針對單晶片系統界面協定之驗證(1/3)

計劃編號: NSC 94-2220-E-009-040
執行期間:94 年 8 月 1 日起至 95 年 7 月 31 日
主持人：周景揚 教授 國立交通大學電子所

一. 中文摘要

在系統單晶片的時代中，晶片設計人員將大量的矽智產整合至一個系統當中已是一種趨勢。為了增加這些矽智產的重覆使用性及加速整合的速度，矽智產通常會依循著特定的界面協定而設計。今日的界面協定為了提供更高速、更具有彈性的使用，其規格也愈益複雜。因此，驗證這些矽智產單元群是否吻合其傳輸界面，能夠在整合後正確地溝通資料，便成為系統單晶片驗證上的一大課題。

一般的界面規格往往是使用自然語言或時序示意圖來描述，在功能驗證的問題中，首先會遇到的問題便是如何將這些界面規格轉化為一個適當而精準的表示法。我們可以用這種表示法來規範界面行為，進而用來偵錯及除錯，我們在第一年裡的主要成果就是發展一種適合界面規格的表示方式，並自動合成協定檢查器，用來檢查待測矽智產是否有功能上的錯誤。

關鍵字: *設計驗證；協定檢查器*

英文摘要

In the system-on-a-chip (SOC) era, designers tend to integrate a large number of IP (Intellectual Property) components into a system. To increase the reusability of the IPs and facilitate the integration, IPs are usually compliant with certain interface protocol, then they can concordantly communicate with each other within the system. For high speed and flexibility considerations, the specification of interface protocol gets more complex today. Therefore, to verify if the IPs can work correctly after integrating into a system becomes a big issue in SOC verification.

The interface specifications are often written in natural language or timing-diagram. The first problem of functional verification is how to translate the specification into a more precise description. We can model the behavior of interface protocol with a well-defined description, and then use it to detect errors and debug. We develop a description style which is suitable for interface specification and automatically synthesis a protocol checker to detect any protocol error violating the specification.

Keywords: *Design Verification; Protocol Checker*

二. 計劃的緣由及目的

In modern system-on-a-chip (SoC) designs, certain number of building blocks should be reusable intellectual property (IP) cores to accelerate the design process. To achieve an even higher level of reusability, the platform-based design methodology, in which all IP cores are pre-verified, is usually adopted [1]. An IP core is wrapped with the appropriate interface (I/F) logic that complies with certain I/F protocol (usually an on-chip bus protocol) so that it can concordantly communicate with other IP cores within the system. When the IP core is desired in another platform utilizing a different I/F protocol, all one has to do is simply changing the I/F logic wrapper without altering the core function logic. Thus, by separating the core function logic from the I/F logic, the IP core can be easily and quickly integrated into different system platforms utilizing different I/F protocols [2]. In addition, even under a given I/F protocol, the I/F logic can still vary in a big dynamic range due to numerous valid configurations and options. Therefore, the interface compliance must be verified thoroughly during SoC integration.

There are two major categories in the field of interface compliance verification: simulation-based (dynamic) methods and formal (static) ones. The simulation-based verification approach is age-old but popular. Several studies are based on this approach. In [3], the properties of a protocol specification are represented in HDL monitors. These HDL monitors are simulated along with the design under verification (DUV) to determine its correctness. In [4][5][6][7], methods of automatically generating monitor circuits, coverage metrics, or verification patterns from high-level specification styles are proposed. In [8], a commercial tool ACT is developed to facilitate the AMBA [9] compliance verification. In general, all simulation-based approaches suffer from this common false positive problem.

Formal verification can avoid such false positive problem. Model checking [10] techniques are used for I/F protocol compliance verification in [11][12][13]. In these works, properties of the I/F protocol are specified in the CTL language. Then the model checker verifies the DUV against these properties. Once the model checker reports a success, the design is guaranteed to be 100% compliant with these properties. However, properties in CTL are not easily thorough and the process of extracting properties from a specification document written by natural languages is generally complicated and painful. It is very likely that some properties are actually implied by the specification but accidentally not extracted and thus ignored during the formal verification. Moreover, memory explosion and excessively long runtime may be further serious problems while the design size increases. All these issues potentially prevent model checking techniques from being effectively and efficiently applied on interface compliance verification.

Recently, the assertion-based verification (ABV) methodology is getting popular and several property specification languages (such as PSL [14], OVL [15], OVA [16], and SVA [17]) are developed to provide alternative ways to specify properties in additional to CTL. These emerging languages are relatively more easily understood than CTL at the semantic and syntactic

level. However, no matter which emerging property specification language is used, either the (simulation-based) dynamic ABV or (formal) static ABV inherently suffers from the same problems described earlier.

Our approach intends to verify whether the I/F logic is compliant with the I/F protocol at the FSM level. The properties of the I/F protocol are specified as a specification FSM. We believe that the FSM style is relatively more readable and systematic than other specification styles proposed in [3][4][5][14][15][16][17][18][19] and thus enables complete property extraction. The golden specification FSM is only created once for a specific I/F protocol and then can be used to verify all designs claimed to be compliant.

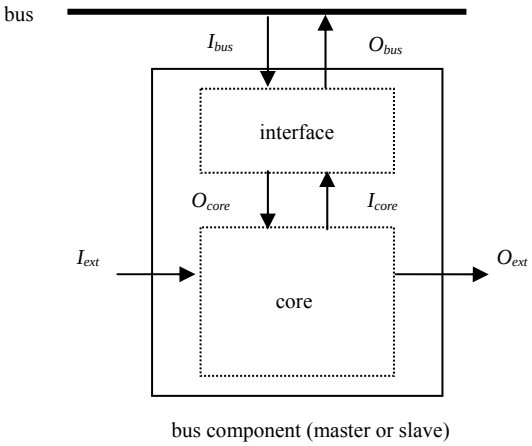

Figure 1. Notations of bus signals.

三. 研究方法及成果

Typical I/O signals of a bus interface logic are shown in Figure 1.

I_{bus} the set of input signals from bus to I/F
O_{bus} the set of output signals from I/F to bus
I_{core} the set of input signals from core to I/F
O_{core} the set of output signals from I/F to core
I_{ext} the set of external input signals to core
O_{ext} the set of external output signals of core

In addition,

I_{ctrl} $I_{ctrl} \subseteq I_{bus}$, the subset of bus inputs that directly controls the bus behavior from the protocol perspective

O_{ctrl} $O_{ctrl} \subseteq O_{bus}$, the subset of bus outputs that directly controls the bus behavior from the protocol perspective

We use the AMBA AHB slave interface as an example,

I_{bus} = { HSEL, HREADYin, HADDR, HWRITE, HTRANS, HSIZE, HBURST, HWDATA,

HMASTER, HMASTERLOCK }

O_{bus} = { HREADY, HRESP, HRDATA, HSPLIT }

I_{ctrl} = { HSEL, HREADYin, HTRANS }

O_{ctrl} = { HREADY, HRESP, HSPLIT }

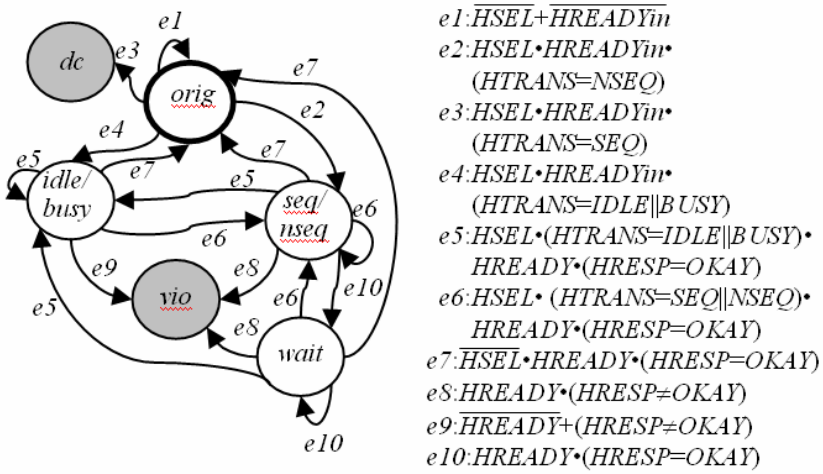

Figure 2. The spec FSM of a simplified AMBA AHB slave protocol.

It is not uncommon that just a small portion of bus I/F signals are classified into I_{ctrl} and O_{ctrl}. For example, what value the address/data bus exactly carries does not affect the bus behavior at the protocol level. Also note that I_{core}, O_{core}, I_{ext}, and O_{ext} may differ from design to design.

An FSM is a quintuple $M=(Q, \Sigma, \Delta, \sigma, q0)$ where

Q	the set of symbols denoting states
Σ	the set of symbols denoting inputs
Δ	the set of symbols denoting outputs
$\sigma: Q\times B_\Sigma \rightarrow Q\times B_\Delta$	the transition function
q0	$q0 \in Q$, the initial state

In our approach, the protocol specification is represented in a specification FSM, or spec FSM. It specifies whether the output response of a specific implementation (DUV) is legal or not under a given input sequence. In other words, the spec FSM actually acts as a functional monitor of the DUV. The possible DUV behaviors are:

1. don't-care: The behavior is not defined since the input sequence is not supposed to appear.
2. legal: The output sequence is allowed by the protocol under a valid input sequence.
3. illegal: The output sequence is prohibited by the protocol under a valid input sequence.

Hence, in every spec FSM, two special states are defined: vio and dc. The spec FSM moves to the state dc if a don't-care input sequence is applied to the DUV. If the DUV behaves illegally under a valid input sequence, the spec FSM moves to the state vio. If the DUV behaves legally under a valid input sequence, the spec FSM moves among other normal states excluding dc and vio. Accordingly, the spec FSM is an FSM $M=(Q, \Sigma, \Delta, \sigma, q0)$ whose behavior is a monitor of certain I/F logic where Q contains all normal states along with two extra special states vio and dc, $\Sigma= I_{ctrl} \cup O_{ctrl}$, and $\Delta = \phi$. Note that, unlike typical functional monitor designs, the output set Δ of a spec FSM is empty since there is no need for extra outputs to indicate whether

the DUV behavior is legal, illegal, or don't-care.

The spec FSM of a simplified AMBA AHB slave protocol is given in Figure 2 as an example. In the state idle/busy, if HREADY is not asserted or HRESP is not set to OKAY, the spec FSM moves to the state vio (e9). This implies that a slave cannot respond anything but OKAY to an IDLE or BUSY transfer, which is explicitly defined in the AMBA specification. In addition, in the state orig, if a transfer is initiated by asserting HSEL and HREADYin as well as setting HTRANS to SEQ, the spec FSM moves to the state dc (e3). This infers that the master should never set HTRANS to SEQ for the first transfer, which is an input constraint to the slave. These inputs can be treated as don't-cares since they are not supposed to appear.

To translate an I/F protocol from a document into a spec FSM is relatively systematic than into rule-based properties (in CTL or emerging property languages). While building the spec FSM, all possible combinations of I_{ctrl} and O_{ctrl} are considered for each normal state, which means all possible transitions of each normal state are fully specified. For property-based methods, however, it is really hard to determine whether all properties are completely identified or not.

By introducing the spec FSM, a DUV is compliant with the specification if and only if there exists no valid input sequence (of any arbitrary length) along with the corresponding DUV output sequence that can drive the spec FSM into its state vio. In this project, we propose a practical approach for I/F compliance verification which is developed by modifying the monitor-based methodology. Figure 3 shows the block diagram of our approach. It consists of four parts: a DUV, a bus master, a FSM model, and a bus monitor. In this case, the DUV is the bus slave which we want to verify. The bus master is used to initiate the transfers to the DUV. The FSM model is used to describe the specification of an interface protocol. The bus monitor is a checker that catches the values of the bus signals between the bus master and the DUV, and then compares them with the FSM model to determine whether the DUV passes this test or not.

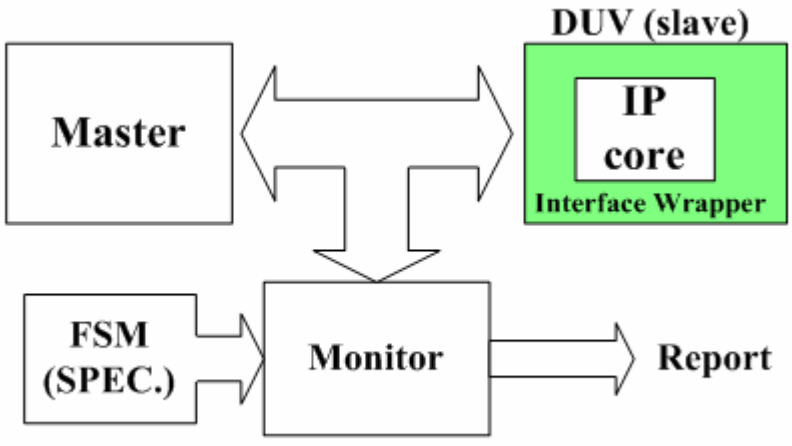

Figure 3. The block diagram of our approach

Our approach is performed in a simulation environment in which the IP component is

exercised. By controlling the bus master, we can use it to deliver the appropriate bus transfers to the DUV. As a result, we can conveniently verify the DUV's compliance to the specification of the I/F protocol. Since the aim of our approach is to verify the bus interface, we assume that the IP core in the DUV operates correctly during the simulation; that is, the original IP without connecting the bus interface must be error-free.

Now we describe our verification flow. First, we model the specification of I/F protocol as a FSM, which specifies the interactive behavior among the bus system, especially for the bus master and the bus slave. Then, we simulate the whole system, including the bus master, the bus monitor, and the DUV. According to the observation of the monitor, we obtain the actual interactive behavior. Next, we compare the interaction with the FSM. If any conflict occurs, the violations will be recorded in the error report. Otherwise, a coverage analysis of this simulation will be executed and then the coverage percentage will be shown in the coverage report.

Table 1 Design information

Design	Protocol	# primary inputs	# primary outputs	Supporting responses
AC97 controller	WISHBONE	11	9	Normal (wait)
SPI	WISHBONE	9	7	Normal, Error
PWM/Timer/Counter	WISHBONE	10	6	Normal, Error
RGB2YCrCb	AMBA AHB	8	3	Normal
Convolution	AMBA AHB	8	3	Normal (wait)
MAC	AMBA AHB	8	3	Normal, Error

Table 2 Experimental results for the erroneous designs

Design	Inserted error	Result
PWM/Timer/Counter (WISHBONE)	**ACK** and **ERR** are asserted simultaneously	Test fails in the Normal state: ERR!=0
MAC (AMBA AHB)	The ERROR response is given with a single cycle	Test fails in the ERROR state: ERROR response needs two cycles
Convolution (AMBA AHB)	Provide a wait state to IDLE transfer	Test fails in the "Normal with IDLE/BUSY" state: HREADY!=1 Test fails in the "WAIT from Master" state: HREADY!=1

In this project, we implant a compiler which can translate the spec FSM into a synthesizable protocol checker in Verilog. We also perform some experiments to support our approach. These experiments are conducted over six real designs. The circuit information about these designs is given in Table 1. The first column is the name of design. The second column indicates the

protocol type of the design. The numbers of primary inputs and outputs are shown in the third and fourth columns respectively. The last column indicates the supporting responses of the design. The design AC97 controller is a simple AC97 controller IP core that supports one AC97 codec with 6 output and 3 input channels. The design SPI is the serial peripheral interface IP core. The design PWM/Timer/Counter is a multi-function IP core, which provides the user-programmable PWM, timer, and counter controller. The design RGB2YCrCb is a RGB-to-YCrCb translator. The design Convolution is a convolution calculator for discrete wavelet transfer. The design MAC is a multiplier accumulator.

The experimental results demonstrate that all DUVs can be verified effectively and achieve 100% quickly. Besides, we also detect an error of the design SPI, which is claimed compliance to WISHBONE specification by other compliance verification. After being modified, this design is correct and can achieve 100% coverage in 19 clock cycles.

Furthermore, we also try to inject some errors into the correct designs and then conduct some experiments for these erroneous designs. The results are shown in Table 2. We inject some errors into three correct designs, PWM/Timer/Counter, MAC, and Convolution. The errors that we inject to the designs are indicated in the column “inserted error”. The column “result” shows the error report of the compliance verification. The experimental results for the erroneous designs demonstrate that our approach can indeed detect the errors and indicate the states in which the errors occur. This can help designers to shorten the debugging process.

四. 結論

In the first year, we propose a monitor-based approach for interface compliant verification, in which a bus monitor is used to observe the bus signals among the bus system and to check if they conform to the interface protocol specification. This methodology can help designers to verify the interactive behavior of an Interface protocol more efficiently. In order to systematically extract the properties from an informal Interface specification, we propose a spec FSM model. Based on this model, we can extract the necessary properties and avoid obtaining the redundant properties.

五. 參考文獻

[1] Kurt Keutzer, Sharad Malik, A. Richard Newton, Jan M. Rabaey, and A. Sangiovanni-Vincentelli, ”System Level Design: Orthogonalization of Concerns and Platform-Based Design,” in IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, vol. 19, no. 12, Dec. 2000, pp. 1523-1543.

[2] VSI Alliance, Virtual Component Interface (VCI) Standard - OCB 2 1.0, http://www.vsia.org, Mar. 2000.

[3] Kanna Shimuzu, David L. Dill, and Alan J. Hu, “Monitor-Based Formal Specification of PCI,” in Proceedings of the 3th International Conference on Formal Methods in Computer-Aided Design, Nov. 2000, pp. 335-353.

[4] Marcio T. Oliviera and Alan J. Hu, “High-Level Specification and Automatic Generation of IP

Interface Monitors," in Proceedings of the 39th Design Automation Conference, June 2002, pp. 129-134.
[5] Alan J. Hu, Jeremy Casus, and Jin Yang, "Efficient Generation of Monitor Circuits for GSTE Assertion Graphs," in Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design, Nov. 2003, pp. 154-159.
[6] Jun Yuan, Kurt Shultz, Carl Pixley, Hillel Miller, and Adnan Aziz, "Modeling Design Constraints and Biasing in Simulation Using BDDs," in Proceedings of the 1999 IEEE/ACM International Conference on Computer-Aided Design, Nov. 1999, pp. 584-589.
[7] Kanna Shimizu and David L. Dill, "Deriving a Simulation Input Generator and a Coverage Metric From a Formal Specification," in Proceedings of the 39th Design Automation Conference, June 2002, pp. 801-806.
[8] Andy Nightingale and John Goodenough, "Testing for AMBATM Compliance," in Proceedings of the 14th Annual IEEE International ASIC/SOC Conference, Sept. 2001, pp. 301-305.
[9] ARM Limited, AMBA Specification (Rev 2.0), 13 May 1999.
[10] K. L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.
[11] Pankaj Chauhan, Edmund M. Clarke, Yuan Lu and Dong Wang, "Verifying IP-Core Based System-On-Chip Designs," in Proceedings of the 12th Annual IEEE International ASIC/SOC Conference, Sept. 1999, pp. 27-31.
[12] Ilan Beer, Shoham Ben-David, Cindy Eisner, Yechiel Engel, Raanan Gewitzman and Avner Landver, "Establishing PCI Compliance Using Formal Verification: A Case Study," in Proceedings of the 14th International Phoenix Conference on Computation and Communications, Mar. 1995, pp. 373-377.
[13] Abhik Roychoudhury, Tulika Mitra, and S.R. Karri, "Using Formal Techniques to Debug the AMBA System-on-Chip Bus Protocol," in Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 2003, pp. 828-833.
[14] http://www.eda.org/vfv/docs/psl_lrm-1.01.pdf/.
[15] http://www.verificationlib.org/.
[16] http://www.opervera.org/.
[17] http://www.systemverilog.org/.
[18] Annette Bunker and Ganesh Gopalakrishnan, "Using Live Sequence Charts for Hardware Protocol Specification and Compliance Verification," in Proceedings of the IEEE International High Level Design Validation and Test Workshop, Nov. 2001, pp. 95-100.
[19] Annette Bunker, Ganesh Gopalakrishnan, and Sally A. McKee, "Formal Hardware Specification Languages for Protocol Compliance Verification," ACM Transactions on Design Automation of Electronic Systems, vol. 9, no. 1, Jan. 2004.

六. 計劃成果自評

In the first year of this project, we first study some popular interface protocols for SOC designs and previous specification methods. Then, we develop a FSM-based specification style to model

interface protocol systematically. Furthermore, we also implement a compiler to produce protocol checkers from the spec FSM automatically. This protocol checker can be used in the proposed monitor-based simulation environment.

Besides, we also publish 2 international conference papers and submit 1 journal paper about interface protocol verification:

[1] **Che-Hua Shih, Jinn-Dar Huang, and Jing-Yang Jou, "Stimulus Generation for Interface Protocol Verification Using the Non-Deterministic Extended Finite State Machine Model", IEEE International High Level Design Validation and Test Workshop, November 2005.**

[2] **Man-Yun Su, Che-Hua Shih, Jinn-Dar Huang, and Jing-Yang Jou, "FSM-Based Transaction-Level Functional Coverage", Asia and South Pacific Design Automation Conference 2006, January 2006.**

[3] Ya-Ching Yang, Che-Hua Shih, Chia-Chih Yen, Juinn-Dar Huang, and Jing-Yang Jou, "FSM-Based Formal Compliance Verification of Interface Protocols", IEEE Transaction on Very Large Scale Integration Systems (Submitted).

These research and publication results indeed fit our expect goal in the first year.