Vol.2010-ARC-187 No.15
Vol.2010-EMB-15 No.15
2010/1/29

# 設計空間探索における ハードウェア共有用通信の自動合成

安 藤 友 樹[†1] 柴 田 誠 也[†1,†2] 本 田 晋 也[†1]
冨 山 宏 之[†1] 高 田 広 章[†1]

マルチプロセッサ組込みシステムの設計空間探索における，ハードウェアの共有を実現する通信実装の自動合成について述べる．我々は過去の研究において，高い抽象度で記述されたシステムから実装記述を合成するシステムレベル設計環境であるSystemBuilderを開発した．本研究では，SystemBuilderを拡張し，システム内の複数のアプリケーション間でハードウェアの共有を実現する合成を可能とした．これにより，設計者が自らハードウェアの共有を行うための通信を設計することなく，ハードウェアの共有を含めた，より広い設計空間を容易に探索可能となる．設計事例により，ハードウェア共有を使用した設計空間探索の効果を示す．

# Automatic Synthesis of Hardware Sharing Communication for Design Space Exploration

Yuki Ando,[†1] Seiya Shibata,[†1,†2] Shinya Honda,[†1]
Hiroyuki Tomiyama [†1] and Hiroaki Takada[†1]

We present a hardware sharing method for design space exploration of multiprocessor embedded systems. We had developed a system-level design tool named SystemBuilder which automatically synthesizes target implementations of a system from a functional description. We extended SystemBuilder so that it can automatically synthesize a target implementation which shares a hardware module among different applications. Designers, therefore, only need to design applications of the system, and can easily explore design space including hardware sharing by automatic synthesis of SystemBuilder. A case study shows the effectiveness of the hardware sharing on design space exploration.

†1 名古屋大学　大学院情報科学研究科
Graduate School of Information Science, Nagoya University
†2 日本学術振興会特別研究員 DC
Research Fellow of the Japan Society for the Promotion of Science

## 1. Introduction

System-level design has been proposed in order to design complex embedded systems. In the system-level design, designers design a system at high level of abstraction. They start from describing functionalities of the system as processes and channels, which indicate computations and communications among processes, respectively. Then, they decide mapping of processes to various Processing Elements (PEs) including CPUs and dedicated hardware modules. In order to support such system-level design, a number of tools have been proposed in the past[1)2)3)4)]. The tools have ability to convert processes and channels into compilable software program and synthesizable RTL circuits depending on the mapping decision.

These days, embedded systems consist of multiple applications (such as music and movie players, email, and web browsing), and in many cases the applications include common functionalities (such as DCT, IDCT, encryption, and decryption). In order to optimize the cost/performance efficiency, the common functionalities are often implemented in dedicated hardware modules which are shared by the applications. However, such coarse-grained hardware sharing is not supported by most of the existing system-level design tools. Many tools assume single-application systems. Some tools assume multiple applications, but they do not allow to map processes in different applications onto a single hardware module. Even if allowed, they do not automatically synthesize interface circuitry which realizes mutually exclusive accesses to the shared hardware modules.

In this work, our system-level design tool named SystemBuilder has been extended so that it supports process-level hardware sharing. With SystemBuilder, designers can map processes in different applications onto a single hardware module. Then, SystemBuilder can automatically synthesize communications for the hardware module which are shared by the multiple applications. Since the applications may run concurrently, the interface circuit generated by SystemBuilder realizes mutually exclusive accesses to the shared hardware.

This paper is organized as follows. Section 2 explains a brief overview of Sys-

 © 2010 Information Processing Society of Japan

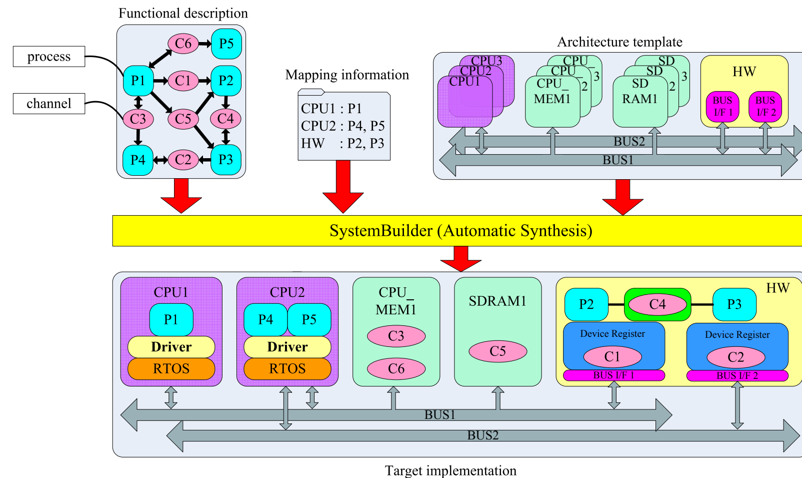



**Fig.1** Overview of SystemBuilder.

temBuilder. Section 3 presents the detail of communication synthesis for hardware sharing. Section 4 shows the effectiveness of hardware sharing through a case study on Advanced Encryption Standard (AES) system, and Section 5 concludes this paper.

## 2. SystemBuilder

In this section, we show a brief overview of SystemBuilder to make this paper self contained. Please refer 5) for the detail of SystemBuilder.

**Fig.1** shows the mapping and synthesis overview of SystemBuilder. SystemBuilder takes functional description, an architecture template and mapping information as input, and generates target implementations of the system. The functional description and the architecture template represent system functionalities and target platforms, respectively. The functional description consists of a set of processes running concurrently and channels representing communications among processes. Processes are written in the C language with communication APIs as interfaces to channels. A process is implemented as either a software task on a Real-time OS (RTOS) or a hardware module with a single FSM depending on SW/HW partitioning described in mapping information.

SystemBuilder provides abstract communications as channels and synthesizes implementation of them. One of the features of SystemBuilder is automatic synthesis of communications among the processes. Channels are classified into two general groups, asynchronous and synchronous. Asynchronous channels are used to transfer data among the processes. Synchronous channels are mainly used between two processes to notify start/end events of execution to synchronize them. Depending on the types of channels and mapping information on software/hardware partitioning, communication APIs used in each process description are converted to interface programs and logics to communicate with each other through channels.

## 3. Automatic Communication Synthesis with Hardware Sharing

### 3.1 The Design Flow for Hardware Sharing

It is assumed that a system consists of more than one application. In **Fig.2**, there are two applications. Designers first design applications in the system independently as shown in **Fig.2(a)**. Without synthesis option on hardware sharing, SystemBuilder generates the system implementation as shown in **Fig.2(b)**. With hardware sharing option, SystemBuilder converts an input description (Fig.2(a)) to an internal description (**Fig.2(c)**). In order to turn on hardware sharing option, processes that designers want to share among several applications must satisfy following requirements;

- All channels of the processes have same characteristics such as a synchronous channel.
- The processes have same functionality and are mapped to hardware.

In Fig.2, since both processes P_B and P_Y have same functionality and same channels, and they are mapped to hardware, they can be shared. P_S whose functionality is the same as both P_B and P_Y is shared by two applications.

In our hardware sharing method, channels in both application1 and application2 remain if P_S is shared by two applications as shown in Fig.2(c). Then SystemBuilder automatically generates the system as shown in **Fig.2(d)** from



© 2010 Information Processing Society of Japan

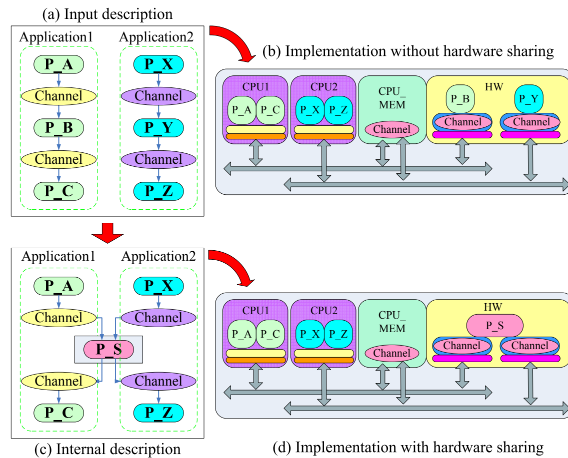



**Fig.2** Design Flow for Hardware Sharing.

internal description (Fig.2(c)).

The hardware cost of the system (Fig.2(d)) will be less than that of the system (Fig.2(b)) since two applications share a hardware module in the system (Fig.2(d)). On the other hand, the performance of the system (Fig.2(d)) may be worse than that of the system (Fig.2(b)) since one application is blocked to use a shared hardware module while the shared hardware module is used by the other application. Therefore, there is a trade-off between hardware cost and performance of the system.

SystemBuilder automatically completes the synthesis flow of hardware sharing in Fig.2 with a sharing option in the mapping information. Since designers only need to turn on the option in the mapping information to share the hardware, designers will be able to explorer wider design space than that without hardware sharing in short time. Note that designers can share hardware among more than two applications although Fig.2 only shows two applications.

### 3.2 Implementation of Communication for Hardware Sharing

A shared process starts its execution by a start event of synchronous communication sent by the preceding process in the application. While the shared process

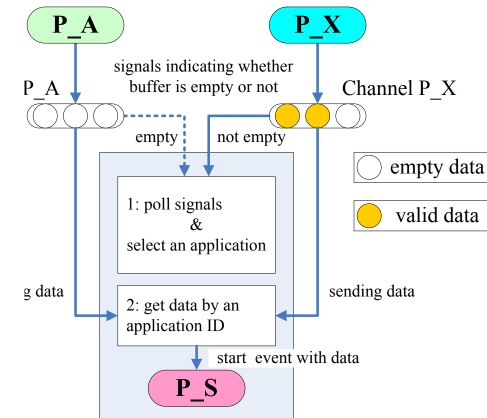



**Fig.3** Detail of the Wrapper Generated by SystemBuilder.

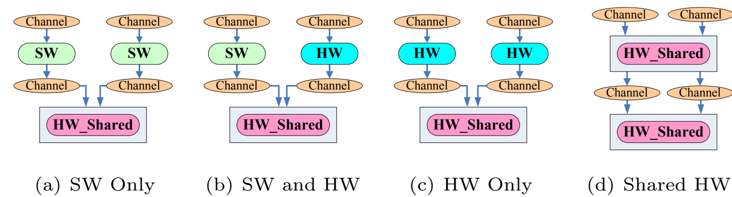
is used by an application, other applications which access the shared process are forced to wait. In this way, the shared process is used by multiple applications exclusively. In our method, each application writes data to its own channels. Since applications which use the shared process have their own channels and applications use the shared process exclusively, there is no data conflict in the shared process. Instead, the shared process needs to select an application from (to) which the shared process should read (write) data.

SystemBuilder automatically adds a wrapper to a shared process, which realizes mutual exclusion and identifies the channel to be accessed as shown in **Fig.3**. Also, SystemBuilder adds a signal to channels connected to the shared process, which indicates if the buffer in the channel is empty or not. Then, the wrapper works as follows.

First, the wrapper polls the signals from the channels in order to select an application which can use the shared process. SystemBuilder supports two types of a polling, a priority-based polling and a round-robin one. With a priority-based polling, every time the shared process completes its execution, the channel of the highest priority application is checked at first. If the channel's signal indicates empty, the lower priority application will be checked. With a round-robin polling,



© 2010 Information Processing Society of Japan




(a) SW Only (b) SW and HW (c) HW Only (d) Shared HW

**Fig.4** Mapping of Processes with Hardware Sharing.

the channels are checked in a round-robin manner. This polling continues until non-empty signal is found. Designers define the polling type and priorities of applications in mapping information.

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

The shared process may communicate with other processes not only at starting and finishing times of the process but also during its execution. Every time the shared process communicates with another process, the wrapper accesses the channel of the selected application.

### 3.3 Mapping of Processes with Hardware Sharing

In our method, there is no limitation on mapping of processes which are connected to the shared processes. **Fig.4** shows four patterns of process mapping with hardware sharing supported by SystemBuilder. Our method can be applied to other system-level design tools if they provide a synchronous communication.

## 4. A Case Study

In this section, a case study on three AES systems is presented to show effectiveness of design space exploration with hardware sharing. Each system consists of two, three, or four AES applications. Each AES application in the system is numberd from 1 through 4. AES1 is the highest priority, and the priorities of AES2, AES3, and AES4 are lowered in order of AES2, AES3, and AES4. The AES application is selected from CHStone Benchmark Suite[6]. Each AES application consists of 4 processes, aes_mainX, encryptX, decryptX, and check_resultX (X is 1, 2, 3, or 4). In this case study, software processes are compiled and linked with TOPPERS/FDMP kernel[7] which is a Real-Time OS for multi-processors. Hardware processes are converted to RTL descriptions by a commercial HLS tool, YXI eXCite3.2c[8]. Hardware processes in RTL are synthesized by Quartus II 8.1 logic synthesizer and implemented on Altera Stratix II FPGA board with four Nios II soft-core processors[9]. Since each application is allocated to its own processor, they can run in parallel. The performance of the AES system was measured by the time where each application in the systems completes encryption and decryption of 16 integer data for 1000 times. **Table 1** shows 14 designs with their process mapping and a polling type.

**Fig.5** shows the execution time of each application on three AES systems. With a priority-based polling, a higher priority application mostly completes its execution earlier than lower priority applications. The execution time of AES1 and AES2 of design ♯6, however, are same in both **Fig.5(b)** and **Fig.5(c)**, though the design uses a priority-based polling. Since the shared hardware module ended its execution during running of encrypt1 that was mapped to software, AES2 was able to use the shared hardware module even its priority was lower than AES1. Therefore, encrypt1 and encrypt2 were able to run in parallel and their execution times resulted in almost same. By contrast with a priority-based polling, the execution times of each application in design using a round-robin polling were averaged. Since a round-robin polling enables systems to use a shared hardware module effectively, total execution time of the systems with a round-robin polling was earlier or equal to that of the systems with a priority-based polling. Therefore, designers can select a priority-based polling in order to end the highest priority process at first, and they can select a round-robin polling in order to shorten the entire execution time.

**Fig.6** shows the trade-offs between performance and ♯ALUTs on three AES systems. ♯ALUTs shows only hardware area of processes mapped to hardware and it does not include processors and peripherals. In **Fig.6(a)**, design ♯1, ♯6, ♯7, and ♯8 were better balanced between the performance and ♯ALUTs, and two of them turned on the hardware sharing option. In **Fig.6(b)**, design ♯7


© 2010 Information Processing Society of Japan

**Table 1** Mapping and the Polling Type of AES Systems.

| ♯design | encrypt1 | encrypt2 | encrypt3 | encrypt4 | decrypt1 | decrypt2 | decrytp3 | decrypt4 | polling |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SW | SW | SW | SW | SW | SW | SW | SW | — |
| 2 | HW | HW | HW | HW | SW | SW | SW | SW | — |
| 3 | HW_Share | | | | SW | SW | SW | SW | priority |
| 4 | HW_Share | | | | SW | SW | SW | SW | round-robin |
| 5 | SW | SW | SW | SW | HW | HW | HW | HW | — |
| 6 | SW | SW | SW | SW | HW_Share | | | | priority |
| 7 | SW | SW | SW | SW | HW_Share | | | | round-robin |
| 8 | HW | HW | HW | HW | HW | HW | HW | HW | — |
| 9 | HW | HW | HW | HW | HW_Share | | | | priority |
| 10 | HW | HW | HW | HW | HW_Share | | | | round-robin |
| 11 | HW_Share | | | | HW | HW | HW | HW | priority |
| 12 | HW_Share | | | | HW | HW | HW | HW | round-robin |
| 13 | HW_Share | | | | HW_Share | | | | priority |
| 14 | HW_Share | | | | HW_Share | | | | round-robin |

was better balanced as well as that in Fig.6(a). Unlike design ♯7 in Fig.6(a) and Fig.6(b), design ♯7 in **Fig.6(c)** was not balanced since the shared process became the bottle neck of the system. The performance of the system will be lowered if several applications share a hardware module, however, hardware sharing can expand the design space. We conclude hardware sharing enabled to explore better cost/performance trade-offs.

**Table 2** shows sizes of hardware area and ratio of them in design ♯8 which turned off the hardware sharing option and design ♯13 which turned on the hardware sharing option. The size of hardware area of two AESs, three AESs, and four AESs were reduced by 43%, 59%, and 65% at maximum, respectively, with hardware sharing option. Therefore, hardware sharing is effective to reduce the size of hardware area in this case study.

## 5. Conclusion

This paper proposed the hardware sharing method with our system-level design tool named SystemBuilder. With SystemBuilder, designers only need to change mapping of processes onto either software or hardware in order to explore design space. Since SystemBuilder can automatically add the wrapper to the shared process, designers only need to turn on the sharing option in mapping information in order to share the hardware. Hardware sharing will bring designers wider design spaces and chance to reduce the hardware size.

**Table 2** Comparation of ♯ALUTs.

| System | ♯design | ♯ALUTs | Ratio |
|---|---|---|---|
| Two AESs | 8 | 28985 | 1 |
| | 13 | 16432 | 0.57 |
| Three AESs | 8 | 43015 | 1 |
| | 13 | 17732 | 0.41 |
| Four AESs | 8 | 56864 | 1 |
| | 13 | 19747 | 0.35 |

We conducted a case study of hardware sharing on three AES systems which


© 2010 Information Processing Society of Japan

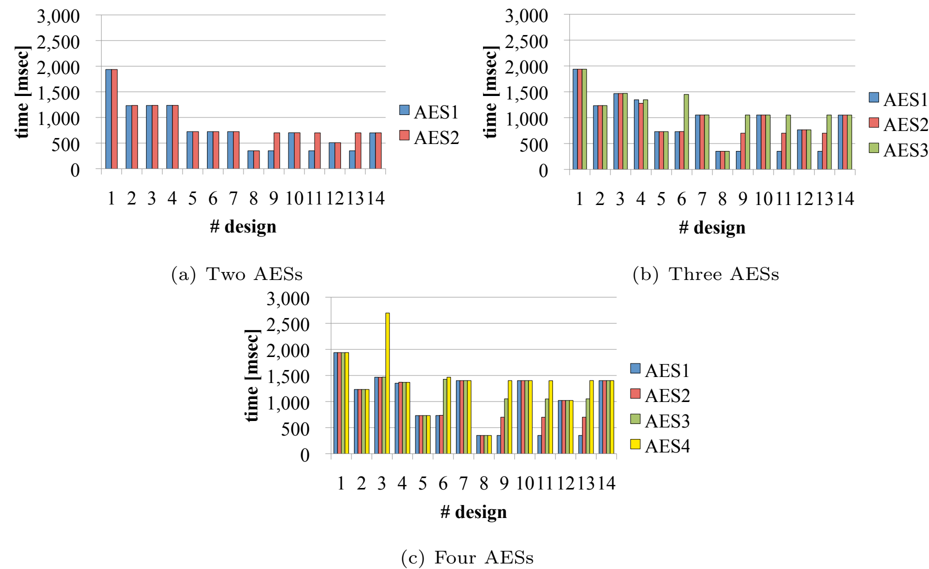



**Fig.5** Execution Time of AES Encryption and Decryption Systems.

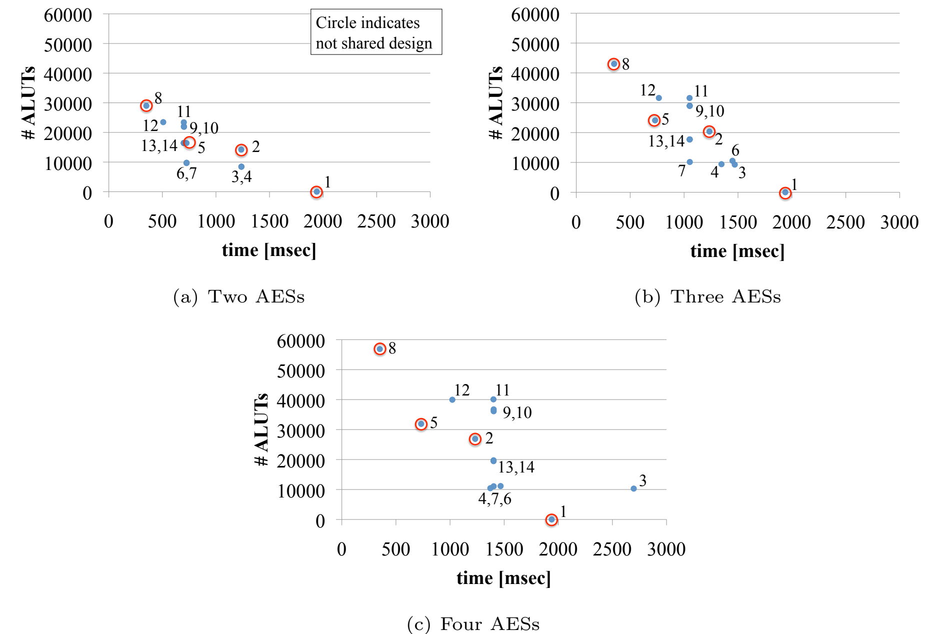



**Fig.6** Trade-offs for Performance and Hardware Size.

have two, three, or four AES applications. In our case study, hardware sharing expanded the design space and reduced hardware size by 43%, 59% and 65% at maximum with two, three, and four AES applications, respectively.

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

## References

1) Dömer, R., Gerstlauer, A., Peng, J., Shin, D., Cai, L., Yu, H., Abdi, S. and Gajski, D. D.: System-on-Chip Environment: A SpecC-Based Framework for Heterogeneous MPSoC Design, *EURASIP Journal on Embedded Systems*, Vol. 2008, pp. 1–13 (2008).
2) Pimentel, A. D.: The Artemis workbench for system-level performance evaluation of embedded systems, *International Journal of Embedded Systems*, Vol.3, No.3, pp. 181–196 (2008).
3) Ha, S., Kim, S., Lee, C., Yi, Y., Kwon, S. and Joo, Y. P.: PeaCE: A Hardware-Software Codesign Environment for Multimedia Embedded Systems, *ACM Trans. Design Automation of Electronic Systems*, Vol.12, No.3, pp.1–25 (2007).
4) Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C. and SangiovanniVincentelli, A.: Metoropolis: An Integrated Electronic System Design Environment, *Computer*, Vol.36, No.4, pp.45–52 (2003).
5) Honda. S., Tomiyama, H. and Takada, H.: RTOS and Codesign Toolkit for Multiprocessor Systems-on-Chip, *ASP-DAC*, pp.336–341 (2007).
6) Hara, Y., Tomiyama, H., Honda, S., Takada, H. and Ishii, K.: CHStone: Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis, *Journal of Information Processing*, Vol. 17, pp. 242–254 (2009).
7) TOPPERS Project, http://www.toppers.jp/en/index.html.
8) Y Explorations Inc., http://www.yxi.com/.
9) Altera Corporation, http://www.altera.com/.



© 2010 Information Processing Society of Japan