PAPER *Special Section on VLSI Design and CAD Algorithms*

Automatic Communication Synthesis with Hardware Sharing for Multi-Processor SoC Design**

Yuki ANDO†a), **Seiya SHIBATA**†*, *Nonmembers*, **Shinya HONDA**†, **Hiroyuki TOMIYAMA**††, *and* **Hiroaki TAKADA**†, *Members*

SUMMARY We present a hardware sharing method for design space exploration of multi-processor embedded systems. In our prior work, we had developed a system-level design tool named SystemBuilder which automatically synthesizes target implementation of a system from a functional description. In this work, we have extended SystemBuilder so that it can automatically synthesize an area-efficient implementation which shares a hardware module among different applications. With SystemBuilder, designers only need to enable an option in order to share a hardware module. The designers, therefore, can easily explore a design space including hardware sharing in short time. A case study shows the effectiveness of the hardware sharing on design space exploration.

key words: *system-level design, hardware sharing, design space exploration, MPSoC*

1. Introduction

Embedded systems have been increasing their complexity, consisting of more applications. In many cases, some of the applications include common functions. In order to optimize the system performance and their hardware size, the common functions are often implemented in a dedicated hardware module which is shared by the applications.

In order to design complex embedded systems, system-level design has been proposed. In the system-level design, designers design a system at a high level of abstraction. They start the design from describing system functionalities as a set of applications, and an application in turn consists of processes and channels. Processes and channels indicate computations and communications, respectively. Then, the processes are mapped to Processing Elements (PEs) such as CPUs and dedicated hardware modules, and the channels are mapped to buses and memories.

A number of system-level design tools which support process and channel mapping were proposed in the past [1]. However, process-level hardware sharing, i.e., mapping of processes which exist in different applications onto a single hardware module, is not supported by most of the existing system-level design tools. Most of the tools assume single-application systems. Although some tools assume multiple applications, they do not allow process-level hardware sharing. Even if it is allowed, the tools do not automatically synthesize interface circuitry which realizes mutually exclusive accesses to the shared hardware module. Therefore, the designers need to implement the interface circuitry manually.

This paper presents automatic synthesis of communications for hardware modules which are shared by multiple applications. Our system-level design tool named SystemBuilder has been extended so that it supports process-level hardware sharing. SystemBuilder automatically generates interface circuitry for the shared hardware module. Since the applications may run concurrently, the interface circuitry generated by SystemBuilder realizes mutually exclusive accesses to the shared hardware module.

The rest of this paper is organized as follows. Section 2 introduces the related works. Section 3 explains a brief overview of SystemBuilder. Section 4 presents the detail of communication synthesis for hardware sharing. Section 5 shows the effectiveness of hardware sharing through a case study on Advanced Encryption Standard (AES) system, and Sect. 6 concludes this paper.

2. Related Works

Various researches have been conducted on system-level design tools. The tools mainly assume heterogeneous Multi-Processor System-on-a-Chip (MPSoC) as target architecture.

SCE (System-On-Chip Environment) [2] is a system-level design framework based on the SpecC language [3]. It realizes an interactive and automated design flow with a consistent and seamless tool chain, and supports all the way from specification of the system down to hardware/software implementation.

Artemis [4] provides modeling and simulation methods and tools for efficient performance evaluation and exploration of heterogeneous embedded multimedia systems. Artemis's design flow start at a sequential application specification, and it is transformed to a concurrent application specification. Then, Artemis allows designers to estimate performance through co-simulation of a concurrent application specification.

PeaCE (Ptolemy extension as a Codesign Environ-

Manuscript received March 17, 2010.
Manuscript revised June 21, 2010.

†The authors are with the Graduate School of Information Science, Nagoya University, Nagoya-shi, 464-8601 Japan.
††The author is with College of Science and Engineering, Ritsumeikan University, Kusatsu-shi, 525-8577 Japan.
*The author is a research fellow of the Japan Society for the Promotion of Science.

**A preliminary version of this paper has been accepted for regular presentation at The IEEE International Symposium on Circuits and Systems (ISCAS), May 2010.

a) E-mail: y_ando@ertl.jp

DOI: 10.1587/transfun.E93.A.2509

ment) [5] is a hardware-software codesign environment that provides seamless codesign flow from functional simulation to system prototyping. Its target application is multimedia applications with real-time constraints. Unlike other system-level design tools, PeaCE is a reconfigurable environment into which other design tools can be easily integrated.

Metropolis [6] is a modeling and simulation environment based on the platform-based design paradigm. It provides a general, proprietary metamodel language that is used to capture separate models for behavioral model, platform model, and their binding and scheduling. Metropolis itself does not define any specific design tools but rather a general framework and language for modeling with support for simulation, validation and analysis of models.

ARTS [7] provides a simulation platform for modeled in SystemC. It supports multiple PE models and network model among PEs. ARTS assumes that the application model simulated on it is already developed and separated properly in order to explore allocation to PEs.

These five tools do not support hardware sharing. Our system-level design tool, SystemBuilder, is the first system-level design tool which automatically synthesizes communication for hardware sharing.

3. SystemBuilder

In this section, we show a brief overview of SystemBuilder to make this paper self-contained. Like other system-level design tools, SystemBuilder assumes MPSoC as target architecture. Please refer [8] for the detail of SystemBuilder.

Figure 1 shows the mapping and synthesis overview of SystemBuilder. SystemBuilder takes a functional description, an architecture template and mapping information as input, and generates target implementation of the system. The functional description, the architecture template and the mapping information represent the system functionalities, a target platform and an allocation of processes to PEs, respectively. The functional description consists of a set of applications, each of which in turn consists of a set of processes and channels. The applications are written in the C language with communication APIs which are interfaces to channels. Depending on the mapping information, a process is implemented as either a software task running on a real-time OS or a hardware module.

Channels represent communications among processes. Channels are generally classified into two types: one is asynchronous and the other is synchronous. Asynchronous channels are used to transfer data among processes. On the other hand, synchronous channels are mainly used to transfer events for activation of processes as well as synchronization between processes. In order to store multiple events, the synchronous channels have buffers. Depending on the mapping information, communication APIs are translated into either interface programs or hardware logics so that the processes can communicate with each other through the channels.

4. Automatic Communication Synthesis with Hardware Sharing

4.1 The Design Flow with Hardware Sharing

Figure 2 shows the design flow of SystemBuilder. Designers first design applications independently as shown in (a). Without hardware sharing, SystemBuilder generates the system implementation as shown in (b).

We assume that a system consists of more than one application, and some processes in the different applications have same functionality. In the description (a), there are two applications, Application1 and Application2, and processes P_B and P_Y have same functionality. With hardware sharing, SystemBuilder automatically converts the description (a) into an internal description (c). During the conversion, processes P_B and P_Y are merged into a new process P_S with the same functionality, where process P_S is

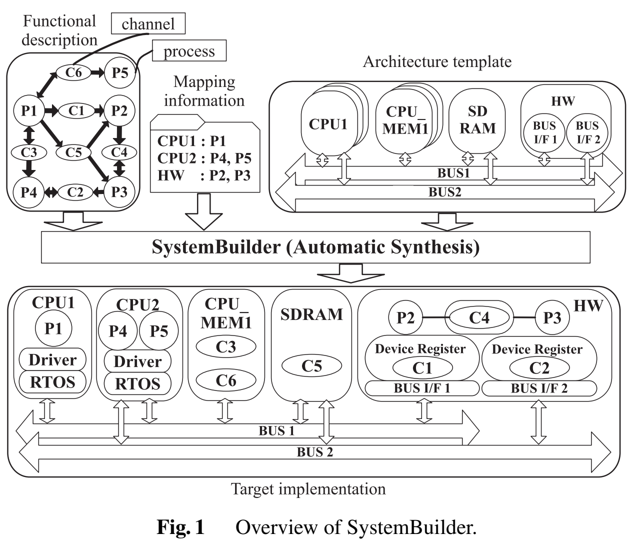


Fig. 1 Overview of SystemBuilder.

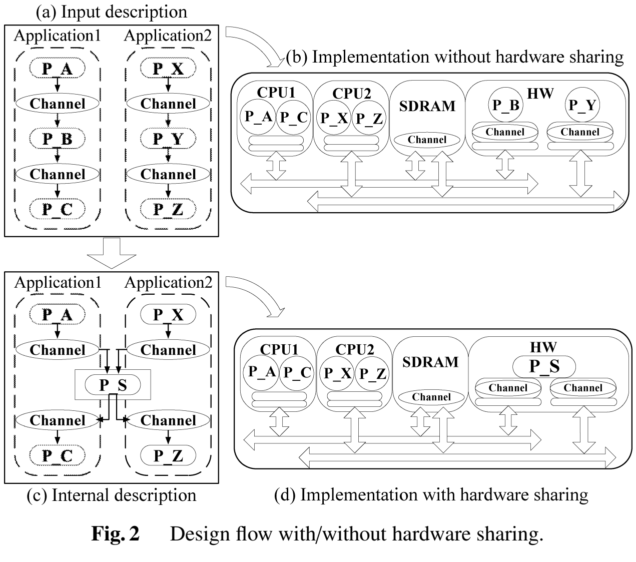


Fig. 2 Design flow with/without hardware sharing.

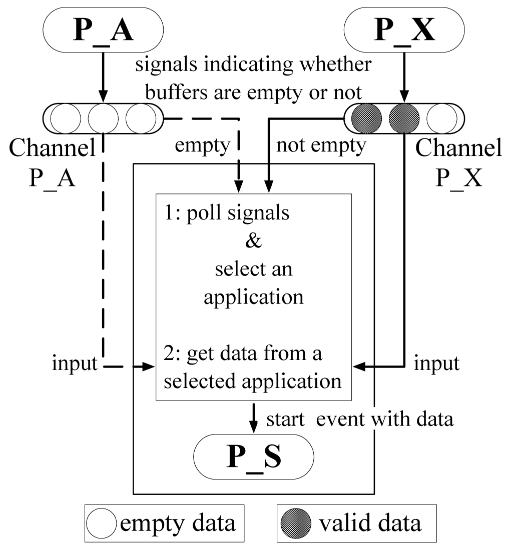

P_A
P_X
signals indicating whether buffers are empty or not
Channel P_A
Channel P_X
empty
not empty
1: poll signals & select an application
2: get data from a selected application
input
input
start event with data
P_S
empty data
valid data

Fig. 3 Detail of the wrapper generated by SystemBuilder.

shared by the two applications. In our hardware sharing method, the number of channels in Application1 and Application2 does not change. In other words, the channels are not shared by Application1 and Application2. Thus, no data conflict occurs within the communication channels. Then, SystemBuilder automatically synthesizes the implementation as shown in (d) from the internal description (c).

SystemBuilder automatically completes the design flow from (a) to (d) in Fig. 2 if a sharing option is enabled in the mapping information. Designers only need to turn on the option so that the two applications share a hardware module. The designers, therefore, will be able to explore a wider design space in short time. Note that more than two applications can share a hardware module although Fig. 2 only shows two applications.

4.2 Implementation of Communication for Hardware Sharing

Process P_S in Fig. 2(c) is shared by Application1 and Applicaiton2, and thus process P_S requires two sets of channels, one for Application1 and one for Application2. However, note that the process originally has only a single set of the channels. Also note that the functionality inside the process should not be modified for reusability and easiness of debugging.

SystemBuilder automatically inserts a wrapper to the shared process as shown in Fig. 3. The wrapper has two sets of external channels, i.e., one for each application. In addition, the wrapper provides an interface to the shared process. The wrapper realizes mutual exclusion and selects a channel to which the shared process should access. Also, SystemBuilder inserts a signal to channels which are connected to the shared process. The signal indicates whether a buffer in the channel is empty or not.

The wrapper works as follows. First, the wrapper polls the signals whether the channels have valid data or not. If more than one channel have valid data, the wrapper selects an application to be served. SystemBuilder supports two types of polling, priority-based polling and round-robin one. Designers select the polling policy and decide priorities of the applications in the mapping information. With priority-based polling, every time the shared process starts polling, the channel of the highest priority application is checked first. If its signal indicates empty, the next highest priority application will be checked. With round-robin polling, the channels are checked in a round-robin manner. The polling continues until non-empty signal is found.

Next, data are read from the channel of the selected application, and the wrapper sends a start event and the data to the shared process. Then, the shared process starts its execution.

The shared process may communicate with other processes not only at entry and exit points of the process but also during its execution. Every time the shared process communicates with another process, the wrapper passes the data between the shared process and the channel of the selected application.

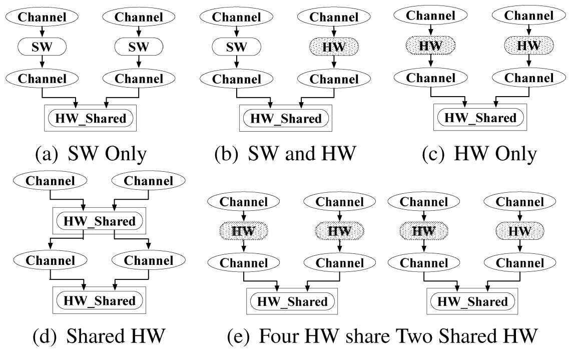


Fig. 4 Mapping of processes with hardware sharing supported by SystemBuilder.

4.3 Mapping of Processes with Hardware Sharing

Figure 4 shows four patterns of processes' mapping with hardware sharing supported by SystemBuilder. Our hardware sharing method does not restrict hardware/software mapping possibilities. This means that shared processes are able to communicate with processes to be implemented in software as well as ones to be implemented in hardware as shown in Fig. 4(a), Fig. 4(b), and Fig. 4(c). Furthermore, shared processes can communicate with other shared processes as shown in Fig. 4(d). Also, our hardware sharing method does not restrict the number of shared processes. For example, there can be two shared processes among four processes as shown in Fig. 4(e).

5. A Case Study

In this section, we present a case study to show the effectiveness of our hardware sharing method. Section 5.1 explains the target systems. Section 5.2 shows the evaluation of design space exploration with hardware sharing. Section 5.3 indicates the reduction of hardware size and Sect. 5.4 makes clear the relation between polling policy and the execution time of each application.

5.1 Target Systems

We present a case study with three systems, Dual-AES, Triple-AES, and Quad-AES. Dual-AES, Triple-AES, and Quad-AES consist of two, three, and four AES applications [9], respectively. Each application in the systems is numbered from 1 through 4. In other words, there are four applications named AES1, AES2, AES3, and AES4 as shown in Fig. 5. The four AES applications are identical, and each application encrypts and decrypts data which consist of 16 integers for 1000 times. The AES applications consist of four processes, aes_mainX, encryptX, decryptX and check_resultX (X differs from 1 to 4 depending on the AES application). We assume that each AES application runs on a dedicated processor, and that, in case of Quad-AES, it consists of four processors and the four AES applications run on their own processors.

In this case study, we have explored different mapping solutions on Dual-AES, Triple-AES, and Quad-AES as summarized in Table 1, Table 2, and Table 3, respectively. We have varied mapping of the encryptX and decryptX processes on either a software (SW), a hardware (HW), or a shared hardware (Shared HW). In case of hardware sharing, we have also changed the polling policy. Note that, through the design space exploration, we rewrote the mapping information file only. According to the mapping information, SystemBuilder automatically synthesizes the implementation which is executable on an FPGA. On an average, SystemBuilder took about an hour to synthesize an implementation. To complete the exploration of an AES system, it took less than 24 hours by a single designer. In this work, we used Altera StratixII FPGA board with four Nios II soft-core processors [10] as target architecture. Software processes were cross-compiled and linked with the TOPPERS/FDMP kernel [11], which is a real-time OS for multi-processors, to be executed on the Nios II soft-core processors. Hardware processes were synthesized with a commercial behavioral synthesis tool eXCite [12]. These compilation and synthesis tasks were automatically done by SystemBuilder.

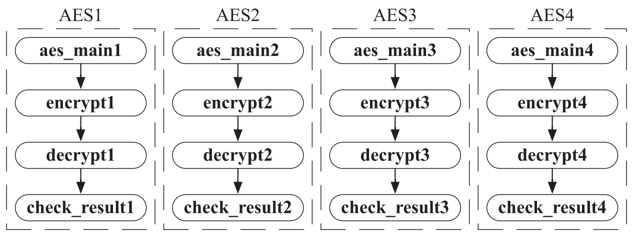


Fig. 5 Construction of Quad-AES.

Table 1 Mapping information and the polling policy of Dual-AES.

Mapping	encrypt1	encrypt2	decrypt1	decrypt2
1	SW	SW	SW	SW
2	HW	HW	SW	SW
3	Shared HW(Priority)		SW	SW
4	Shared HW(Round)		SW	SW
5	SW	SW	HW	HW
6	SW	SW	Shared HW(Priority)	
7	SW	SW	Shared HW(Round)	
8	HW	HW	HW	HW
9	Shared HW(Priority)		HW	HW
10	Shared HW(Round)		HW	HW
11	HW	HW	Shared HW(Priority)	
12	HW	HW	Shared HW(Round)	
13	Shared HW(Priority)		Shared HW(Priority)	
14	Shared HW(Round)		Shared HW(Round)	
15	Shared HW(Round)		Shared HW(Priority)	
16	Shared HW(Priority)		Shared HW(Round)	

5.2 Design Space of Hardward Sharing

In terms of hardware size and execution time on the FPGA, we evaluated the 16 mapping solutions on Dual-AES, and 20 mapping solutions on Triple-AES and Quad-AES as shown in Table 1, Table 2, and Table 3, respectively. Figure 6 shows the hardware size (in #ALUTs) and the execution time (in milli-seconds) of each mapping solution on the three systems. #ALUTs shows the hardware size of processors, peripherals, and processes mapped to hardware and shared hardware. In the figures, the solid lines and the broken lines represent the trade-offs of all mapping solutions and those without hardware sharing, respectively. It is easily observed that the solid lines (with hardware sharing) represent better trade-offs than the broken lines (without hardware sharing) on the three AES systems. In Fig. 6(b), the mapping solution #7 with hardware sharing which is on the solid line, has less hardware size and better performance than the mapping solution #2 without hardware sharing. Hardware sharing, therefore, can bring better area-performance trade-offs.

As mentioned in Sect. 4.3, the designers can explore the number of processes to be shared. In Fig. 6(c), mapping solution #20 has two shared hardware each of which are shared by two processes. Since the mapping solution #20 is on the solid line, it is a candidate of an optimized solution. This result indicates that it is important to explore the number of processes to be shared, which is supported by our hardware sharing method.

5.3 The Reduction of Hardware Size

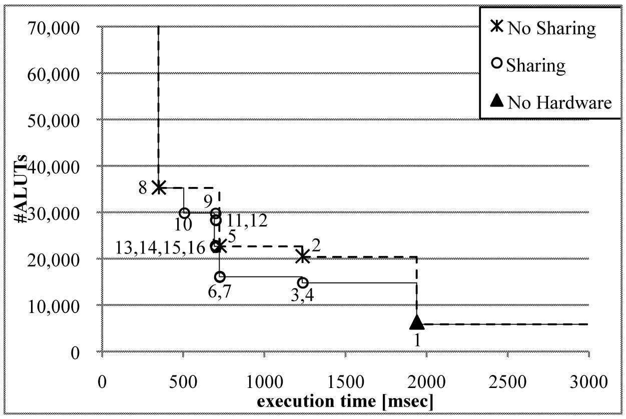
In Fig. 6(a), if we look at mapping solutions #2, #3, and #4 whose difference of mapping is shared or not shared, the hardware size was reduced by 28% thanks to hardware sharing. The same can be said for mapping solutions #5, #6, and #7. Also, if we look at mapping solutions #2, #3, and

Table 2 Mapping information and the polling policy of Triple-AES.

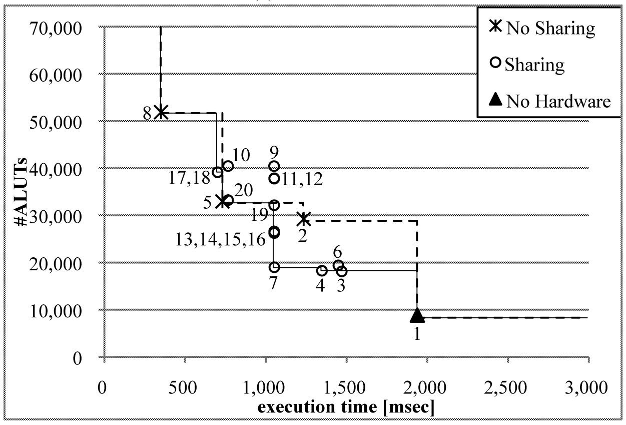
Mapping	encrypt1	encrypt2	encrypt3	decrypt1	decrypt2	decrypt3
1	SW	SW	SW	SW	SW	SW
2	HW	HW	HW	SW	SW	SW
3	Shared HW(Priority)			SW	SW	SW
4	Shared HW(Round)			SW	SW	SW
5	SW	SW	SW	HW	HW	HW
6	SW	SW	SW	Shared HW(Priority)		
7	SW	SW	SW	Shared HW(Round)		
8	HW	HW	HW	HW	HW	HW
9	Shared HW(Priority)			HW	HW	HW
10	Shared HW(Round)			HW	HW	HW
11	HW	HW	HW	Shared HW(Priority)		
12	HW	HW	HW	Shared HW(Round)		
13	Shared HW(Priority)			Shared HW(Priority)		
14	Shared HW(Round)			Shared HW(Round)		
15	Shared HW(Round)			Shared HW(Priority)		
16	Shared HW(Priority)			Shared HW(Round)		
17	Shared HW(Round)		HW	Shared HW(Round)		HW
18	Shared HW(Round)		HW	HW	Shared HW(Round)	
19	Shared HW(Round)		HW	Shared HW(Round)		
20	Shared HW(Round)			Shared HW(Round)		HW

Table 3 Mapping information and the polling policy of Quad-AES.

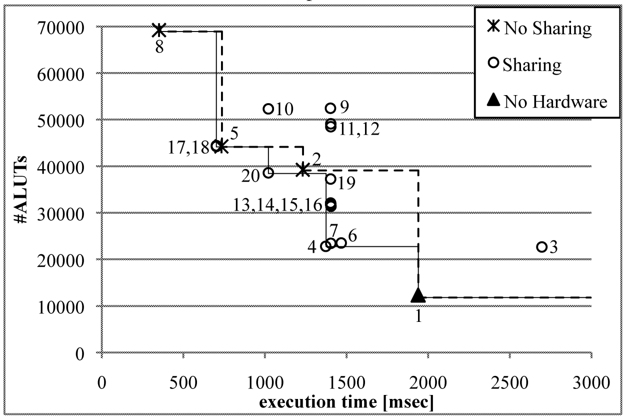
Mapping	encrypt1	encrypt2	encrypt3	encrypt4	decrypt1	decrypt2	decrytp3	decrypt4
1	SW	SW	SW	SW	SW	SW	SW	SW
2	HW	HW	HW	HW	SW	SW	SW	SW
3	Shared HW(Priority)				SW	SW	SW	SW
4	Shared HW(Round)				SW	SW	SW	SW
5	SW	SW	SW	SW	HW	HW	HW	HW
6	SW	SW	SW	SW	Shared HW(Priority)			
7	SW	SW	SW	SW	Shared HW(Round)			
8	HW	HW	HW	HW	HW	HW	HW	HW
9	Shared HW(Priority)				HW	HW	HW	HW
10	Shared HW(Round)				HW	HW	HW	HW
11	HW	HW	HW	HW	Shared HW(Priority)			
12	HW	HW	HW	HW	Shared HW(Round)			
13	Shared HW(Priority)				Shared HW(Priority)			
14	Shared HW(Round)				Shared HW(Round)			
15	Shared HW(Round)				Shared HW(Priority)			
16	Shared HW(Priority)				Shared HW(Round)			
17	Shared HW(Round)		Shared HW(Round)		Shared HW(Round)		Shared HW(Round)	
18	Shared HW(Round)		Shared HW(Round)		Shared HW(1&4)(Round)	Shared HW(Round)		Shared HW(1&4)(Round)
19	Shared HW(Round)		Shared HW(Round)		Shared HW(Round)			
20	Shared HW(Round)				Shared HW(Round)		Shared HW(Round)	



(a) Dual-AES



(b) Triple-AES



(c) Quad-AES

Fig. 6 Trade-offs between performance and hardware size.

#4 in Fig. 6(b) and Fig. 6(c), the hardware size was reduced by 37% and 42%, respectively. In the figures, the solutions marked with a circle which has shared hardware have the least hardware size among the mapping solutions except the solution #1 marked with black triangle which has no hardware. Thus, our hardware sharing method is effective to reduce the hardware size.

In the three systems, the ratio of hardware size on mapping solutions #3 and #4 is almost the same, and the only difference between mapping solutions #3 and #4 is the polling policy. The same can be said for mapping solutions #6 and #7 in the three systems. In other word, the polling policy did not influence reduction of hardware size if the processes have the same mapping information. Our hardware sharing method, therefore, can reduce the hardware size with both priority-based polling and round-robin one.

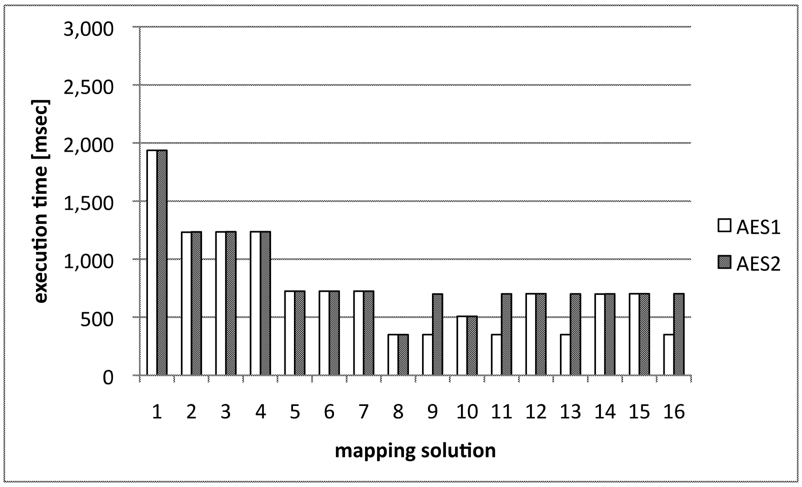
5.4 The Execution Time with Polling Policy

In order to make clear the relation between the polling policy and the execution time of each application, we measured the execution time of each application in the solutions as shown in Table 1, Table 2, and Table 3. Figure 7 shows the execution time of each AES application on the three systems. The polling policy of mapping solutions #4, #7, #10, #12, and #14 on the three systems is only round-robin polling. Also, mapping solutions #18, #19, and #20 on Triple-AES and Quad-AES use only round-robin polling. In these mapping solutions, the execution time of each application was averaged. This result indicates that the wrapper with round-robin polling equally selected the application running on the shared process. Mapping solutions #9, #11, and #13 on the three systems shows typical results of priority-based polling. In these mapping solutions, AES1 which has the highest priority is completed at first. Then AES2, AES3, and AES4 which have the second, the third, and the lowest priority, respectively, were completed in the order of the priorities. In mapping solution #6 with priority-based polling, however, AES1 and AES2 on the three systems were completed at almost the same time. Since encryptX processes in mapping solution #6 were mapped to software, the execution of encryptX processes was not faster than that of the shared process. In particular, when the wrapper started the polling, encrypt1 process was running while encrypt2 process had written the data to the channel. Thus, the wrapper selected AES2 instead of AES1 which the previous process of the shared process was running. As a result, AES1 and AES2 were selected by round-robin manner, and they were completed at the same time. Then, AES3 and AES4 in mapping solution #6 were completed after the completion of AES1 and AES2 in the systems.

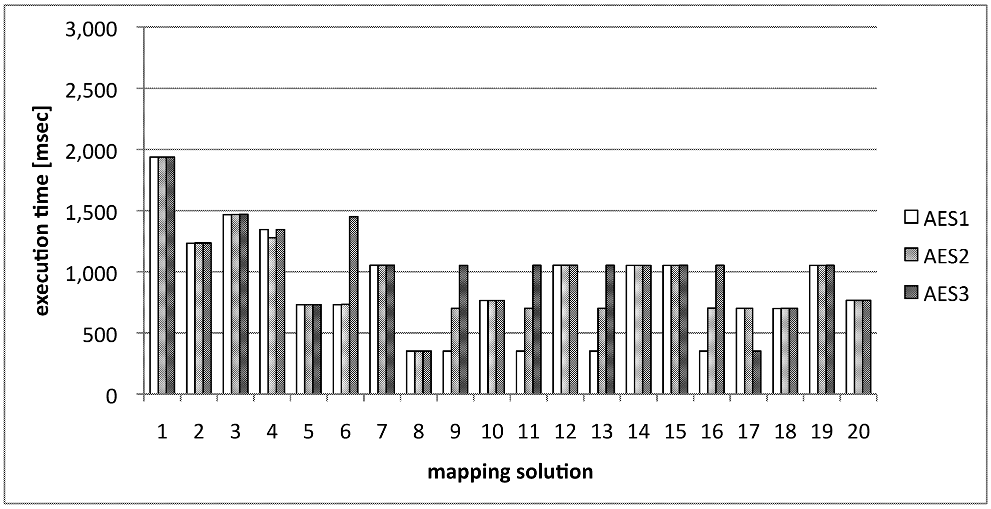
Comparing mapping solution #9 with #10 in the three systems, mapping solution #9 with priority-based polling took longer than mapping solution #10 with round-robin one in the total execution time. In the case of that the deadline of a particular application is very strict, priority-based polling is more suitable than round-robin one. On the other hand, in the case of that the execution times of all application should be averaged, round-robin polling is more suitable.

6. Conclusion

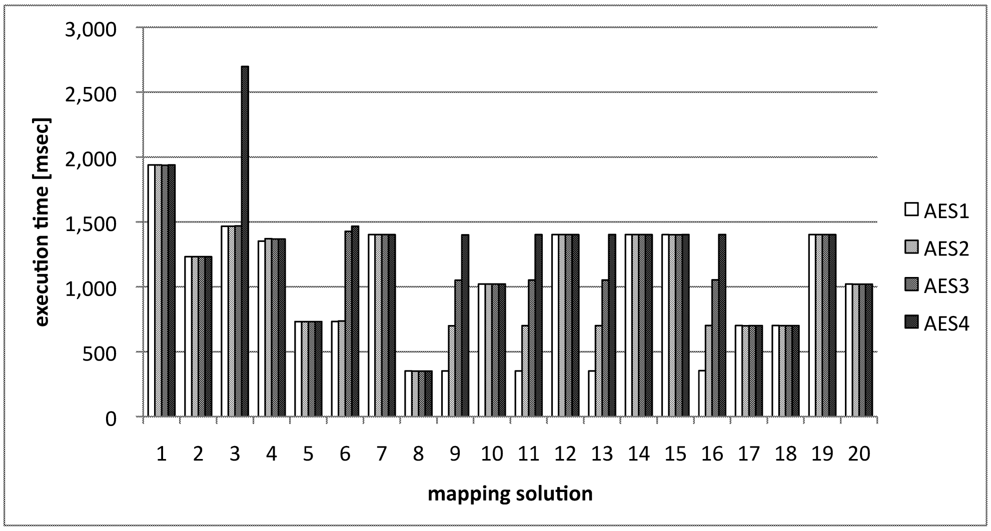
In this paper, we have proposed an automatic communication synthesis method for hardware sharing and implemented it in our system-level design tool named SystemBuilder. With SystemBuilder, the designers can explore wider design space including hardware sharing in short time



(a) Dual-AES



(b) Triple-AES



(c) Quad-AES

Fig. 7 Execution time of AES applications.

since SystemBuilder automatically synthesizes communication for hardware sharing with only rewriting the mapping information. We have conducted a case study on hardware sharing with AES applications. The case study demonstrated that hardware sharing brought better area-performance trade-offs and a wider design space. The case study also demonstrated reducing hardware size while keeping the performance. In future, we plan to conduct additional case studies with more complicated applications.

Acknowledgments

This work was in part supported by STARC (Semiconductor Technology Academic Research Center).

References

[1] A. Gerstlauer, C. Haubelt, A.D. Pimentel, T.P. Stefanov, D.D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.28, no.10, pp.1517–1530, Oct. 2009.

[2] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D.D. Gajski, "System-on-chip environment: A SpecC-based framework for heterogeneous MPSoC design," EURASIP Journal on Embedded Systems, vol.2008, pp.1–13, Jan. 2008.

[3] D.D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, SpecC: Specification language and design methodology, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[4] A.D. Pimentel, "The Artemis workbench for system-level performance evaluation of embedded systems," International Journal of Embedded Systems, vol.3, no.3, pp.181–196, Sept. 2008.

[5] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y.P. Joo, "PeaCE: A hardware-software codesign environment for multimedia embedded systems," ACM Trans. Des. Autom. Electron. Syst., vol.12, no.3, pp.1–25, Aug. 2007.

[6] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metoropolis: An integrated electronic system design environment," Computer, vol.36, no.4, pp.45–52, April 2003.

[7] S. Mahadevan, K. Virk, and J. Madsen, "ARTS: A SystemC-based framework for multiprocessor systems-on-chip modelling," Design Automation for Embedded Systems, vol.11, no.4, pp.285–311, Dec. 2007.

[8] S. Honda, H. Tomiyama, and H. Takada, "RTOS and codesign toolkit for multiprocessor systems-on-chip," Proc. 12th ASP-DAC, pp.336–341, Jan. 2007.

[9] Y. Hara, H. Tomiyama, S. Honda, and H. Takada, "Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis," J. Information Processing, vol.17, pp.242–254, Oct. 2009.

[10] Altera Corporation, http://www.altera.com/

[11] TOPPERS Project, http://www.toppers.jp/en/index.html

[12] Y Explorations Inc., http://www.yxi.com/

Yuki Ando received the B.E. degree in information engineering from Nagoya University in 2009. Currently he is a M.S. candidate at the Information Science from Nagoya University. His research interests include system-level design automation and embedded systems.

Seiya Shibata received his B.E. degree in information engineering and M.S. degree in Information Science from Nagoya University in 2007, and 2009, respectively. Currently he is a Ph.D. candidate at the Graduate School of Information Science, Nagoya University. His research interests include system-level design and embedded systems.

Shinya Honda received his Ph.D. degree in the Department of Electronic and Information Engineering, Toyohashi University of Technology in 2005. From 2004 to 2006, he was a researcher at the Nagoya University Extension Course for Embedded Software Specialists. In 2006, he joined the Center for Embedded Computing Systems, Nagoya University, as an assistant professor, where he is now an associate professor. His research interests include system-level design automation and real-time operating systems. He received the best paper award from IPSJ in 2003. He is a member of IPSJ and JSSST.

Hiroyuki Tomiyama received his B.E., M.E. and D.E. degrees in computer science from Kyushu University in 1994, 1996 and 1999, respectively. From 1999 to 2001, he was a visiting postdoctoral researcher with the Center of Embedded Computer Systems, University of California, Irvine. From 2001 to 2003, he was a researcher at Institute of Systems & Information Technologies/KYUSHU. In 2003, he joined the Graduate School of Information Science, Nagoya University as an assistant professor, and became an associate professor in 2004. In 2010, he joined the College of Science and Engineering, Ritsumeikan University as a full professor. His research interests include system-level design automation, architectures and compilers for embedded systems and systems-onchip. He currently serves as an associate editor-in-chief of IPSJ Transactions on System LSI Design Methodology, an associate editor of IEEE Embedded Systems Letters, and an editorial board member of International Journal on Embedded Systems. He has also served on the organizing and program committees of several premier conferences including ICCAD, ASP-DAC, DATE, CODES+ISSS, and so on. He is a member of ACM, IEEE and IPSJ.

Hiroaki Takada is a Professor at the Department of Information Engineering, the Graduate School of Information Science, Nagoya University. He is also the Executive Director of the Center for Embedded Computing Systems (NCES). He received his Ph.D. degree in Information Science from University of Tokyo in 1996. He was a Research Associate at University of Tokyo from 1989 to 1997, and was a Lecturer and then an Associate Professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a member of ACM, IEEE, IPSJ, and JSSST.