VCore-based Platform for SoC Design

Yoichi Onishi, Michiaki Muraoka, Makoto Utsuki, Naoyuki Tsubaki

Semiconductor Technology Academic Research Center (STARC)
Shin-Yokohama 3-chome, Kohoku-ku, Yokohama, 222-0033, Japan
TEL : 045-478-3300, FAX : 045-478-3299
{onishi, muraoka, utsuki, tsubaki}@starc.or.jp

Abstract

The reuse-based design paradigm is the key to improve the design productivity of SoCs (System on a Chip). However, SoC designers have difficulty in using conventional IPs (Intellectual Property) because they don't have enough variability, it is difficult to customize them. In this paper, we propose the variability of VCores (Virtual Cores) and show VCores are superior to IPs for system-level design property. Based on VCores, we have developed a VCore-based platform. We show the SoC design productivity will be improved by using VCores and the VCore-based platform.

1. Introduction

High integration of LSIs has brought design complexity more than before. In this situation, design reuse and raising the level of abstraction are the keys to improve the design productivity of SoCs. IPs have been used as design property up to now. However, its abstraction level is usually RT (Register Transfer)-level or less than that, so IPs are not suitable for system-level design property which is usually described using C-based languages and refined to SoCs with top-down methodology. In addition, conventional IPs have a problem that their variability is very limited. When considering the variability, we should take account of both functionalities and structures. However, since IPs can only vary their structures partly, it is difficult for SoC designers to customize and reuse them. Moreover, this causes another problem that IPs which are different as chalk and cheese are generated one after another, and this makes it difficult to handle them. In these circumstances, we have proposed VCores as system-level design property and VCDS (Virtual Core Design System) as a top-down design system for SoCs [1,2].

The feature of VCore modeling is their variability. While IPs have very limited variability (e.g. bit width), VCores can vary both functionalities and structures. VCores realize this variability using 3-tier models. There are three types of VCores, that is, method/algorithm variable models called "Source", structure variable models after being fixed method/algorithm called "Generic" and models called "Type" whose method/algorithm and structures are both fixed. SoC designers can vary VCores' functionalities and structures by changing the relations between the 3-tier models.

Besides, we have developed a VCore-based platform to manage VCores efficiently and promote reuse designs. The VCore-based platform consists of (1) VCore entry support system, (2) VCore utilization system, (3) Performance estimation system and (4) VCore distribution system.

In this paper, we present the details of VCore variability first. Second, we describe VCDB (VCore DataBase system) and VCore-based platform that centers VCDB and integrates tools for supporting VCore utilization. Finally, we discuss the relation between reuse designs and SoC design productivity and show VCores and the VCore-base platform are effective in SoC designs.

2. VCores and design reuse

2.1 Requirements for system-level design property

The requirements for system-level design property are (1) raising abstraction [3] and (2) improving reusability shown in Fig.1. As regards abstraction, describing the function of

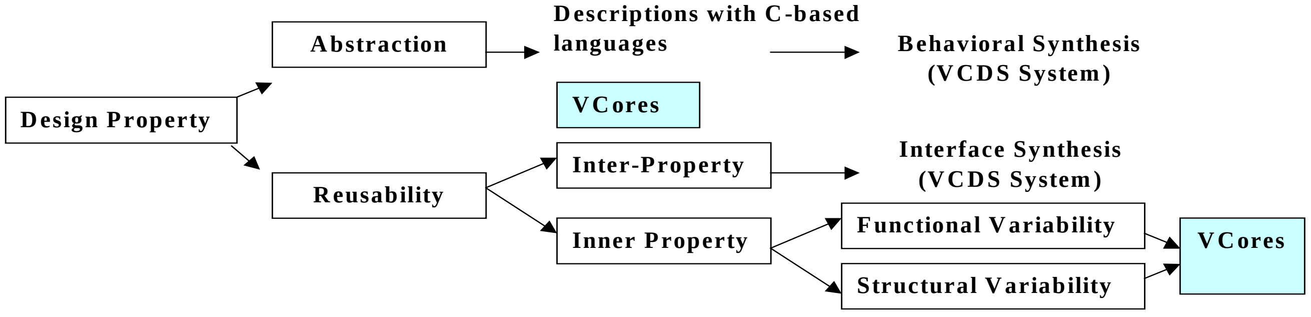
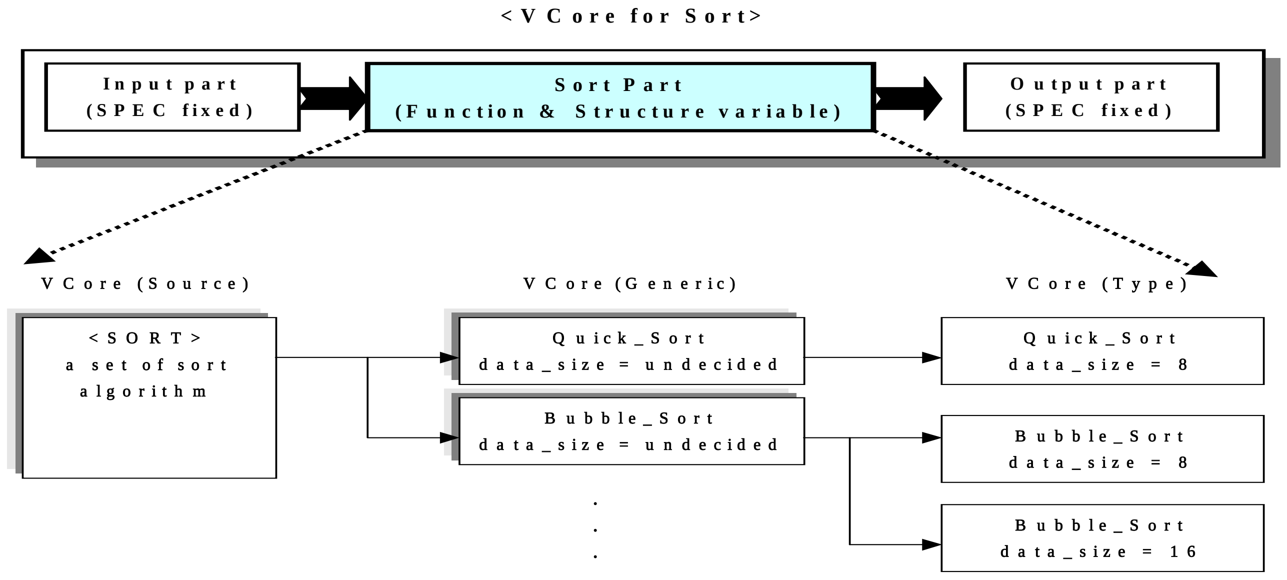


Fig.1 Requirements for design property in system-level designs



<VCore for Sort>
Input part (SPEC fixed)
Sort Part (Function & Structure variable)
Output part (SPEC fixed)
VCore (Source)
VCore (Generic)
VCore (Type)
<SORT> a set of sort algorithm
Quick_Sort data_size = undecided
Bubble_Sort data_size = undecided
Quick_Sort data_size = 8
Bubble_Sort data_size = 8
Bubble_Sort data_size = 16

Fig. 2 VCores' 3-tier model

property using C-based languages such as SystemC [4,5,6] and SpecC [7,8,9] without considering the concrete implementation of SoCs is a common approach. As for reusability, we have to consider the flexibility of both inter-property and inner property. But the flexibility of inter-property should be coped with in SoC design systems, such as interface synthesis tools. So we focus on the reusability and the flexibility of inner property here. On occasion when we consider the reusability of design property, we have to keep in mind that there are two types of reusability, that is, functionality and structures. However, as a matter of reality, preparing all kinds of design property with all possible functional and structural combinations in advance is almost impossible. So we adopt an approach of preparing functional and structural variable models (VCores) and generating necessary property.

2.2 Classification of VCores and 3-tier models

VCores are system-level design property and the information on VCores is described using VLEF (VCore Library Exchange Format) that is mentioned later, and stored in a VCore database after being compiled using VLEF compiler.

In terms of abstraction, there are three types of VCores. Functional VCores are used for the definition and the verification of system-level designs. Hardware VCores are used for the implementation of hardware parts of SoCs, and Software VCores are used for software parts of SoCs. On the other hand, there are also three types of VCores in terms of variability, that is, Source, Generic and Type that are mentioned in section 1. SoC designs with VCDS are proceeded by selecting VCores that have appropriate combinations of the abstraction and the variability.

Fig.2 shows a simple example of 3-tier models using sort functions. In this case, Source is a set of sort algorithm. Suppose you use two kinds of algorithm, quick sort and bubble sort. The attribution of selected VCores is Generic. The data size, which is a parameter of the VCore in this case, is not decided at this phase. Next, you specify the size parameter according to the specifications and Type whose algorithm and parameter are fixed is decided.

Next example is a SoC design using the 3-tier models (see Fig. 3). In this case, the SoC consists of two functional VCores F1 and F2. (In Fig.3, initial letter F means functional VCores, H means hardware VCores and S means software VCores.) If the function of F1 is image compression, F1 Source is a set of image compression methods or algorithm (e.g. MPEG1, MPEG2 etc.,). If you select a specific one according to the SoC specifications, then F1 Generic is fixed. The verifications for the system-level design are done at this stage. Next thing you have to do is to determine hardware implementation or software implementation. If you choose hardware implementation at the request of the specifications, H1 Generic corresponding to F1 Generic is selected. Finally, by specifying the concrete structural parameters of H1 Generic (e.g. bit width), H1 Type, which is the final implementation for F1 Source, is determined. Regarding F2 Source, if you choose software implementation, S2 Type is the final implementation.

As just described, you can select or specify (1) hardware or software implementation, (2) methods/algorithm and (3) structural parameters dynamically by using VCores. For this reason, VCore-based designs have the following features.

- It is possible to try several architectures in architecture designs [10].
- Because you can generate necessary design property according to the necessity, you do not have to store a lot of design property that are similar but different.

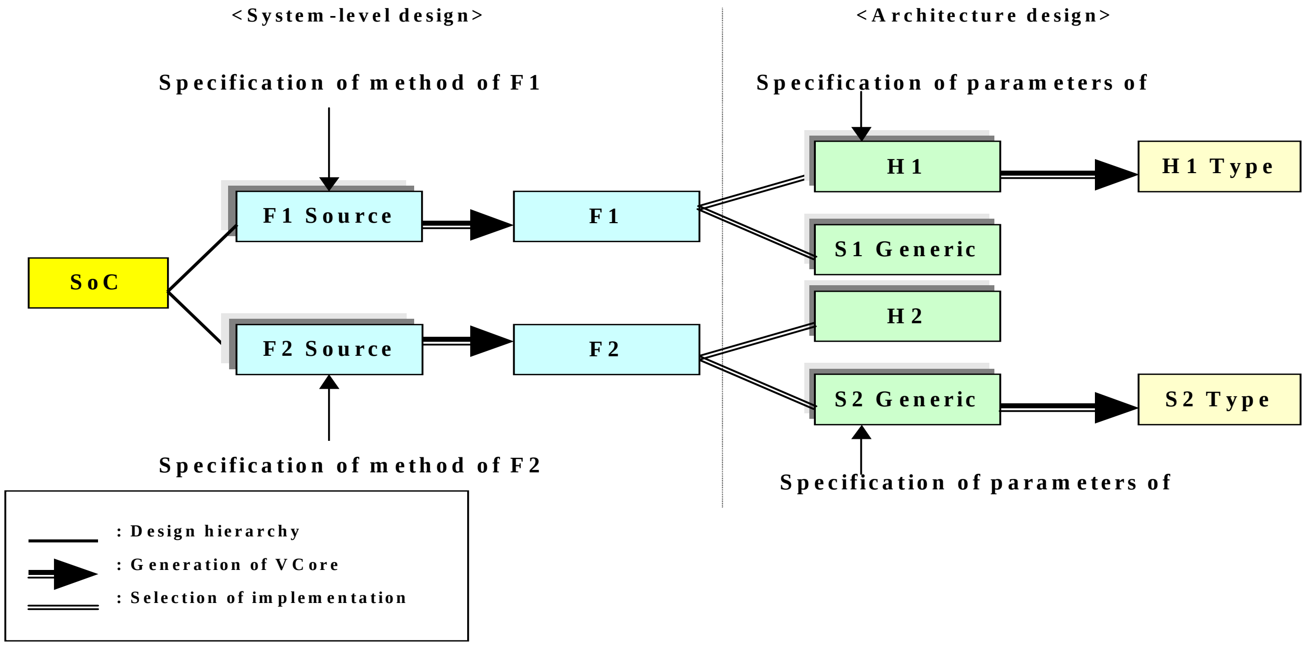

<System-level design>
<Architecture design>
Specification of method of F1
Specification of parameters of
SoC
F1 Source
F1
H1
H1 Type
S1 Generic
F2 Source
F2
H2
S2 Generic
S2 Type
Specification of method of F2
Specification of parameters of
: Design hierarchy
: Generation of VCore
: Selection of implementation

Fig. 3 3-tier models in SoC design

• As the relations between the VCores (Source-Generic, Generic-Type etc.) are easily modified using a VCore-based platform tool mentioned later, it is easy to adopt new methods or algorithm, or define new VCores.

In this way, VCore have enough abstraction levels and the reusability, they are superior to conventional IPs as system-level design property.

3. VCDB (VCore DataBase)

To manage and reuse property like VCores that are flexible enough, a special database for VCores is required. To achieve this purpose, we adopt an object-oriented database and developed VCDB which consist of APIs (Application Programming Interface) for handling VCores, a VCore search system and utilities for managing the database. Generally, VCDB is generated for each design site, so plural number of VCDB exists. In Fig. 4, my_vcore.vcdb shows a generated VCDB for one site. Information specified directly to VCores are stored in my_vcore.vcdb. However, storing documents such as specifications generated by a word processor or other documents generated by various tools into my_vcore.vcdb is not a sensible approach considering the independency of VCDB from tools. For this reason, these kinds of files are stored as attached files. In Fig.4, my_vcore.dic manages the addresses of these files. Thus VCDB can manage all kinds of information related to VCores.

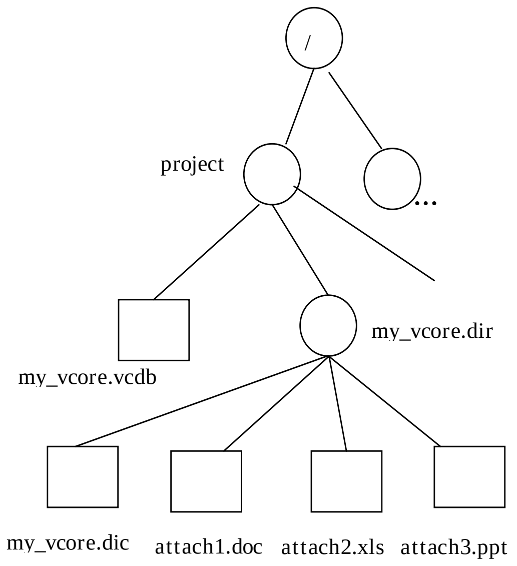


Fig.4 Directory structure of VCDB

4. VCore-based Platform

To design SoCs efficiently using VCores, not only database but also (1) VCore entry support system, (2) VCore utilization system, (3) Performance estimation system and (4) VCore distribution system are necessary. For this reason, we have developed a VCore-based platform that centers VCDB and integrates tools mentioned above. To be precise, (1) a GUI-based VCore entry system, a VCore Library Exchange Format VLEF and its compiler/reverse compiler, (2) a Java-based Web system for using VCDB from remote places, (3) an estimating system for VCore performance and

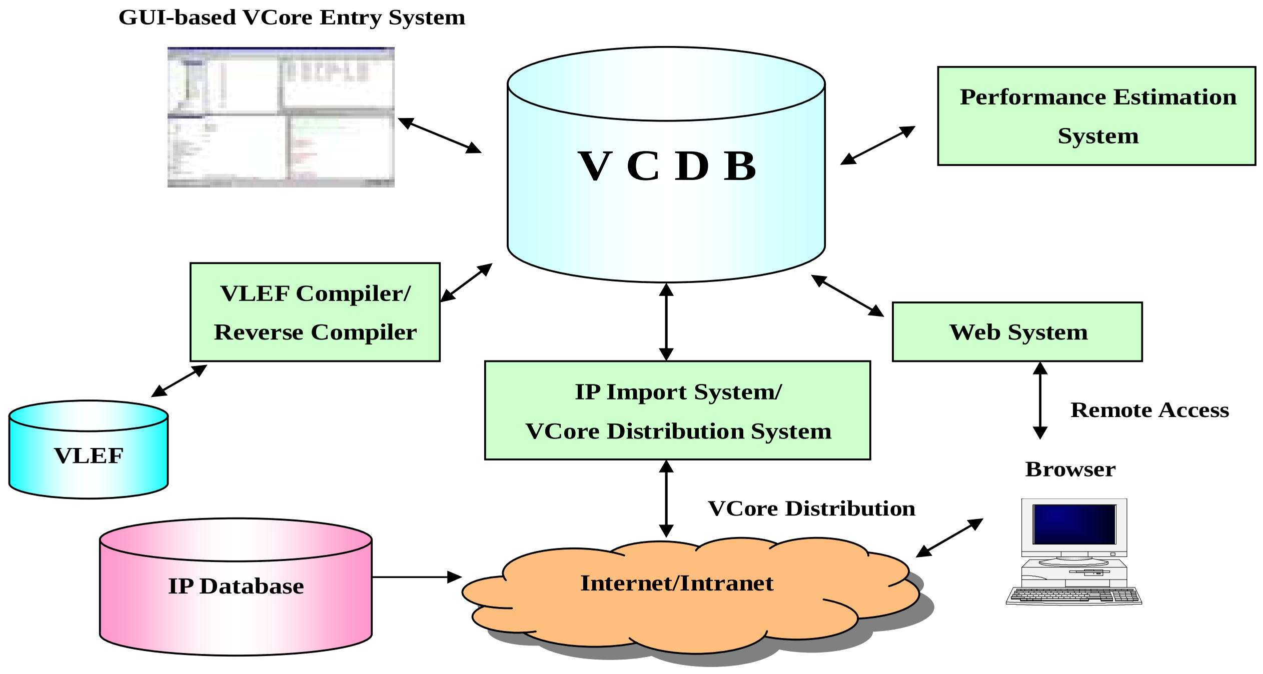

GUI-based VCore Entry System
V C D B
Performance Estimation System
VLEF Compiler/ Reverse Compiler
Web System
IP Import System/ VCore Distribution System
Remote Access
VLEF
Browser
VCore Distribution
IP Database
Internet/Intranet

Fig. 5 VCore-based platform

(4) a VCore distribution system and an IP import system into VCDB using IP distribution protocols. Fig.5 shows the system structure of the VCore-based platform.

4.1 VCore entry system

4.1.1 VCore Library Exchange Format (VLEF)

Registering VCores with a VCDB, it is desirable that VCores have not only C-based behavioral descriptions but also information on properties attached to VCores, considering reuse and distribution. So we have developed VLEF and its compiler/reverse compiler to define VCores clearly. The basic structure of VLEF is shown in Fig.6.

4.1.2 VCore entry system

When registering, editing or searching VCores, VCDBManager, which is a GUI-based VCore entry system, is used. Since VCDBManager has a VLEF compiler/reverse compiler, a VCore searching function and a display function of information on VCores, SoC designers can easily register and retrieve VCores using it. In addition, the edit of relations between Source-Generic and Generic-Type can be carried out using VCDBManager.

4.2 VCore utilization system

When using or distributing VCores, it is a common way to access VCDB from distant places through Internet or Intranet. To support this, we have developed Java-based Web system. This function enables SoC designers to access

```
VLEF
  <Version>
  <parameter definition>
  <Macro definition>
  <Unit metric definition>
  <Unit type definition>
  <Property type definition>
  <Key type definition>
  <File type definition>
  <Complex type definition>
  <Search list definition>
  <VCDB property definition>
  <VCDB attachment file definition>
  <VCDB comment definition>
  <VCore template definition>
  <VCore template quotation>
  <Calculation definition>
  <VCore definition>
  <Channel definition>
  <Interface definition>
ENDVLEF
```

Fig. 6 VCore structure

VCores from all over the world.

4.3 Performance estimation system

VCores have characteristic values that show their performance (including area, speed and power). When registering VCores or reconfiguring, it is desirable that designers can estimate the values rapidly. To support this, we have developed a RT-level rapid estimation tool on VCores

Table 1. The change of design productivity

	How to calculate	Year Explanation	1999	2002	2005	2011
(a)	From 1999 ORTC	Area ratio of logic gates to SoC(%)	80	50	35	15
(b)	From 1999 ORTC	Gate count(Mgates)	4	6.75	11.64	30.41
(c)	Assumption	Reuse rate of circuits(%)	20	50	70	90
(d)	(a) * (b)	New design parts of SoC(Mgates)	3.2	3.38	3.49	3.04
(e)	30%/3year (Assumption)	Design manpower ratio to 1999	1	0.7	0.49	0.24
(f)	(d) * (e)	Substantial new design (Mgates)	3.2	2.37	1.71	0.73
(g)	30%/3year (Assumption)	Improvement of reuse overhead (%) (Gate conversion)	50	35	24.5	12.01
(h)	(b) * (c) * (g)	Reuse resource(Mgates)	0.4	1.18	2	3.29
(i)	(f) + (h)	New design + reuse resource (Mgates)	3.6	3.55	3.71	4.02
	(i) / 3.6 * 10	Manpower (MY) (Assumption : 3.6Mgates(1999) requires	10	9.86	10.31	11.17

performances and integrated it into the VCore-based platform. When designers estimate performance of VCores, RT-level descriptions output by behavioral synthesis tools are used as input of the tool whose performance is 50-100 times faster than the de facto logic synthesis tool.

4.4 VCore distribution system

Regarding the distribution of design property, IP distribution has been established. So when distributing VCores, it is desirable to use the same protocols. The VCore-based platform can distribute VCores and IPs using the same protocols.

5. Analysis on reuse and design productivity

Table 1 shows how manpower on a SoC design changes depending on the improvement of reuse overhead and the efficiency of design methodology [11]. This table shows the manpower won't increase in spite of the increase of design size of SoCs if the reuse overhead decreases at 30% per 3 years and also the manpower for new designs decrease at 30% per 3 years. Now, we adopt the improvement of reuse overhead and the improvement of manpower for new designs as two parameters and simulate by fixing one of the two parameters within the framework of Table 1. Fig.7 shows the case that the improvement of manpower for new design is fixed at 30% per 3 years and Fig. 8 shows the case that the improvement of reuse overhead is fixed at 30% per 3years. As shown in Fig. 7, the improvement of reuse overhead affects overall design manpower sharply. On the other hand, Fig. 8 shows that if the improvement of

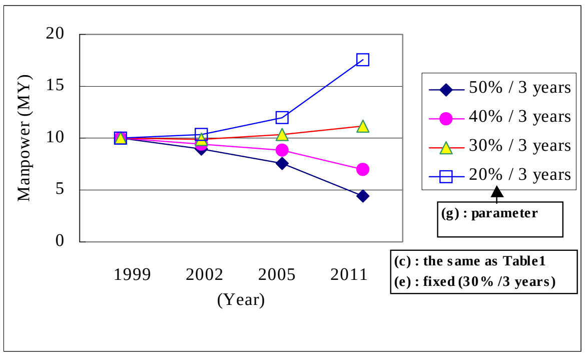


Fig. 7 Soc design productivity depending on the reuse overhead improvement

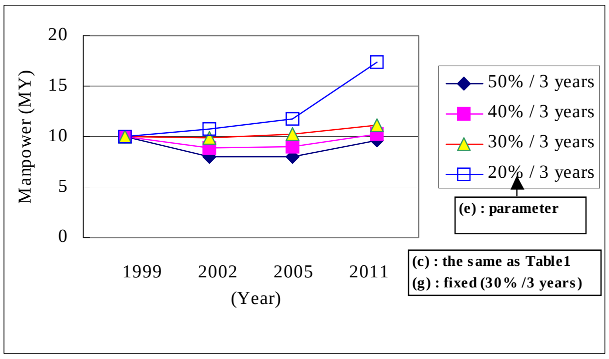


Fig. 8 SoC design productivity depending on the improvement of design efficiency

manpower for new designs is less than 30% per 3 years, overall design manpower will increase and even if the improvement rate is more than 30% per 3 years, the overall manpower will be flatten. This means, in future SoC designs, the design reuse is the key to improve design productivity. Since we have given enough variability to VCores and integrated tools into the VCore-based platform to reuse VCore efficiently, VCores and the VCore-based platform are precisely the reuse environment for SoCs.

6. Conclusion

In this paper, we presented the variability of VCores using 3-tier models first, and a database system (VCDB) for managing VCores and the VCore-based platform for reuse of VCores next. We also discussed the importance of the reuse in SoC designs and show VCores and the VCore-based platform are effective in the future SoC designs. To make the VCore-based platform a more useful reuse system, the enrichment of VCores is indispensable. So we are planning to develop a abstraction system from conventional IPs to VCores.

7. Acknowledgement

This work is sponsored by NEDO (New Energy and Industrial Technology Development Organization) as VCDS Project (SoC advanced design technology development project).

8. References

[1] : M.Muraoka, "VCDS:Virtual Core based Design System", ASP-DAC 1999.
[2] : M.Muraoka, H. Hamada, H. Nishi, T.Tada, Y.Onishi, T.Hosokawa, "VCore-based Design Methodology", ASP-DAC 2003.
[3] : A. Gerstlauer, D. Gajski, "System-Level Abstraction Semantics", Proceedings of the 15th international symposium on System Synthesis, October 2002.
[4] : T. Grötker, S. Lio, G. Martin, S. Swan, "System Design with SystemC", Kluwer Academic Publishers, 2002.
[5] : http://www.systemc.org.
[6] : D. Verkest, J. Kunkel, F. Schirrmeister, "System Level Design Using C++", Proceedings of the conference on Design, Automation and Test in Europe, January 2000.
[7] : A. Gerstlauer, R. Dömer, J. Peng, D. Gajski, "System Design: A Practical Guide with SpecC", Kluwer Academic Publishers, 2001.
[8] : http://www.specc.org.
[9] : http://www.cecs.uci.edu/~specc.
[10]: H. Nishi, M.Muraoka, R. Morizawa, H. Yokota, H. Hamada, "Synthesis for SoC Architecture using VCires", ASP-DAC 2003.
[11] : 1999 JEITA STRJ Report.