

System-On-Chip Validation using UML and CWL

Qiang Zhu Ryosuke Oishi
Fujitsu Laboratories LTD.
1-1, Kamikodanaka 4-chome,
Nakahara-ku,
Kawasaki 211-8588, Japan
{shiyu, roishi}@labs.fujitsu.com

Takashi Hasegawa
Fujitsu Limited
1-1, Kamikodanaka 4-chome,
Nakahara-ku,
Kawasaki 211-8588, Japan
thasegaw@jp.fujitsu.com

Tsuneo Nakata
Fujitsu Laboratories LTD.
1-1, Kamikodanaka 4-chome,
Nakahara-ku,
Kawasaki 211-8588, Japan
nakata@labs.fujitsu.com

ABSTRACT

In this paper, a novel method for high-level specification and validation of SoC designs using UML is proposed. UML is introduced as a formal model of specification for SoC design. The consistency and completeness of the specification is validated based on the formal UML model. The implementation is validated by a systematic derivation of test scenarios and specification based coverage metrics from the UML model. The method has been applied to the design of a new media-processing chip for mobile devices. The application of the method shows that it is not only effective for finding logical errors in the implementation, but also eliminates errors due to inconsistency and incompleteness of the specification.

Categories and Subject Descriptors

C.0 [Computer Systems Organization]: General – *System specification methodology*; I.6.4. [Simulation and Modeling]: Model Validation and Analysis; J.6 [Computer-Aided Engineering]: Computer aided design (CAD)

General Terms: Verification, Design.

Keywords: Verification Process, UML, Validation and Verification, Specification Modeling

1. INTRODUCTION

System-On-Chip (SOC) validation has become increasingly difficult with increasing complexity. Current SoC validation techniques focus primarily on implementation verification. They are unable to provide specification validation due to the absence of a methodology that includes development and analysis of a formal model of the SoC specification. The requirements specifications for the SoC are often written in informal natural language, which invariably leads to ambiguities in interpretation of the specification and, therefore, validation is infeasible.

Unified Modeling Language (UML) [1] has been increasingly used to capture requirements specification in software engineering. Use case analysis technique [2] has been proposed to clarify the requirements with the concept of use case, actor and

event flows. An object-oriented design process for SoCs is proposed using extensibility mechanism of UML in order to represent the parallel, architecture and timing using UML diagrams [3].

In UML-based software development, interfaces are modeled by an executable operation or an object-oriented method. However, interfaces in SoCs cannot be simply modeled by operations and methods. The SoC interface protocol at a signal communication level has to be elucidated. Component Wrapper Languages (CWL) [4] is a formal interface specification language proposed to model the specification of signal changes at input/output ports for a formal and compact representation of Intellectual Property (IP) cores [5]. SystemC is a design language for SoC from behavior level to Register Transfer Level (RTL) [6].

In this paper, we propose a novel verification process for high-level specification and validation of SoC designs using UML and CWL. The proposed methodology, as shown in Figure 1, is:

- Use UML for modeling the component specification and use CWL for modeling the interface protocol specification.
- Introduce an explicit validation step for the UML-modeled components: validating the completeness and consistency of UML models in order to improve the quality of specification by eliminating errors from due to incompleteness and inconsistency.
- Derive test scenarios and coverage metrics from the UML and CWL model for implementation verification through black box testing.

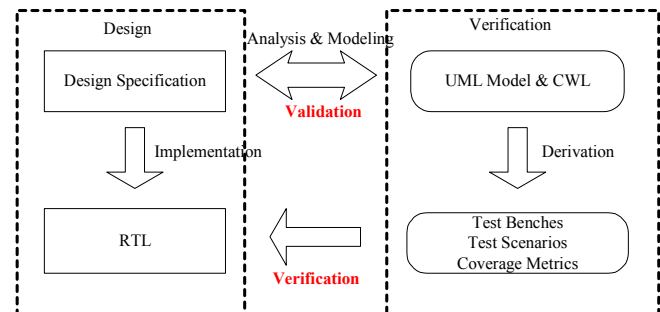


Figure 1. Specification Based Verification Process

We have applied our proposed process to a new media-processing chip for mobile devices [9] for validating its specification and implementation. The application results show our methodology is not only effective for discovering logical errors in the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CODES+ISSS'04, September 8–10, 2004, Stockholm, Sweden.

Copyright 2004 ACM 1-58113-937-3/04/0009...\$5.00.

implementation, but also is able to eliminate errors due to inconsistency and incompleteness of the specification.

2. THE MODELING OF SPECIFICATION

2.1 Modeling Components

The UML modeling flow for specification starts with use case analysis shown in Figure 2.

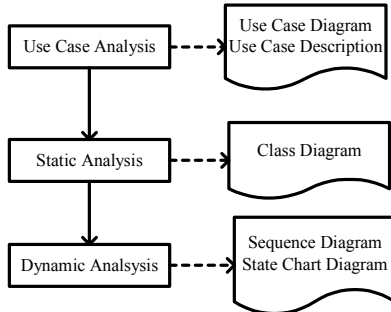


Figure 2. UML modeling flow of specification

In the **use case analysis** phase, functional requirements from customers are listed. We extract use cases and actors based functional requirements and the structure of the target system. The use case diagram of UML is employed for representing the analysis result. Figure 3 shows an example of use case diagrams. Actors can be captured from external components that use the target system. Use cases represent a set of functional services that are provided to actors by the target system. Associations are used for describing relationships among actors and use cases. Use case analysis can help us to clarify functions, users, and their relationships of the target system.

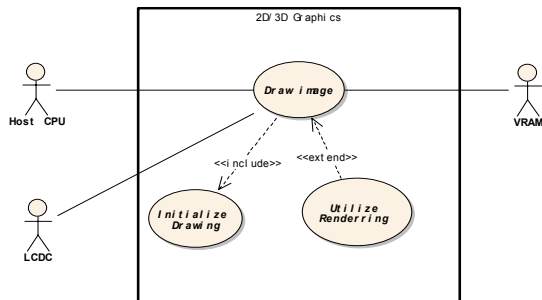


Figure 3. An example of use case diagrams

For clarifying the detail of each use case, use case description, which is written in natural language, describes event flows among actors and the target system. The event flow for a use case includes pre-condition, post-condition, basic, alternative, exceptional paths; the pre-condition represents the condition, which must be true before a use case is performed. The post-condition is a goal condition after a use case is performed. The basic path is the most typical interaction reaching post-condition among actors and system. Alternative paths indicate the other possible interactions that are able to reach the post-condition, but different from basic path. The exceptional path indicates abnormal interactions that cannot reach the post-condition and cause errors or abnormal actions. Through this process, we can capture the interactions among external components and the target system. We can also clarify the results after use cases performing based on specification.

The **static analysis** is used for extracting data objects and control objects from the specification. Data objects can be captured from input/output data. Control objects can be derived from components of the target system. We use UML class diagrams to model data and control objects in the specification. The class diagram describes the classes and their relationships using associations defined in UML. Figure 4 shows an example of class diagrams for the structure of a system. The stereotype `<<SoCModule>>` depicts the instance of such class is a component of the target system. Class analysis and class diagram can help us to clarify the static structure from specification include data structure and physical structure.

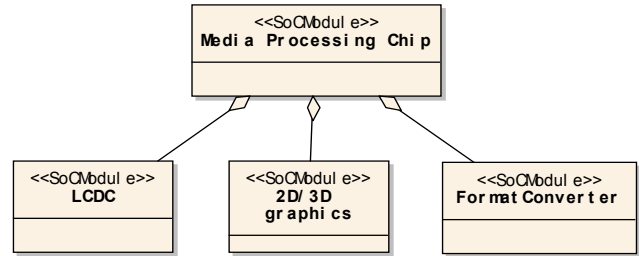


Figure 4. An example of class diagrams for system structure

The final step is the **dynamic analysis**, which models the behaviors of a system using sequence diagrams and state chart diagrams. Each event flow of use case written in informal natural language is formalized with the sequence diagram through clarifying operations (events and parameters) among the target system and actors. Figure 5 shows an example of sequence diagram. After modeling all event flows of the system with sequence diagrams, a system-level state chart diagram can be captured by analysis of sequence diagrams. Figure 6 shows an example of system level state chart diagram. Sequence diagrams and state chart diagrams can help us to formalize event flows of use cases and capture the system-level behavior of the target system.

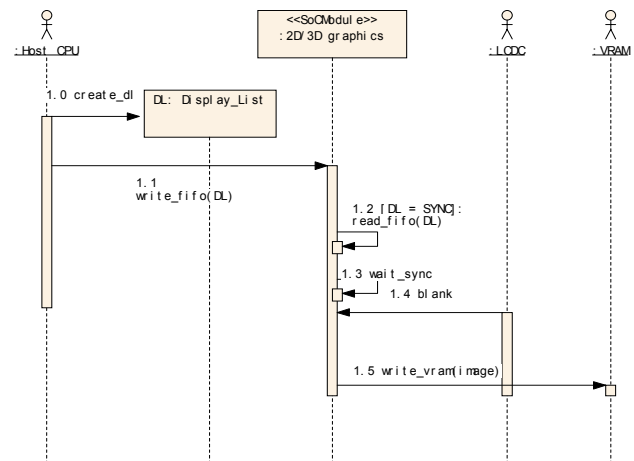


Figure 5. An example of sequence diagrams

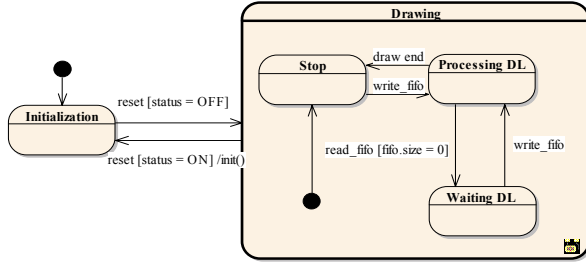


Figure 6. An example of state chart diagrams

2.2 Modeling Interface protocols

The interface of software can usually be modeled using an executable operation or an object-oriented method. In SoC designs, we need a specification description for modeling the signal communication among the input and output ports. Unfortunately, current UML does not have a proper diagram to model the communication protocol at the signal level. Component Wrapper Language [5] has been proposed to model the interface protocol of IP cores compactly and formally. Figure 8 shows an example of CWL description for a full handshake interface protocol. In our approach, we adopt CWL as a modeling language to represent the interface communication protocol among components instead of UML.

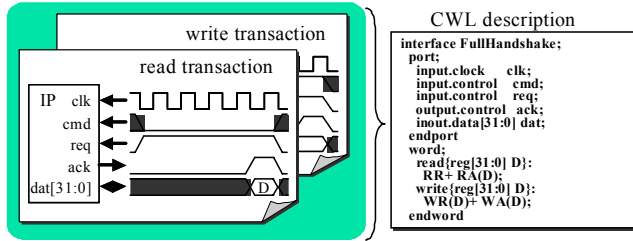


Figure 7. An example of CWL description

3. SPECIFICATION VALIDATION

We use UML to model the component specification, and CWL to model the interface protocol. In this section, we describe how to utilize UML models to validate the correctness of specification. Specification validation can be performed by checking the completeness and consistency of UML model. Through this process, we can eliminate design errors from specification before validating the implementation. Here, we describe the basic notions of completeness and consistency of specification and show the method to discover incompleteness and inconsistency from UML model.

The completeness of specification includes:

- **The completeness of use case** indicates that use cases must cover all functions in specification and each function in specification must be used by a use case once, at least.
- **The completeness of event flow** indicates that the listed paths of event flow include all possible scenarios that can occur in a use case.

The consistency of specification includes:

- **The consistency between a use case and its event flows** indicates all normal paths (basic path, alternative path) of

event flows in a use case can reach the given post-condition under the pre-condition defined in a use case description.

- **The consistency among UML diagrams** indicates there are no violations among UML diagrams.

The completeness of use cases can be confirmed by validating the functional coverage of use cases. For example, Figure 8 depicts a simple method for checking the functional coverage of a use case. First, we list all functions that we are able to find in the specification. If the use case utilizes any functions, we use associations notated by stereotype << use >>, to link the use case and its corresponding functions secondly. If we find any functions do not have associations from use cases, we can say the derived use cases are not complete because they cannot cover all of functions in the specification. In such case, we must reconfirm the use case diagram and the specification, and then add a new use case to realize the unlinked functions or add new associations to the use case that uses such functions.

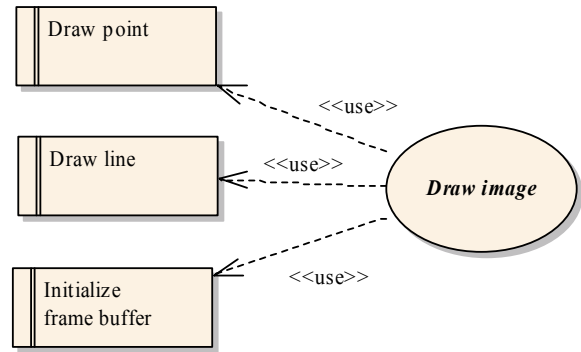


Figure 8. The link among functions and use cases

Another type of incompleteness can happen in event flow for a use case due to incomplete paths of event flow. Normally, the specification can give some examples of scenarios to show how to use functions in the target system. Such scenarios can help us to find event flow paths of use case. However, such examples do not always show all possible paths of the use case, namely incompletely specified specification. The incompletely specified event flow reveals the facts that the specification and the implementation can be incomplete. Finding such neglected paths is not only helpful for specification validation, but is also useful to generate test scenarios for implementation verification.

For finding incompleteness of event flows, we extract the system-level state chart diagram according to the sequence diagrams for all paths of event flows firstly. After that, we use the concept of incompletely specified state machine [8] for validating whether the event flow paths are complete or not. The simple method is to use STT (State Transition Table) for checking whether the state chart diagram is completely specified or not. Figure 9 shows the incompletely specified state chart and its STT. The rows of the table depict states and the columns indicate events. Each cell of the table indicates the next state transition. In the incompletely specified state chart diagram, the cell marked with “??” shows the undefined transition based on sequence diagrams. In such cases, we do not know what will happen when the event is inputted in that state. After confirming and improving the specification, we can modify the STT to a completely specified state machine shown in Figure 10. Note that “invalid” means that such

combination of event and is impossible to happen based on the specification. After finishing the completely specified state machine, we validate that all paths of event flow cover the all transitions in state chart diagram. If event flow paths do not cover any transitions, we should add a new path to the event flow to make it complete.

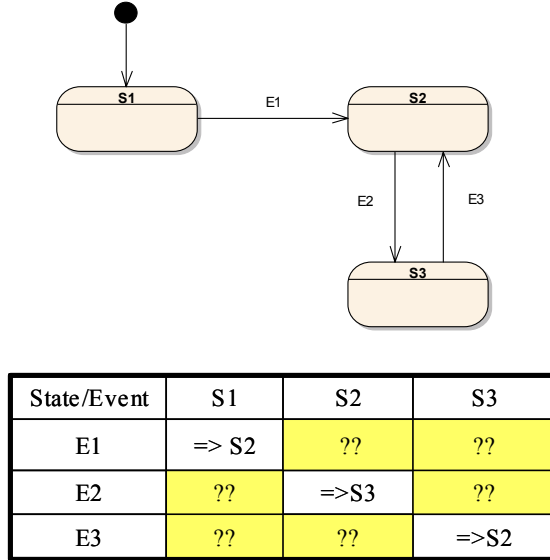


Figure 9. Incompletely specified specification

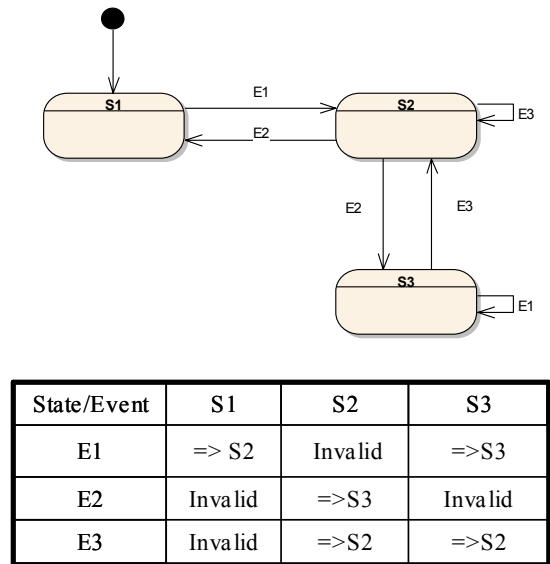


Figure 10. Completely specified specification

The consistency between a use case and its event flow can be discovered by validating if all paths of the event flow can reach the post-condition under the given pre-condition. For validating such properties, we use both sequence diagrams and state chart diagrams. Sequence diagrams give us the order of events, and state chart diagrams indicate the state transitions when such events occur. The pre-condition and post-condition in the use case description can be linked to the states in state chart diagram. We

can use the reachability analysis techniques [8] to confirm that the state, which linked to post-condition, can be reached from the state that linked to pre-condition for all sequence diagrams except for exceptional paths. If any sequence diagrams of a use case cannot reach the post-condition linked state, we should reconfirm the specification or sequence diagrams whether they are correct or not.

The consistency among UML diagrams can be validated by checking whether there are any violations in different UML diagrams or in different objects in a UML diagram. Especially, the relationship among UML objects can lead to some conflicts during analysis and modeling process. Most of them can be modeling mistakes due to misunderstanding, analyzers' careless miss. However, some of them can be specification faults.

For example, the class diagram shown in Figure 4 depicts the component "2D/3D graphics" has no associations with the component "LCDC" based on the architecture block diagram in the specification firstly. However, after finishing the use case analysis, we find the component "LCDC" sends a "blank" signal to "2D/3D Graphics" in the sequence diagram that describes the alternative path of event flow for use case "Drawing image" shown in Figure 5. In this case, there are violations between the sequence diagram and the class diagram. We confirm such inconsistency to designers and find a mistake in the architecture block diagram in the specification. Accordingly, we add a new association between the class "2D/3D graphics" and the class "LCDC" in the class diagram and correct the architecture block diagram to append a new connection between them.

Through validating the completeness and consistency of UML model, we can obtain a correct, well-defined formal model for specification. The errors due to incompleteness and consistency in a specification can be eliminated before validating the implementation. This can help us to improve the quality of specification and reduce the cost of implementation verification.

4. IMPLEMENTATION VERIFICATION

In traditional verification process, verification engineers have to derive the test scenarios from an informal specification document. Usually such task needs a huge effort and long time. Furthermore, performing this task needs an experienced verification engineer because there are no metrics to measure the quality of test scenarios that derived from an informal specification. The quality only depends on the experience in traditional method.

In our approach, we find UML model for specification is not only able to improve the quality of specification mentioned in section 3, but also can be effectively used for implementation verification. In this section, we describe how to derive test scenarios from UML models to systematically validate implementation and introduce specification based coverage metrics to measure the quality of derived test scenarios.

4.1 Extracting Test Scenarios from UML Models

The basic idea for extraction of test scenarios from UML models is to utilize diagrams, objects, relations and descriptions in UML models. First, because use cases cover all functions of the target system, test scenarios can be derived from each use case. The pre-condition indicates the condition before executing the normal test scenarios. The post-condition shows the expected values after completing the normal test scenarios. The derived test scenarios

should cover all sequence diagrams of use cases. Class diagrams can help us to determine the detailed parameter values in each test scenario. The state chart diagram can be used for generating all possible paths of event flow that do not appear in sequence diagrams.

In addition, UML model can help us to derive test scenarios to validate the functionality of the target system, but such test scenarios do not cover the interface communication protocol at the signal level. We use CWL description to check the correctness of interface protocol by generating a protocol checker from the CWL2HDL [9] tool.

4.2 Specification Based Coverage Metrics

The specification based coverage metrics include **functional coverage** and **transaction coverage**. Functional coverage is a metric to measure how much functionality of the target system is covered by test scenarios. Functional coverage can be obtained from how many objects in an UML model are covered by test scenarios. If the coverage is not 100%, that means the test scenarios are not sufficient to test all functionality of the target system. Transaction coverage is a metric to measure how many combinations of interface protocol are covered by test scenarios. The transaction coverage can be obtained from the transition coverage of the state machine, which is generated from CWL. The functional coverage and transaction coverage can help us to quantify the quality of the derived test scenarios to define how much functionality and protocols have been verified by test scenarios.

5. APPLICATION RESULTS

We have applied our verification process to a media-processing chip for mobile devices in Fujitsu [7]. We organized a verification team separated from the design team to validate the specification written in natural language by designers and validated implementation with black box test. The UML and CWL were used as modeling languages to analyze and formalize the component specification and interface protocol specification respectively. The specification validation was performed by validating the completeness and consistency of the UML model. The specification was brushed up by eliminating those errors due to incompleteness, inconsistency in specification at analysis and modeling phase using UML. Then we derived test scenarios from the UML model and generated interface protocol monitors to

verify the functionality and interface protocols. We applied functional coverage and transaction coverage to measure the quality of the derived test scenarios. Furthermore, test benches were also generated from UML model and implemented using SystemC. This let us transform the results of analysis and modeling into test scenarios, and test benches systematically.

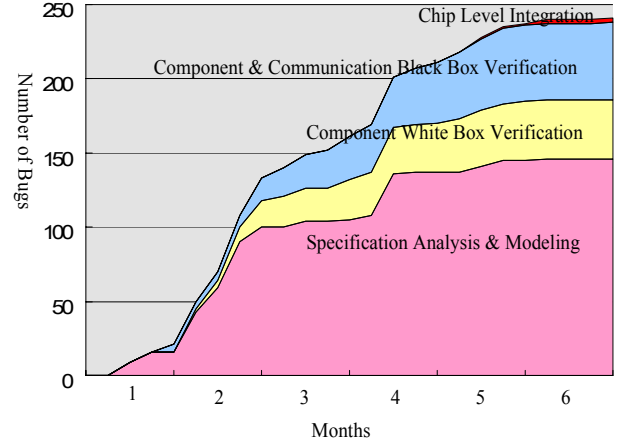


Figure 11. The results for errors and their discovery time

We took six months and 13 verification engineers to validate the specification and perform black box test for components and chip-level integration. Note that there was only one experienced verification engineer in verification team. Designers were responsible for white box test of each component and guaranteed code coverage for each component to reach 100%. The verification team assured functional coverage and transaction coverage for each component reached 100% in black box test. The chip released in about 1.5 months after tape out. So far, we have not found any critical errors after the first chip release.

Figure 11 shows the results for errors and their discovery time. We found 132 errors that include oversights, mistakes, and ambiguities due to incompleteness and inconsistency in the specification at analysis and modeling phases. We also discovered 14 specification errors in the implementation test phase. After that, we found 51 errors from the implementation with black box test using the test scenarios and test benches derived from the UML model. Meanwhile, designers found 40 errors at white box test

Table 2. Errors of specification

The type of Components	#Pages	#Use Cases	#Operations	#Paths of Event Flow	#Incomplete Errors	#Inconsistent Errors
Image Processing	191	39	96	92	6	21
	21	2	49	10	6	4
	51	1	61	12	0	4
	28	1	70	12	2	3
	17	1	36	10	2	0
Controller	74	3	263	9	0	16
	80	9	61	39	4	2
	21	12	6	23	4	17
	44	62	16	26	1	11
	13	5	6	17	0	6
	22	1	6	57	0	9

phase. At chip-level test phase, we only found three errors. From these results, we can say that most of errors can be found at early stage of design, especially in analysis and modeling phases using our proposed verification process.

Table 2 shows the number of errors for each component we found in the specification in analysis and modeling phases. The components named “Image Processing” indicate image-processing components that include 2D/3D graphics, MPEG codec, JPEG codec etc. The “Controller” components are communication or control components such as “Arbiter”, “Bridge”, “Host Interface” and so on. The “#Pages” shows the number of pages for each component. The “#Use Cases” indicates the number of use cases we derived from the specification. The “#Operations” depicts the number of system operations that appear in event flows of use cases. The “#Paths of Event Flow” shows the number of event flows includes basic, alternative, exceptional paths. The number of pages and the number of UML objects show the functional complexity of each component. The “#Incomplete Errors” shows the number of errors in specification due to incompleteness of use cases and event flows. The “#Inconsistent Errors” represents the number of errors in specification due to inconsistency between event flows and use cases and inconsistency among UML diagrams. The result of the Table 2 shows our proposed process can effectively help us to improve the quality of specification before implementation verification.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a verification process for SoC based on specification using UML and CWL. UML is used for modeling the component specification, and CWL is utilized for describing the interface protocols. We can improve the quality of the specification by validating the completeness and consistency in UML models. On the other hand, we can also derive test scenarios, test benches and coverage metrics from UML and CWL model systematically. We have succeeded in applying our proposed verification process to the media-processing chip for mobile devices in Fujitsu without respin. The results showed our process is not only useful for specification validation, but also helpful for implementation verification.

However, we suffered from the following issues in our practice.

- Most of the tasks for validation and verification are performed manually because we did not have a good tool to help us run our process automatically. This made us take much time to validate the UML model and derive test scenarios from the UML model.
- So far, we have not found an exact formal expression to model the specification UML model. There were some informal notations, expressions in UML model, especially

for timing constraints. We need some modeling techniques to improve such issues [10] [11] in future.

In future work, we will try to find the solutions to solve the problems mentioned above. On the other hand, we are applying our verification process to real SoC designs continuously to find the best practices from our proposed process.

7. ACKNOWLEDGMENTS

We would like to appreciate the MMPs design team in Fujitsu for their perfect collaboration. We are grateful to the reviewers for their constructive suggestions. We also would like to thank Dr. Praveen K. Murthy, Dr. Sreeranga P. Rajan in Fujitsu Laboratories of America (FLA) and Dr. Rafael K. Morizawa in Fujitsu Laboratories for their valuable discussion and review.

8. REFERENCE

- [1] OMG home page, <http://www.omg.org/>
- [2] I. Jacobson, *Object-Oriented Software Engineering A Use Case Driven Approach*, Addison-Wesley Toppan, 1995.
- [3] Q. Zhu, A. Matsuda, S. Kuwamura, T. Nakata, and M. Shoji, "An object-oriented design process for System-on-Chip using UML", Proc. of the 15 th Int. Symposium on System Synthesis (ISSS 2002), 1-4 October, Kyoto, Japan, pp. 249-254.
- [4] K. Suzuki, K. Ara, and K. Yano, "An interface description language for comprehensive IP specification", IP2000 Europe, pp. 195-205, Oct. 2000.
- [5] Component Wrapper Language, <http://www.labs.fujitsu.com/en/techinfo/cwl/index.htm>
- [6] SystemC OSCI, <http://www.systemc.org>
- [7] Press Release of Fujitsu Microelectronics America Inc., <http://www.fma.fujitsu.com/newsArt.asp?code=033004b>
- [8] G. D. Hachtel, F. Somenzi, *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, 1996.
- [9] CWL2HDL Home Page, http://www.labs.fujitsu.com/en/techinfo/cwl/download_tools.htm
- [10] S. Tasiran, Y. Yu, B. Batson, "Using a Formal Specification and a Model Checker to Monitor and Direct Simulation", In Proc. 2003 Design Automation Conference, 40th DAC, pp. 356-361, 2003.
- [11] G. Graw, P. Herrmann, and H. Krumm, "Verification of UML-based real time system designs by means of cTLA", Proc. of 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC2K), pp 86-95, 2000.