System Level Design Technologies and System Level Design Languages
- Activities Report -

This paper summarizes the technical trends of system level design based on the results of an investigation made by the SLD Study Group during the fiscal year ended in March 2001.

Introduction of System Level Design Study Group (SLD-SG) of JEITA

The SLD-SG was established under EDA-TC of JEITA (previously EIAJ) in Nov 1998. Currently, it consists of 21 members including not only semi-conductor companies and electrical companies but also EDA vendors and three advisory members from universities. Our objectives are to investigate the trends of system level design technologies from the point of “needs” and “seeds” and to give our feedback to related organizations. In this paper, we summarize the technical trends and propose a desirable system level design flow based on the results of our investigation.

Designers’ Needs and Solutions

In order to extract actual requirements of system designers, the SLD-SG made inquiries of 163 designers in 14 companies on Oct. 1999. As a result, the following problems are extracted according to the design phases:

(1) Functional design
•Handling ambiguity of specification
•Changes of system specification
•Slow simulation speed of system verification

(2) Architecture design
•Lack of EDA tools for architecture determination
•Insufficient estimation accuracy in architecture design

(3) Implementation
•Lack of effective design tools that support from high level to RTL
•Mismatch between specification and implementation

(4) Others (design database and co-simulation)
•Difficulty of IP customization
•Insufficient performance of co-simulation tools for SW design

These problems should be solved in the system design flow. The SLD-SG tried to solve them by introducing the following key strategies:
-A methodology to define functional specification and an executable system level design language
-Fast exploration method for finding optimum architecture
-Good estimation models and utilization of design database
-Interface synthesis between HW/SW and their equivalency check
-Standardized IP interface, IP synthesis technique, highly abstracted HW model

Technology Trends as "Seeds"

The SLD-SG investigates the trend of EDA technologies for system level design. According to the "EDA Technology Roadmap Toward 2002" by EDA Vision-SG (Slide 5), almost all the technologies of system level design are in the trial phase in 2001.

The following systems are typical examples of system level design tools developed by universities or vendors.

POLIS, which is developed by UCB, is a co-design environment for embedded systems of CPU/DSP. It supports functional design through prototyping. It performs architecture exploration by mapping behavior to architecture. Accurate and high-level performance estimation is available using a fuzzy instruction set.

OCAPI-xl is developed by IMEC. It uses a C++ model that can describe hardware and software uniformly. It performs optimal block partitioning using performance analysis and automatically generates VHDL/Verilog-HDL/C.

IpChinook is a system based on IP-based/reused-design methodology. It was developed by University of Washington. It can handle Java, C, and Pia, and generates control circuits and interface between functional blocks.

CoCentric is developed by Synopsys Inc. It supports hardware side and co-verification of SoC design. It is a hardware synthesis system from behavioral description in SystemC/VHDL/Verilog-HDL. It also supports conversion from floating-point data to fixed-point data.

N2C is developed by CoWare Inc. It supports the design flow from functional level to implementation. In N2C, the system is defined in Untimed/BCASH/BCA abstraction levels by original C-based language CoWareC. It performs interface synthesis between hardware and software by communication refinement. They plan to support SystemC in the future.

VCC is developed by Cadence Inc. based on POLIS technologies. It can perform architecture exploration by mapping function to architecture, performance estimation using a fuzzy instruction set, and communication synthesis. It supports IP reuse and offers the design phases through implementation.

The result of our investigation is summarized in slide 18. According to the investigation, function verification, architecture mapping, performance estimation, interface synthesis, and HW/SW co-verification may be solved in the near future. These technologies are applied to a restricted area under some conditions. On the other hand, function partitioning, high-speed estimation, estimation of cost and power, and definition of system specification and constraints are still immature. Only one system supports the function partitioning by verifying the partitioned system model. The estimation should be improved and the description of specification and constraints need to be standardized. There remain a lot of problems in the system level design. We need to improve the current technologies and try to establish new technologies for unsolved problems.

Desirable System Level Design Flow

Based on the investigation above, the SLD-SG proposed a desirable system level design flow. The current design flow is shown in slide 10 and the proposed design flow is shown in slide 12.

In the current design flow, there is a big gap between the system specification phase and the implementation phase. This gap causes various problems such as misunderstanding and re-spins. In order to fill the gap, we replaced it with function determination and architecture determination.

In the former phase, we define the system function with a system level design language and verify it by simulation. The function of the system is clearly determined in this phase. In the latter phase, we determine the architecture that realizes the system function determined by the previous phase. This phase can also be divided into two phases, that is, design space generation and design space exploration. In the former phase, "partitioned functional definition" and "candidates of architectures" are generated. This is done based on the information from "profiling" described later.

In the latter phase, each partitioned functional definition is mapped into an architecture selected from the candidates, and the performance is estimated by a method such as a simulation. The architecture selection and mapping are repeated and the optimal architecture that satisfies the constraints is found. After deciding the architecture, it is converted to an RTL description of hardware or embedded software manually or automatically by a behavioral and interface synthesis tool. Furthermore, a test specification and expected values are generated from the system specification and system level simulation results, and test benches are generated. We ensure consistency of the design by using the test bench throughout the design flow. We also try to increase the reusability of previous design by using a "design database" which is composed of IPs (Intellectual Properties) and an estimation database in every design phase.

Our design flow has two features; the design space generation using profiling techniques and the design space exploration using transaction simulation. In the design space generation phase, we produce the partitioned functional definition and the candidates of architectures. The partitioned functional definition is a system model that is represented by functional blocks and their communication. We call it the "process model". In comparison that the function model is a single program model and focused on the functional algorithm, the process model is a multiple process model focused on the function blocks and communication. In the process model, the concurrency of blocks is also introduced. Each block is to be implemented as a hardware block or a software block. The architecture model is an implementation model that represents the system by processors, memories, other hardware blocks, and their connections. We first produce multiple candidates of architectures because we cannot decide which is the best. Because some information is required to generate the process model and the architecture model, we introduce the profiling techniques for this purpose. The method is to extract the information for generating the process model and the architecture model by analyzing the function model statically or dynamically. By profiling, various information such as the number of variable accesses, the amount of data transfers and the amount of calculations may be extracted. Sometimes the statistical analysis of overall system is performed.

In the design space exploration phase, the partitioned function definition is repeatedly mapped into architectures and the best architecture is determined. In this phase, although the estimation is required, it is impossible to attain the simulation in VHDL/Verilog-HDL. Therefore, we introduce a new model called the "transaction model" which represents only the amount of operations instead of the process itself. We aim for a fast simulation by separating the performance information from functions. For example, suppose a simple transaction model in which the amount of calculations of each block can be represented by a constant value independent of its inputs/outputs. Because the capability of each block is given from the architecture model, the expected processing time is given by a simple formula. Total performance is calculated by summation of the formulae for each block. In fact, because there is a trade-off between the accuracy and the estimation speed of the transaction model, we need to develop methods to generate the information required to do that efficiently.

Trend of System Level Design Languages

The SLD-SG is watching the trend of system level design languages because the system level design languages play an important role for the system level design flow. Moreover, the standardization activities of C based system level languages (SystemC, SpecC, etc.) started in these years. Slide 20 shows the trend of standardized system level design languages. The vertical axis represents the abstract level of each language and the horizontal axis represents the year of its announcement. We selected SpecC, SystemC, and UML.

SystemC was proposed based on the technologies of Synopsys Inc., CoWare Inc., and Frontier Design Inc. on Sept. 1999. It can represent from the system level to HW/SW by adding dedicated class libraries without enhancing C/C++ grammar. Therefore, it is available on the standard C/C++ development environment. On the other hand, the readability of description is being compromised. It was developed from HW design using C/C++ adapting it to the system level and it is easy to move from the HDL environment. The standardization of SystemC is promoted by Open SystemC Initiative (OSCI) that consists of more than 70 of EDA vendors, IP vendors, and semiconductor companies (Apr. 2000).

SpecC was developed by Prof. Gajski at University of California at Irvine in 1997. It is a unified description of HW/SW with enhanced ANSI-C grammar and has high readability. However, special tools and environments are required to capture SpecC. Its methodology covers system modeling through architecture determination. The standardization of SpecC is promoted by SpecC Technology Open Consortium (STOC), and system houses and embedded software tool vendors have strong interest in its activities.

UML was developed as a modeling language for complicated large systems. It is a unification of existing object-oriented modeling methods. The standardization was started on 1996 by OMG (Object Modeling Group). It is a visual language with high modeling ability, and visual and multi-viewed system modeling can be defined by various kinds of diagrams. Moreover, it can describe constraints by using a constraint description language. However, the methodology of applying it to hardware design is incomplete. Currently, it is mainly used in business software. It has just started to be used in embedded software and is in a trial phase for hardware design.

We applied SystemC, SpecC, and UML to our proposed flow to evaluate them. Slide 24 shows the result. In the table, ◎ represents that it is one of the characteristics of the language, ○ represents that it is possible, △ represents that it is insufficient, and × represents that it is impossible. The system specification consists of requirements, not programs. In this sense, UML is the most suitable for the system specification because it can formally use natural language and can describe constraints by using Object Constraint Language (OCL). Because SpecC and SystemC do not assume such kind of definitions, some extension or link to UML is required. Both SpecC and SystemC are applicable to function determination and they have merits and demerits respectively. SpecC has an advantage in the modeling methodology and SystemC has an advantage in applicability of existing tools. Although the state chart diagram of UML is verifiable, this phase seems to be the limit of UML. The architecture determination needs languages, tools, and methodologies. In this sense, both SpecC and SystemC can be applied to the function definition of design space generation and its verification but they are insufficient for architecture generation, mapping, and estimation. For the interface to implementation, SystemC has an advantage because it supports fixed-point and RTL, while SpecC needs some extensions. Test benches can be described with SpecC and SystemC. But both languages are insufficient from the point of design flow, because they do not cover from function IP to implementation IP seamlessly.

Conclusion

In this paper, we have presented the technical trends of system level design based on the results of investigation made by the SLD-SG of JEITA. The SLD-SG has tried to consider various points of view to clarify the desirable system level design flow. First, we presented the results of designers' requirements for the system level design. Next, we investigated the technical trend of EDA tools and their technologies. Based on the needs and seeds, we have proposed a desirable system level design flow. Finally, we summarized notable system level design languages, and we have evaluated and considered each by applying them to the proposed design flow. Because the technologies for system level design are still rapidly changing now, the SLD-SG will continue to watch the trend and provide feedback to related organizations.