party VHDL simulator to simulate the model and verify the timing chart. This step helps designers to find major design oversights and generates output data for performance estimation.

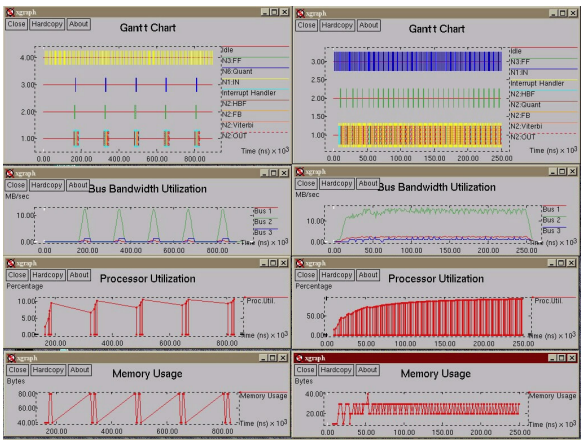

Fig.2: Simulation results

Once the simulation has been successfully done, the simulation model generates output files which contain interesting data with respect to computation, communication and data storage. We then use the performance estimation function to analyze the files. As shown in Fig.2, the system-level verification plots graphs for performance measures. In the figure, we can see two different results generated from two different architecture specifications. Using these results, designers can make early design choices. The simulation time of each architecture is short (< 10 seconds). According to our experience in applying the environment to a rather complex application with six DSP processors, 36 tasks and 1 ms of simulation, the overall simulation time is less than ten minutes.

四、成果自評

本計畫第一年已建立基礎成本與效能模型，可有助於組織探索階段完成驗證工作。本計畫之研究成果已發表下列兩篇國際會議論文與一篇國內會議論文：

1. Lan-Rong Dung, Yen-Lin Lee, Chun-Ming Wu, “A Reconfigurable Architecture for DSP System-on-a-Chip,” SCI2001
2. Yen-Lin Lee and Lan-Rong Dung, “The Configurable Scheduler for IP-based SOC Synthesis,” VLSI/CAD Symposium 2001
3. Lan-Rong Dung, Yen-Lin Lee, Chun-Ming Wu, “A Reconfigurable Architecture for DSP SOC,” IWMATT2001

另外，部分研究成果將發表於 Canadian Journal of Electrical and Computer Engineering **第四季期刊。**

經由本計畫之執行已培養六名碩士畢業生。該六名碩士畢業生目前服務於系統晶片相關之高科技企業。

五、參考文獻

[1] Lan-Rong Dung, Vijay K. Madisetti, and John W. Hines, “Model-Based Architectural Design and Verification of Embedded DSP Systems,” 1996 VLSI Signal Processing Workshop, October 1996

[2] Lan-Rong Dung, Mohamed Ben-Romdhane and Marius Vassiliou, “IP-Based Architecture Exploration,” DesignCon99, February, 1999

[3] Mohamed Ben-Romdhane, Marius Vassiliou and Lan-Rong Dung, “Rapid Prototyping of Multimedia Chip Sets”, ICASSP-99, March,1999

[4] Richard Goering, “New Tools will Force Embedded Designers to Link Hardware/Software Efforts – Codesign turns workplace on its head,” EETimes, Issue:988, January 12, 1998

[5] J.K. Adams and D.E. Thomas, “The Design of Mixed Hardware/Software Systems,” Proc. Of 33rd DAC, 1996, 515-520