Synthesis for SoC Architecture using VCores

Hiroaki Nishi, Michiaki Muraoka, Rafael K. Morizawa, Hideaki Yokota, Hideyuki Hamada

Semiconductor Technology Academic Research Center (STARC)

Yusen Shin Yokohama Bldg,
3-17-2, Shin Yokohama, Kohoku-ku, Yokohama, 222-0033, Japan
E-mail: {nishi, muraoka, morizawa, yokota, hamada}@starc.or.jp

Abstract

In this paper, we propose a novel architecture synthesis method for SoC using VCores. VCores are reusable and configurable high-level descriptions. An initial SoC architecture, which consists of a CPU, buses, and peripherals, is generated based on an architecture template. The hardware and software tradeoff is possible on the architecture model after assignment of software VCores or hardware VCores. The assignment is based on the results of the architecture's performance estimation. We present a prototype of the synthesis for SoC architecture using VCores and an architecture level design experiment using this prototype.

1. Introduction

Reuse of high-level design intellectual properties is indispensable to reduce SoC (System-on-chip) design time. The VCDS (Virtual Core based Design System); a high-level design methodology using VCores (Virtual Cores) has been proposed [9] to address this problem.

VCores are reusable and configurable high-level descriptions. There are three types of VCores. Functional VCores are abstract function of hardware and/or software which is used at the system level. Hardware VCores and software VCores are the cores actually used for the design of SoC architecture at the architecture level. In the VCDS, system designers define the system level functional model of a SoC using functional VCores based on the specifications of SoC. Functional verification is performed using a simulator. The hardware and software tradeoff is possible on the architecture model after assignment of software VCores or hardware VCores.

It has been difficult for SoC designers to design and compare two or more architectures in a given product design period. In this paper we propose a novel architecture synthesis technique in VCDS, which performs architectural explorations in a short design time. The proposed architecture synthesis technique explores SoC architecture candidates by iteratively assigning software VCore and hardware VCore to the functional VCores present in the system level model specification.

In the proposed architecture synthesis flow, the basic hardware architecture is initially generated from an architecture template. An architecture template consists of the declarations of architectural components such as CPUs, buses, I/O controllers, RTOS (Real Time Operating System); and the information on the dependencies among them. Software VCores or hardware VCores are effectively assigned to the functional VCores in a system level model, and communication methods between these VCores are generated. After VCore assignment and communication generation, the performance of generated architecture is estimated. Tradeoffs between software and hardware are performed by reassigning hardware VCore to the software VCore that is a part of performance bottlenecks. The generated architecture model is tuned up by optimizing the VCore's parameters.

In order to evaluate the procedure of the proposed architecture synthesis, we have developed a prototype and have experimentally designed SoC architectures of a wearable computer.

The rest of this paper is organized as follows. In the next section we discuss previous work on architecture exploration. In section 3, we describe the functional VCore, software VCore, and hardware VCore. In section 4 we describe what is an architecture template for basic architecture generation. The proposed architecture synthesis flow is illustrated in section 5. Section 6 shows a design experiment using the proposed synthesis technique. Conclusions are presented in section 7.

2. Related work

Many research works on hardware/software codesign have been proposed [1] [2] [4] [6] [13] [14]. COSYMA [14] firstly partitions software components from a system specification and then iteratively moves parts of software to hardware until timing constraints are satisfied. On the other hand, VULCAN [6] starts from a complete hardware solution and moves parts of the system to software until the performance constraints are fulfilled. POLIS [1] has focused on control-dominated applications with a processor and custom hardware. It is the base technology that has been implemented to a commercial tool.

However, no research on hardware/software codesign using high-level design intellectual properties (IP) has been proposed.

During the hardware/software codesign, both hardware and software components should be modeled at the same abstraction level such as the transaction level. Also, in order to properly evaluate an architecture we must use the same performance measurement units for both hardware and software. With a few exceptions, most of the proposed methodologies do not take into account the performance measurement unit differences between hardware and software.

In order to address the above described problems we propose the concept of VCores and defined a methodology that uses it for architectural synthesis.

3. VCores

The VCDS uses three types of VCores for SoC design. Functional VCore is used for system level design of a SoC. Software VCore and hardware VCore are used to tradeoff the hardware and software architecture of a SoC. We explain the three types of VCores in the next sections.

3.1 Functional VCore

The functional VCore is a reusable functional-design element used at the system level design. It does not consider a particular software or hardware architectures. This means that it is not described considering the algorithms and the execution time on a target processor.

The functional VCore has event type ports and data ports and data is delivered through their ports.

The functional VCore consists of sequential operation parts and parallel operation parts. Functional VCore waits in a standby state until an event is received. Data transmission and reception between functional VCores is triggered by the event. Data transmission and reception consumes no time. This means that the functional VCore is defined as an untimed model.

We actually described functional VCores using the SpecC language [5] for a design experiment. Sequential operations between functional VCores are derived from an input event; the “par” statement of SpecC language models a parallel operation between functional VCores.

3.2 Software VCore

The software VCore is a reusable high-level software element. It is used for modeling embedded software at the architecture level design. The algorithms and data structures of a software VCore are selectable. It is also independent of the instruction set of target processors. Software VCore is modeled taking into consideration restrictions of memory size and processing algorithm that differs from functional VCore.

The Software VCore also has event type ports and data type ports. The Software VCore behaves sequentially or in parallel like a functional VCore. In parallel operation, synchronization is established by event input and output between software VCores.

There exists a relationship between functionally equivalent functional and software VCores. The ports of a functional and software VCores that have the same functionality are one to one related. The ports of software VCore and the related ports of a functional VCore have to be almost equal except for their data type. The reason for this difference is that the data type is a parameter determined by a particular implementation.

Software VCore has three characteristics: execution time, power consumption, and object code size. These characteristics are used for performance estimation at the architecture level design. The execution time and the power consumption of the software VCore are measured or calculated on an accurate processor, bus, and memory architecture model. The code size is determined by choosing a particular compiler and its optimization options for a processor.

3.3 Hardware VCore

The hardware VCore is a reusable functional element used at the high-level hardware design. Computation (processing) and input/output interface parts are separated. The computation algorithm of a hardware VCore can be reconfigured independently from the input/output interface part.

The hardware VCore has data ports and data is delivered through their ports.

The interface part of a VCores is synthesized by the hardware-hardware interface synthesis [10]. Behavioral synthesis tools are used to synthesize the computation part. There are restrictions for data types. Only integer type and their structures are allowed. Pointers are not allowed.

There exists a relationship between functionally equivalent hardware VCore and software VCore. This relationship is used for reassigning software VCore that is performance bottleneck to the hardware VCore.

The hardware VCore has three characteristics: area (transistor count) under a particular design rule, latency under a particular operation frequency, and power consumption under a particular CMOS device parameters.

4. Architecture template

The architecture of a SoC is usually designed by experts for every application domain such as cellular phones, digital still cameras, personal digital assistants (PDA), set-top boxes, network routers, etc. The architecture template is similarly designed by specialists for specific application domains.

The architecture template has declarations of hardware and software components that, together, will constitute the architecture of a SoC. The hardware components are processors, buses, I/O peripherals, etc. The software

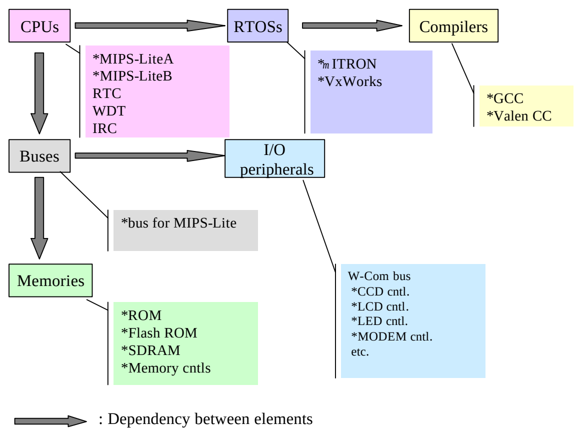


Figure 1. An example of the architecture template

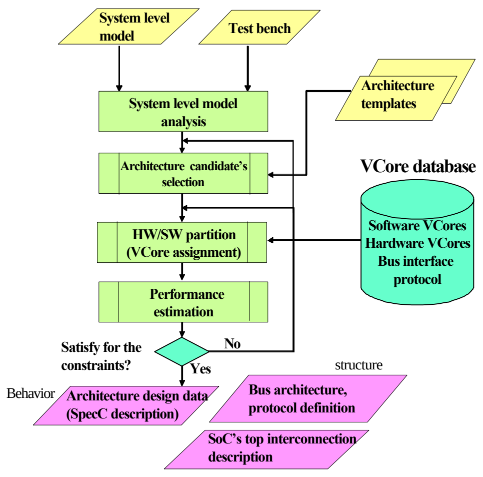

System level model
Test bench
Architecture templates
System level model analysis
Architecture candidate's selection
VCore database
Software VCores
Hardware VCores
Bus interface protocol
HW/SW partition (VCore assignment)
Performance estimation
Satisfy for the constraints?
No
Yes
structure
Behavior
Architecture design data (SpecC description)
Bus architecture, protocol definition
SoC's top interconnection description

Figure 2. Architecture synthesis flow

components are RTOSs, device drivers, development environments (such as compilers, debuggers, and instruction set simulators), etc. An example of architecture template is shown in Figure 1.

The dependency between components is the design knowledge that enables a SoC designer to adequately and easily select hardware and software architecture components.

For example, the ARM7XX processor and the AMBA bus can be chosen based on their dependency registered in the architecture template. The SH4 processor and the AMBA bus, however, cannot be chosen because the SH4 processor and the AMBA bus do not usually have a dependency, neither there is such dependency registered in the architecture template. The knowledge of dependencies between hardware/software components guarantees that an incorrect selection of components is never carried out.

The description of how the hardware components are interconnected to form an architecture is also provided in the architecture template. This hardware interconnection description is reconfigurable, thus enabling the addition custom hardware.

The architecture synthesis uses the architecture template in order to generate a basic hardware architecture (processor, bus, peripherals and their connections) of a SoC in a short time.

5. Architecture synthesis flow

Figure 2 shows the proposed architecture synthesis flow. In the VCDS, system designers define the system level functional model of a SoC using functional VCores based on the specifications of a SoC. Funcional verification is performed using a simulator.

The system level model analysis receives the system level functional model and its test data (a functional VCore network and their test bench) designed at the system level. Probes are inserted to the system level functional model to calculate the computation and communication load between functional VCores. It performs a simulation using the system level functional model with the input test data and outputs the log file as a result. The computation load is calculated as the number of times each functional VCore is executed per simulation from the log. The communication load is calculated as the integrated data transfer between functional VCores as an amount (bytes) of data per simulation.

In the selection of an architecture candidate, an architecture template is chosen based, for example on the maximum performance of a processor or the maximum bandwidth of a bus. After selecting an architecture template, the designer chooses the hardware and software components such as the processor and busses based on their dependencies, which are registered in the architecture template. The interconnection between the chosen hardware components is reconfigured.

In the VCore assignment step a software VCore or a hardware VCore are assigned to a functional VCore in order to trade off hardware and software, and to generate a model to estimate the architecture's performance. The VCore assignment strategy is to first assign software VCores to the functional VCores. Then, by iteratively estimating the hardware performance, determine the hot spots. Finally, reassign hardware VCores to the appropriate software VCores in order to eliminate the hot spots.

In the selection and generation of an interface between VCores, an abstract bus and a virtual device driver model (communication methods) are generated. The communication method between software VCore and hardware VCore is modeled as procedure of data transmission and reception. This procedure is the abstraction bus data transfer. Communication methods between VCores are refined to a synthesizable model. The interface synthesis tools support the synthesis of a single or burst bus transfer circuits from the synthesizable model. The communication method refinement is shown in Figure 3.

The performance of the software and hardware architecture is estimated analytically using the characteristic of VCores. Since VCores may execute two or more times

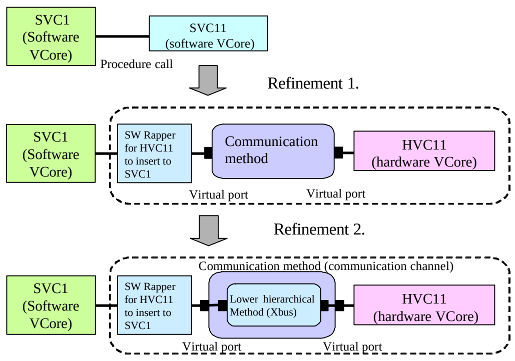


Figure 3. Refinement of the interface model between VCores

during a test operation, we have to give the number of times a VCore executes. This number can be obtained from the result of the system level model analysis. The software VCore execution time is defined as the time interval between the start and the end of a process. It is measured by an instruction set emulator considering the RTOS.

The performance estimation algorithm calculates the processing time of the hardware and software by multiplying the number of times of operation with the VCore's latency (in the case of hardware VCores) or execution time (in the case of software VCores) characteristic to find bottlenecks.

If the performance satisfies the design constraints, the HW/SW model description at the architecture level is generated. It consists of software VCores and hardware VCores. Furthermore the SoC's hardware interconnection description is generated (HDL description). The hardware interconnection description is needed to connect the bus and the bus interface circuits for hardware VCores after interface synthesis.

Explanation of the VCDS's interface synthesis is out of the scope of this paper. We next present the VCore assignment and the architecture performance estimation in detail.

5.1 VCore assignment

In the VCore assignment step, software VCores or hardware VCores are assigned to the system level functional model (the functional VCore network) in order to trade off hardware and software. The system level functional model consists of functional VCores and *direct communication* interconnecting them. A *direct communication* is defined as a data exchange that is implementation independent.

The processor is chosen at the architecture candidate's selection step using an architecture template. Software VCores are effectively assigned to the system level functional model as software that operates on the selected processor as shown in Figure 4. After assigning a software VCore to a functional VCore, a child software VCore that is

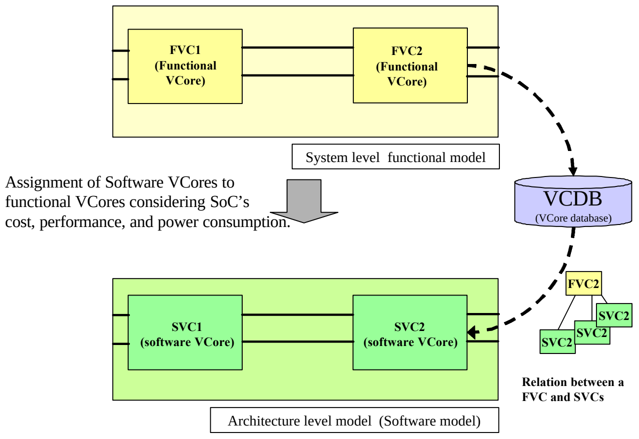

FVC1 (Functional VCore)
FVC2 (Functional VCore)
System level functional model
Assignment of Software VCores to functional VCores considering SoC's cost, performance, and power consumption.
VCDB (VCore database)
FVC2
SVC2
SVC2
SVC2
SVC1 (software VCore)
SVC2 (software VCore)
Relation between a FVC and SVCs
Architecture level model (Software model)

Figure 4. Assignment of software VCores to functional VCores

a performance bottleneck, according to the results of the performance estimation, is assigned to a hardware VCore.

Functional VCore and software VCore are designed to be functionally equivalent. Hardware VCores are also designed to be functionally equivalent to the leaf software VCores. A leaf VCore is defined as a VCore that cannot be further subdivided into other VCores.

Each VCores port must have a one to one correspondence in order to enable the implementation of an automatic assignment of functional VCore to software VCore, and software VCore to hardware VCore. We present the formalization of VCore assignment procedure below.

Preconditions: The communication methods between software VCore and hardware VCore, hardware and hardware VCores are registered into a VCore database. Also, the functional, software, and hardware VCores are registered into a VCore database.

Definition 1: The system level model SLM_i consists of a subset F_i of the functional VCore set F registered into a VCore database, a subset CF_i of set CF that is the *direct communication* between functional VCores in F_i, and a set SP_i of input/output ports.

$SLM_i = \{F_i, CF_i, SP_i\}$

Definition 2: The hardware and software model ALM_i at the architecture level consists of a subset Si of software VCore set S, a subset H_i of harware VCore set H, a set $CMSS_i$ of communication methods between software VCores in S_i, a set $CMSH_i$ of communication methods between software VCore in S_i and hardware VCore in H_i, a set $CMHH_i$ of communication methods between hardware VCores in H_i, and a set AP_i of input/output ports.

$ALM_i = \{S_i, H_i, CMSS_i, CMSH_i, CMHH_i, AP_i\}$

Definition 3: The VCore assignment problem is defined as assigning the hardware VCores, software VCores, communication methods, and the ports into a system level model SLM_i to satisfy design constraints (A_c, T_c, W_c). A_c, T_c, W_c are the area, the processing time, and the power consumption, respectively. Design constraints are satisfied by finding $ALM_i \subset ALM$ such that $Ai<=A_c$, and $T_i <=T_c$, and $W_i <=W_c$.

VCore assignment procedure 1: Generate an ALM_1 that implements all functions of SLM_1 with software VCores. S_1 is a set of software VCores. $CMSS_1$ is a set of communication methods between software VCores svc_i and svc_j in S_1, and AP_1 is a set of port at the architecture level.

$ALM_1 = \{S_1, CMSS_1, AP_1\}$

VCore assignment procedure 2: A software VCore S_j ($S_j \subset S_1$) and it is the performance bottleneck in ALM_1, is assigned to hardware VCore H_j ($H_j \subset H$). S_j and H_j are functionally equivalent. As a result, $CMSH_2$ ($CMSH_2 \subset CMSH$), which is a set of communication method between a software VCore and the hardware VCore, is generated. AP_1 is refined into AP_2 if the H_j communicates with hardware on the outside of ALM_2.

$ALM_2 = \{S_1\text{-}S_j, H_j, CMSS_1\text{-}CMSS_j, CMSH_2, AP_2\}$

VCore assignment procedure 3: Search for the VCores that satisfies the design constraints (A_c, T_c, W_c) among the sets of ALM_2; and determine the VCore elements; svc_i and hvc_i, the communication method elements, and the port elements.

$ALM_3 = \{svc_1,svc_2,..,svc_l;hvc_1,hvc_2,..,hvc_m;cmss_1,cmss_2,.., cmss_r;cmsh_1,cmsh_2,.., cmsh_t; ap_1, ap_2,..,ap_n\}$

Communication method refinement : We will show how communication method between VCores is refined by assuming that they are implemented in the SpecC language. The send and the receive communication methods between VCores can be specified using SpecC's channel class. The channel class can be hierarchically defined such that data transfers can be refined from the functional transaction to a bus transaction.

5.2 Performance estimation

Performance estimation for hardware and software partitions is carried out using static analysis (such as the path traces analysis in a hardware and software model) or the dynamic analysis with simulators [3]. It is known that the performance estimation accuracy and speed have a trade-off relation.

In architecture synthesis, we newly developed a static performance estimation tool to allow rapid iterations of hardware and software partitioning. In the architecture synthesis the performance of the generated software and hardware model is estimated by using the VCore characteristics.

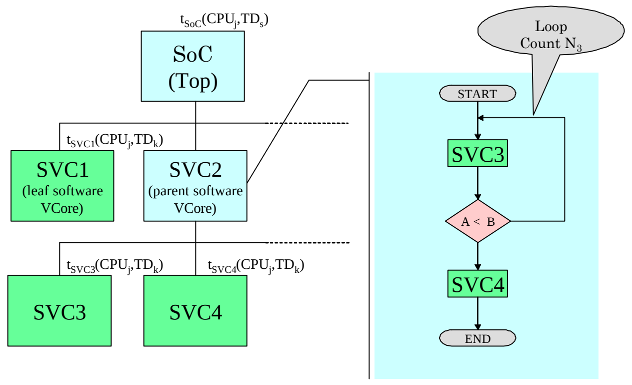
Performance is estimated using the characteristic of the leaf VCores. The designer must give the number of times a VCore is executed considering the test data because this number cannot be statically determined.

Definition 4: The execution time $t_{SVCi}(CPU_j,TD_k)$ of a software VCore SVC_i is defined as the execution time on the processor CPU_j with input test data TD_k.

When actually measuring the execution time of a software VCore we assume the following: (a) software will be generated from a software VCore; (b) the kernel of an RTOS schedules the software that carry out concurrent operation when they are performed on the single CPU; (c) we assume that no events cause preemption during software execution. In the case of software model as shown in Figure 5, it can be calculated by the following expression.

$$t_{soc}(CPU_j,TD_s) = a \times (t_{svc1}(CPU_j,TD_k) + t_{svc3}(CPU_j,TD_k)\times N_3 + t_{svc4}(CPU_j,TD_k) +,\ldots,+ t_{svcm}(CPU_j,TD_k)\times N_m) \qquad (1)$$

where a is a correlation coefficient between input test data TD_s and TD_k. The numbers of repetition times N_m in a parent VCore are set by a SoC designer.



$t_{SVCi}(CPU_j,TD_k)$:Execution time of SVCi on CPU_j architecture with test data TD_k

Figure 5. Performance estimation example

6. Architecture design experiment

In order to evaluate the design flow of the proposed architecture synthesis, we have developed a prototype system that has the principal functions proposed in this paper.

We experimentally designed a SoC architecture, which is embedded in a wearable computer (W-Com). The wearable computer has a videophone, a LCD, keys, etc. MPEG-4 compression algorithm [8] is used for encoding pictures and sounds. The specification requires that QCIF size video be decoded at a rate of 30 FPS.

In this experiment, we only designed the MPEG-4 decoder and used a simple model (reduced function model) of MPEG-4 algorithms. All the function of this example were specified using functional VCores. The software and hardware models of the architecture design were also designed using software VCores and hardware VCores. The list of the software and hardware VCores of the example is shown in Table 1.

Architecture components registered in the W-Com architecture template are the 32-bit MIPS-Lite processor (almost equivalent to the MIPS R3000 microprocessor [7]), the bus for MIPS-Lite and interface circuits to the peripheral controllers.

We describe an example of hardware and software trade-off. In case 1 (W-Com1), we used the architecture template selecting the 32-bit MIPS-Lite processor, the MIPS-Lite bus, and *m* ITRON Ver4.0 as the RTOS. In case 2 (W-Com2), we also used the same architecture template selecting the MIPS-Lite processor, the MIPS-Lite bus, and *m* ITRON Ver4.0 and transformed the IDCT (a part of MPEG-4 decoder) to a custom hardware.

Firstly, an initial architecture for the W-Com was generated. Architectural components were chosen according to the dependency between components from the architecture template especially defined for this example. Secondly, software VCores were assigned to 10 functional VCores in the system level functional model. We estimated the performance of the architecture assigned the software VCore. Thirdly, the software VCore that is the performance bottleneck was determined using the performance estimation. In this experiment the bottleneck was the IDCT. The IDCT software VCore was further assigned to the hardware VCores. After the communication architecture was chosen,

Table 1. List of VCores used in the experiment

VCore name	number of lines (SpecC)	execution time (10^{-6}sec)	note
MainM	1209	123	Main menu on a display
Ephng	13183	18	Starts memory dial
ShCut	2552	31	Set memory dial
Arrive	1417	118	Arrival function in video-phone
Srmes	1657	30	Answering machine
Srphg	594	18	Transmission and reception on the telephone
Mp4AS	753	21	MPEG-4 audio (simple)
Mp4VS	1146	19	MPEG-4 video (simple)
IndLED	358	39	LED control
DiLCD	637	70	LCD control
Memory	1852	18	Memory function
IDCT(s)	228	1227	Inverse discrete cosine transform (software)
IDCT(h)	217	125	Inverse discrete cosine transform (hardware)

Table 2. Performance figures of the synthesized architectures

architecture name	area (mm^2)	processing time (sec)[1]
W-Com1	1.14	36.6
W-Com2	1.43	25.0

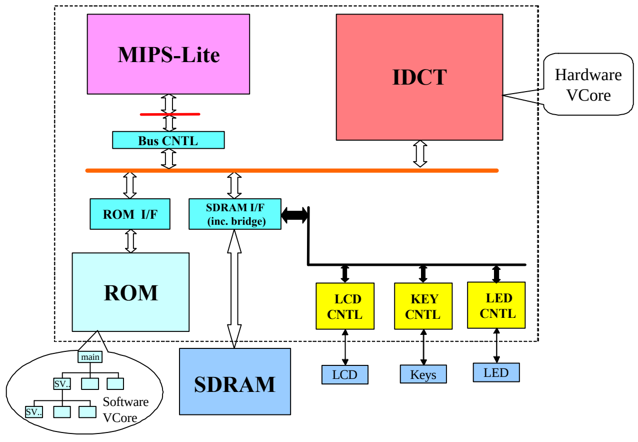


Figure 6. The principal part of the W-Com2 architecture

communication methods between the VCores were generated.

Table 2 shows the comparison of the two generated architectures. The processing time was estimated from the expression (1) using the VCore characteristics (execution time or latency) in Table 1. The area of the W-Coms was estimated from the hardware components characteristics (assuming that the components were pre-synthesized using STARC's 0.13-micrometer design-rule standard cells [12]). The area of W-Com2 increases 25%; the processing time is reduced 32% in comparison with W-Com1. Figure 6 shows the principal part of the W-Com2 architecture.

To design the RTL description for bus interface circuits of hardware VCore, we used the interface synthesis methodology introduced in reference [10]. For the generation of software from software VCores we used the software synthesis methodology introduced in reference [10].

7. Summary and Future work

In this paper, the methodology for synthesizing SoC architectures using VCores is proposed. We showed through an experiment that the proposed method can explore different SoC architectures. Further evaluation of the performance estimation method is needed in order to the architecture synthesis methodology be of practical use. In

[1] Processing time is the time interval measured from powering on the system until the display of the main menu and decoded video.

order to raise the performance estimation accuracy we have to consider the overhead of the RTOS APIs and the device drivers. A simulation-based estimation which takes into account the interface of VCores will be developed in the VCDS project.

We have a plan to develop an algorithm to select the best VCores from a VCore database considering their characteristics such as area, performance and power consumption. Using this algorithm we will be able to automate the VCore assignment step.

The proposed architecture synthesis can only generate a SoC with single CPU. Future work also includes extending the architecture synthesis methodology to support multi-processors.

Acknowledgement

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

References

[1] F. Balarin, E. Sentovich, M. Chiodo, P. Giusto, H. Hsieh, B. Tabbara, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki, and A.Sangiovanni-Vincentelli, *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*. Kluwer Academic Press, Boston, 1997.

[2] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, “Synthesis of Embedded Software from Synchronous Dataflow Specifications,” *Journal of VLSI Signal Processing Systems*, Vol. 21, No. 2, pp.151-166, June 1999.

[3] C.Brandolese, W.Fornaciari, F.Salice, D.Sciuto, “Source-level Execution Time Estimation of C Programs,” In Proceedings of the 9th International Symposium on Hardware / Software Co-Design (CODES), pp. 98-103, 2001.

[4] J.T. Buck, S. Ha, E.A. Lee and D.G. Messerschmitt, “Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems,” *Int. Journal of Computer Simulation*, *special issue on Simulation Software Development*, vol.4, pp. 155-182, April, 1994.

[5] D. Gajski, J. Zhu, R. Doemer, A. Gerstlauer, and S.Zhao, *SpecC: Specification Language and Methodology.* Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.

[6] R. K. Gupta and G. De Micheli, “Hardware-Software Cosynthesis for Digital Systems,” IEEE Design & Test of Computers, 10 (3): pp. 29-41, September 1993.

[7]G.Kane and J.Heinrich, *MIPS RISC Architecture*. Prentice Hall, 1992.

[8]MPEG-4,http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm

[9] M.Muraoka, “VCDS: Virtual Core based Design System,” ASP-DAC 1999.

[10] M.Muraoka, H.Hamada, H.Nishi, T.Tada, Y.Onishi, T.Hosokawa, K.Yoshida, “VCore-based Design Methodology,” ASP-DAC 2003.

[11] Y.Onishi, M.Muraoka, M.Utsuki, N.Tsubaki, “VCore-based Platform for SoC Design,” ASP-DAC 2003.

[12] STARC recommend “Common Design Rules for 0.13 micron,” http://www.starc.jp/kaihatu/ipgr/drlib/dr130-e.html, September 2000.

[13] G.Vanmeerbeeck, P.Schaumont, S.Vernalde, M.Engels, and I.Bolsens, “Hardware/software partitioning of embedded system in OCAPI-xl,” In Proceedings of the 9th International Symposium on Hardware / Software Co-Design (CODES), pp. 30-35, 2001.

[14] W. Ye, R. Ernst, Th. Benner, and J. Henkel, “Fast Timing Analysis for Hardware-Software Co-Synthesis,” In Proc. of the Int. Conference on Computer Design (*ICCD), pp. 452-457, Oct. 1993.*