Design Methodology for SoC Architectures based on Reusable Virtual Cores

Michiaki Muraoka, Hiroaki Nishi, 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: muraoka@starc.or.jp

Abstract

The design reuse methodology, which has been developed at the VCDS Project, is a SoC design methodology to reduce the SoC design time using high level design intellectual properties named as Virtual Cores (VCores). In this paper, we propose the VCore based design methodology to synthesize the SoC architecture from the system level specification. This synthesis methodology generates an initial architecture that consists of a CPU, buses, I/Os peripherals, and RTOS (Real Time Operating System), and makes tradeoffs between hardware and software on assigned software VCores and hardware VCores models to the architecture. The results of an architecture level design experiment using the proposed methodology shows that the partial automation of the communication refinement process, allied with design reuse, accelerates the architecture synthesis, thus reducing the design time required to design an architecture.

I. Introduction

The reuse of high-level design intellectual properties (IPs) is indispensable to reduce SoC (System-on-Chip) design time. The Virtual Core based Design System (VCDS) [13],[14],[15],[18],[19]; a high-level SoC design methodology using Virtual Cores (VCores) has been proposed to address this problem [13]. There are three types of VCores. Functional VCore is a reusable abstract functional model of hardware and/or software, which is used at the system level design. Hardware VCore and software VCore are reusable cores actually used at the architecture level design of SoC architectures [14].

In this paper, the Virtual Core based synthesis methodology of SoC architectures is presented. 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 efficiently assigned to the functional VCores in a system level model, and data transfer mechanism between these VCores is generated. After VCore assignment and communication method 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 optimized by tuning-up 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 VCores in the VCDS. In section 4, we explain the VCore based architecture synthesis flow within the VCDS. In section 5, we describe an architecture design experiment. Conclusion is presented in section 6.

II. Related Work

There have been proposed many research works on hardware/software codesign have been proposed [1] [2] [3] [8] [9] [17]. COSYMA [9] [10] 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 [8] 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, very few research works [11] on hardware/software codesign using high-level design IPs have been so far proposed. During the hardware/software codesign, both hardware and software components should be modeled at the same abstraction level such as the transaction level. In addition, in order to properly evaluate architecture we must use the same performance measurement units for both hardware and software. With a few exceptions, most of the previously proposed methodologies do not take into account the performance

measurement unit differences between hardware and software.

III. 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.

A. Functional VCore

The functional VCore is a reusable functional-design core used at the system level design. It does not consider a particular software or hardware architectures. This means that it is described without considering how it will be implemented in hardware or software.

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 [6] 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.

B. Software VCore

The software VCore is a reusable high-level software core. 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.

C. Hardware VCore

The hardware VCore is a reusable functional core 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 [14]. 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 has a 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.

IV. VCore based architecture synthesis flow

Figure 1 shows the proposed architecture synthesis flow within the VCDS. The input of the architecture synthesis flow is a system level functional model of a SoC, defined by system designers using functional VCores and based on the specifications of a SoC. Functional verification of this specification is performed using a simulator before the architecture synthesis phase.

The system level model analysis inputs are the system level functional model and its test data designed at the system level. It, then, analyzes the performance profile of the system level functional model.

In the selection of an architecture candidate, first an architecture template [15] is chosen based on the maximum performance of a processor, or the maximum bandwidth of a bus considering the result of the system level model analysis. After selecting an architecture template, the designer chooses the hardware components and the software components such as the processors, buses, RTOSes based on their dependencies that are registered in the architecture template.

In the HW/SW partition step, a software VCore or a hardware VCore is assigned to a functional VCore in order to trade off hardware and software. The VCore assignment

strategy is to first assign software VCores to the functional VCores. Then, by iteratively estimating the software performance, determine the hot spots. Finally, reassign hardware VCores to the appropriate software VCores in order to eliminate the hot spots. After VCore assignment, the interface between VCores, which is an abstract bus and a virtual device driver model (communication methods), are generated. The communication methods between VCores are refined to the synthesizable models.

The performance of the software and hardware architecture is statically estimated using the VCore characteristics such as the execution time [15]. The execution time of a software VCore is defined as the time interval between the start and the end of a process on a target processor. It is previously measured by an instruction set simulator considering the RTOS overhead.

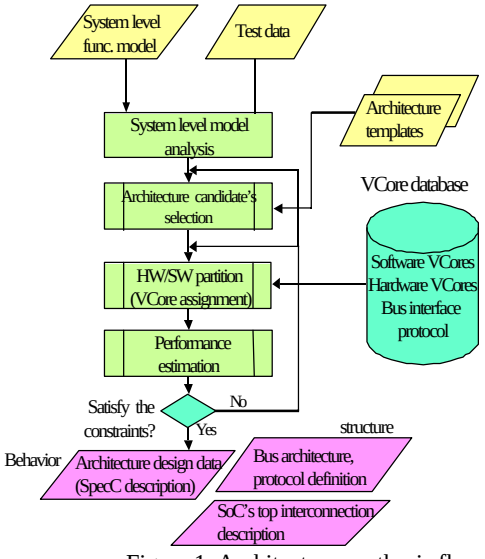

Figure 1. Architecture synthesis flow

The execution time of a hardware VCore is defined as the latency under a particular operation frequency. Since VCores in architecture model may execute two or more times during a normal operation, the designer has to give the number of times that a VCore is executed before estimation. This number can be obtained from the result of the system level model analysis. The performance estimation algorithm calculates the processing time of the hardware and software by multiplying the number of times of a VCore execution with the VCore's latency (in the case of hardware VCores) or execution time (in the case of software VCores) characteristics.

If the performance satisfies the design constraints, the HW/SW model descriptions at the architecture level (architecture design description, bus architecture description, SoC top netlist, etc) are generated.

A. 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

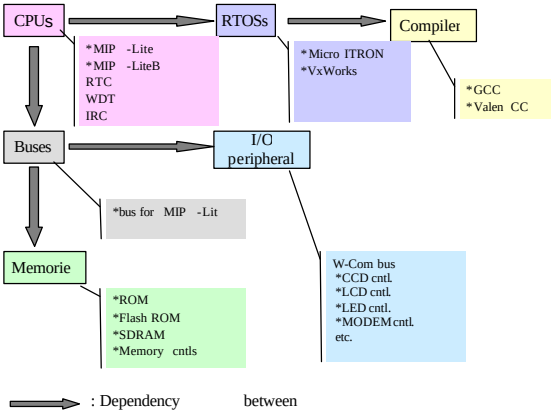

Figure 2. An example of the architecture template

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 2.

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.

B. VCore assignment

Before elaborating on the VCore assignment, we will review some features of a VCore [15]. 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 VCore's 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.

The system level functional model consists of functional VCores and direct communications 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. After assigning software VCores to functional VCores, the direct communications between functional VCores are refined to a transaction level communication such as a packet transformation (data transaction between processes). Leaf software VCores that are the performance bottleneck of the SoC (according to the results of the performance estimation) are assigned to hardware VCores.

C. Communication refinement

We show how to refine the communication method between a software VCore and a hardware VCore 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 packet transaction in a software description to a bus transaction in a hardware description.

The communication refinement is divided into two refinement phases to generate interface descriptions between VCores as shown in Figure 3.

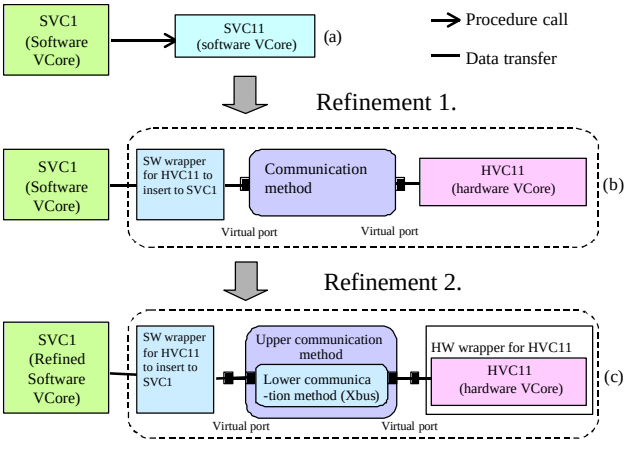

Figure 3. Interface refinement between VCores

Refinement 1

Step1: remove the software VCore (SVC11) internal description that is called from the parent software VCore (SVC1) in Figure3 (a).

Step2: select the hardware VCore (HVC11), which has the equal functionality with that of the software VCore (SVC11), from the VCore database.

Step3: map the selected hardware VCore (HVC11) to the software VCore (SVC11) in Figure 3 (b).

Step4: generate data transaction channel description (communication method) that includes input and output methods between the mapped hardware VCore and the parent software VCore in Figure 3 (b).

Step5: insert an input method and an output method call in the software wrapper for the hardware VCore(HVC11) in Figure 3 (c).

Table 1. The lower level data transfer methods for 32-bit bus

method name	data size (bits)	data transfer mode
Xbus_read_byte	8	Single read byte
Xbus_write_byte	8	Single write byte
Xbus_read_hword	16	Single read half word
Xbus_write_hword	16	Single write half word
Xbus_read_word	32	Single read word
Xbus_write_word	32	Single write word
Xbus_read_byte_burst	8	Burst read byte
Xbus_write_byte_burst	8	Burst write byte
Xbus_read_hword_burst	16	Burst read half word
Xbus_write_hword_burst	16	Burst write half word
Xbus_read_word_burst	32	Burst read word
Xbus_write_word_burst	32	Burst write word

Refinement 2

Step1: refine the communication method described as the data transaction channel under the user defined data transfer mode (Table 1).

Step2: generate data packing and slicing procedures that transform the data through the bus to the parent software VCore considering the bus data transfer mode (transferred data size and burst or single transfer mode).

Step3: insert a comment statement into the refined channel description that presents the connected bus name and protocol.

We show an example of the refinement of four 32-bit data bursts bus transfer between hardware VCore and software VCore with SpecC language in Figure 4. The original software VCore is replaced by the refined description shown in Figure 4(c) (svhstLIDCT10), which has a parallel execution of two behaviors (Figure 4(d)). One behavior is the software wrapper for the hardware VCore(the definition is shown in Figure 4(b)), and the other behavior is the hardware wrapper for the hardware VCore(the definition is shown in Figure 4(a)).

The communication method between the hardware

wrapper(Figure 4(a)) and the software wrapper(Figure 4(b)) is refined using the lower communication methods of xbus_read_word_burst, xbus_write_word_burst (Table 1).

```
// HW wapper for HW VCore
behavior     Gen_svhhtLIDCT10_HWEB(Gen_I_IDCT32_0_in      if_in,
Gen_I_IDCT32_0_out if_out)
{
note partition = "HW";
int InpData[64];          // Input Buffer
int OutData[64];          // Output Buffer
//IDCT hardware VCore instance
svhhtLIDCT10 hv(InpData, OutData);

void main(void)
{
// receive data (bus to hardware VCore)
if_in.Gen_IDCT32_0_in_receive(InpData);
   hv.main();   // IDCT hardware VCore call
// send data (hardware VCore to bus)
   if_out.Gen_IDCT32_0_out_send(OutData);
}
};                                                              (a)

// SW wapper for HW VCore
behavior Gen_svhstLIDCT10_SW(in int InpData[64],
out int OutData[64], Gen_I_IDCT32_0_in if_in,
Gen_I_IDCT32_0_out if_out )
{
note partition = "SW";
 void main(void) {

   // send to HWVcore
   if_in.Gen_IDCT32_0_in_send(InpData);

   //wait HW VCore Execution
   //receive from HW VCore
   if_out.Gen_IDCT32_0_in_receive(OutData)
}
};                                                              (b)

// Refined SW VCore
behavior svhstLIDCT10(in int InpData[64],  out int OutData[64])
{
note partition = "SW";
Gen_Chann_IDCT32_0 chann_InpData;
Gen_Chann_IDCT32_0 chann_OutData;
 Gen_svhstLIDCT10_SW svhstLIDCT10_swrap
(InpData,OutData,chann_InpData,chann_outData);
 Gen_svhhtLIDCT10_HWEB
svhhtLIDCT10_hwwrap(chann_inpData,chann_OutData)

void main
{
par{
 svhstLIDCT10_swwrap.main();
svhhtLIDCT10_hwwrap.main();                                     (d)
   }
 }
};                                                              (c)
```

Figure 4. Wrapper description of SW/HW VCore in SpecC.

The communication refinement result is shown in Figure5. Figure 5 (a) shows the description of the lower communication method, and Figure 5 (b) shows the description of the upper communication method.

V. Architecture design experiment

In order to evaluate the proposed architecture synthesis methodology, we have developed an architecture synthesis tool prototype. 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 [12] is used for encoding pictures and sounds. The specification requires that QCIF size video be decoded at a rate of 30 FPS.

```
//Lowest data channel (lower communication method)
interface I_xbus_EB_0 {
void xbus_write_word_burst(unsigned int data [4]);
void xbus_read_word_burst(unsigned int data [4]);
};
channel xbus_EB_0(void) implements I_xbus_EB_0
{
//definition of bus protocol
note protocol = "Gen_svhhtLIDCT10_HWEB";
unsigned int buff_word_burst[4];       // buffer
bool status_word_burst = false;        // status
event ready_word_burst, done_word_burst;
                                  // event for sync
  void xbus_write_word_burst(unsigned int data [4]){
      // ...omitted 10 lines ...
}
  void xbus_read_word_burst(unsigned int data [4]){
      // ...omitted 10 lines..
  }
};                                                              (a)
// Generated Data channel for IDCT
interface Gen_I_IDCT32_0_in {
//send data method
void Gen_IDCT32_0_in_send(int InpData_send[64] );
//receive data method
void Gen_IDCT32_0_in_receive
(int InpData_receive[64]);
};

//Data transaction channel (upper communication method)
channel Gen_Chann_IDCT32_0(void)
implements Gen_I_IDCT32_0_in, Gen_I_IDCT32_0_out
{
note partition = "HW";
xbus_EB_0 buschannel;
void Gen_IDCT32_0_in_send(int InpData_send[64])
{
int i, k, kmax;
unsigned int buff[4];
k = 0;          kmax = 4;
for (i=0; i<64; i+=kmax){
   // ...omitted  ...
////////// Force xbus write after array //////////
if(k > 0) {buschannel.xbus_write_word_burst(buff); k = 0 ;}
//the lowest write method call
   }
  }
void Gen_IDCT32_0_in_receive(int InpData_receive[64])
{
int i, k, kmax;
unsigned int buff[4];
k = 0;          kmax = 4;
//////// Initial xbus read //////////
buschannel.xbus_read_word_burst(buff);
//the lowest read method call
for (i=0; i<64; i+=kmax){
// ... omitted  ...
}
  }
};                                                              (b)
```

Figure 5. IDCT's channel description after refinement

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 was 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 2.

Functional VCores, software VCores and hardware VCores were designed using SpecC language [4] beforehand. Firstly, an initial architecture of the W-Com, which has 32-bit MIPS-Lite processor (almost equivalent to the MIPS R3000 microprocessor [6]) and W-Com 32-bit read and write buses (almost equivalent to the EC bus [5]) was configured using the W-Com architecture template. Secondly, software VCores were assigned to the 10 functional VCores in the W-Com system level model. We estimated the performance of architecture, which were assigned the software VCores. Thirdly, the software VCores that were performance bottlenecks were further assigned to hardware VCores.

Table 2. List of VCores used in the experiment

VCore name	number of lines (SpecC)	execution time (10 –6sec)	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, 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 for answering machine
IDCT(s)	228	1227	Inverse discrete cosine transform (software)
IDCT(h)	217	125	Inverse discrete cosine transform (hardware)

In this experiment, the performance bottleneck was the Inverse Discrete Cosine Transform (IDCT) in a part of MPEG-4 decoder. The IDCT software VCore was further assigned to the hardware VCores. Before selection of the communication method (data transfer mode on the bus), we calculated channel transmission delay Tcd using the equation (1) [7]

$$T_{cd} = (C_{ss} + (C_{sb}+C_{ct})*N_b)/F_c \qquad (1)$$

C_{ss} is synchronization cycles per transfer session, C_{sb} is the number of synchronization cycle per burst, C_{ct} is the number of transmission cycles per burst, N_b is the number of burst, F_c is clock frequency of the bus.

Consider the W-Com read bus transmission delay between MIPS-Lite processor and a IDCT hardware VCore that has 64 integer I/O ports. We assumed no wait state of the bus and arbitration, $C_{ssr}=0, C_{sbr}=2, C_{ctr}=6$ from the specification, $N_{br}=16$ in case of 4 bursts, $F_c=50*10^6$.

$$T_{cdr} = (0+(2+6)*16)/(50*10^6) = 2.56*10^{-6} (\text{ Sec})$$

Table3. Channel transmission delay between MIPS-Lite and IDCT

Burst size	read (10^{-6}sec)	write(10^{-6}sec)
single	3.84	3.84
4	2.72	2.56
8	2.00	2.56

Table 3 shows the channel transmission delays between MIPS-Lite and IDCT in case of a single, 4 and 8 bursts.

After the generation of communication methods between VCores, the processing time on a CPU_j and hardware VCore HVCl was estimated from the expression (2) using the VCore characteristics (execution time in the case of hardware VCores, or latency in the case of software VCores) in Table 2 and the channel transmission delay (1).

$$t_{SOC}(CPU_j, TD_s) = \sum_m (a_m \times t_{SVCm}(CPU_j, TD_m) \times N_m) + \sum_l (t_{HVC1} \times N_l + T_{cdl} \qquad (2)$$

$t_{SVCm}(CPU_j, TD_m)$ is the software VCore SVCm execution delay on CPU_j with test data TD_m. a_m is a correlation coefficient between input test data TD_s and TD_m. t_{HVCl} is the execution time of hardware VCore under the assumed clock frequency. The numbers of repetition times $N_{m(l)}$ of a VCore is a parameter that must be set by a SoC designer. T_{cdl} is the channel transmission delay between CPU_j and the hardware VCore HVC_l.

Table 4 shows the performance figures of four generated architectures. W-Com1 is the case of all software solution. W-Com2X is the cases of software solution without IDCT. W-Com2A is the case of single mode transfer between MIPS-Lite and the IDCT. W-Com2B is the case of four 32-bit data bursts transfer. W-Com2C is the case of eight 32-bit data bursts transfer.

Table 4. Performance figures of generated architectures

Architecture name	area (mm2)	processing time (sec)1
W-Com1	1.14	107.2
W-Com2A	1.43	36.6
W-Com2B	1.53	36.2
W-Com2C	1.64	36.0

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 [16]). The area of W-Com2B increases 25%; the processing time is reduced 66% in comparison with W-Com1. From the results of W-Com2B, W-Com2C, however, the timing reduction is a little in spite of the area overhead for burst transfer. The performance improvement was obtained by architecture optimization, and we selected the best architecture W-Com2A, which satisfies the design requirements.

To design the RTL description for bus interface circuits of hardware VCore, we used the interface synthesis methodology introduced in reference [14]. For the generation of software from software VCores we used the software synthesis methodology introduced in reference [14].

VI. Conclusion

The authors have presented an architecture synthesis methodology that is useful for performing hardware/software tradeoff on SoC design. We have also developed a tool that can perform the VCore assignment and the interface refinements described above. We have shown that the proposed synthesis methodology can explore different SoC architectures. Future work includes the improvements and extension of these methodologies to multi-processor architecture.

Acknowledgements

This work was 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] 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, 1994.

[3] 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.

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

[5] ECTM Interface Specification,Rev.1.03. MIPS Technologies Inc. ,2000.

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

[7] P. V. Knudsen and J. Madsen, "Integration Communication Protocol Selection with Hardware/Software Codesign," IEEE Trans. On CAD, 18 (8): pp. 1077-1095, 1999.

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

[9] 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.

[10] R.Ernst, J. Henkel, and Th. Benner, "Hardware-Software Cosynthesis for Microcontrollers", IEEE Design & Test of Computers, Vol. 10, No. 4, pp. 64-75, Dec. 1993.

[11] W. Wolf, "A Decade of Hardware/Software Codesign,", IEEE Computer, Vol. 36, No. 4, pp. 38-43, April 1993.

[12] MPEG-4,http://mpeg.telecomitalialab.com/standardsmpeg-4/mpeg-4.htm

[13] M.Muraoka, "VCDS: Virtual Core based Design System," ASP-DAC 1999.

[14] M. Muraoka, H. Hamada, H. Nishi, T. Tada, Y. Onishi, T. Hosokawa, K. Yoshida, "VCore-based Design Methodology," ASP-DAC2003, pp. 441-445, 2003.

[15] H. Nishi, M. Muraoka, R. K. Morizawa, H. Yokota, H. Hamada, "Synthesis for SoC Architecture Using VCores," ASP-DAC2003, pp. 446-452, 2003.

[16] STARC recommends "Common Design Rules for 0.13micron"http://www.starc.jp/kaihatu/ipgr/drlib/dr130-e.html, September 2000.

[17] 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.

[18] Y. Onishi, M. Muraoka, M. Utsuki, and N. Tsubaki, "VCore-based Platform for SoC Design", ASP-DAC2003, pp. 453-458, 2003.

[19] T. Hosokawa, H. Date and M. Muraoka, "A Test Generation Method Using a Compacted Test Table and a Test Generation Method Using a Compacted Test Plan Table for RTL Data Path Circuits", in Proc. of 20th VLSI Test Symposium (VTS'02), pp. 328-335, Monterey, April 2002