IPSJ Transactions on System LSI Design Methodology Vol.8 95–99 (Aug. 2015)

[DOI: 10.2197/ipsjtsldm.8.95]

Short Paper

Automatic Synthesis of Inter-heterogeneous-processor Communication for Programmable System-on-chip

Yuki Ando[1,a)] Yukihito Ishida[1] Shinya Honda[1] Hiroaki Takada[1] Masato Edahiro[1]

Received: December 4, 2014, Accepted: February 13, 2015, Released: August 1, 2015

Abstract: This paper introduces an automatic synthesis technique and tool to implement inter-heterogeneous-processor communication for programmable system-on-chips (PSoCs). PSoCs have an ARM-based hard processor system connected to an FPGA fabric. By implementing the soft processors in the FPGA fabric, PSoCs realize heterogeneous multiprocessors. Since the number and type of soft processors are configurable, PSoCs can be various heterogeneous multiprocessors. However, the inter-heterogeneous-processor communications are not supported by single binary operating systems. Proposed method automatically synthesizes the inter-heterogeneous-processor communications at an application layer from a general model description. The case study shows that automatically generated inter-heterogeneous-processor communication exactly runs the system on heterogeneous multiprocessors.

Keywords: heterogeneous multiprocessors, communication synthesis, programmable system-on-chip

1. Introduction

FPGAs have been used widely in a lot of systems because of their flexibility and high performances. Currently, new FPGAs called programmable system-on-chip (PSoC) show up in the market. Examples of PSoCs are Altera SoC [1] and Xilinx ZYNQ [2]. PSoCs have an ARM-based hard processor system [3] connected to an FPGA fabric by a high-bandwidth interconnect. By implementing soft processors into the FPGA fabric, PSoCs can be a heterogeneous multiprocessor (Hetero-MP). **Figure 1** shows an example of Hetero-MP which consists of an ARM-based hard processor and three NiosII soft processors.

There are two types of multiprocessor, a homogeneous multiprocessor (Homo-MP) and a Hetero-MP. A Homo-MP consists of multiple same processors. Since the Homo-MP has same processors, all processors can run a single binary of a general-purpose operating system (GPOS) such as Linux. The GPOS running on the Homo-MP generally takes long time to handle an interrupt processing which requires low latency. To solve this problem, Hetero-MPs have been getting the attention. A Hetero-MP consists of multiple different processors. In addition, different operating systems (OSs) can run on them. For example, a GPOS runs on ARM-based hard processor to execute applications, and a single-processor real-time OS (SP-RTOS) runs on NiosII soft processors to handle processes in low latency (see Fig. 1). Thus, Hetero-MPs potentially have an advantage to improve the real-time property.

The single binary OS running on the Homo-MP provides communication services among the same processors. However, the OSs running on the Hetero-MP do not provide the communication services among the different OSs. Instead of the OSs, the designers usually implement the communications among the different OSs at the application layer. Furthermore, various architectures of the Hetero-MP can be realized in PSoCs. The designers have to modify the implementation of communication among different OSs because each architecture has different number and type of soft processors. Therefore, a standard specification of communication among the different OSs is required.

The Multicore Association [4] has been working to specify standard APIs for Hetero-MPs, which is named Multicore Communication API (MCAPI). MCAPI specifies the APIs for synchronization and its semantics. In particular, MCAPI mentions that a shared memory and interrupt signals raised by other processors (inter-processor interrupt) are needed to realize the communication for Hetero-MPs. Meakin et al. [5] and Matilainen et al. [6] have been implemented a communication architecture based on MCAPI. However, they have to modify their implementations if the configuration of the Hetero-MP changes.

This paper introduces a synthesis tool which automatically implements the communications for Hetero-MP on PSoCs. The tool takes a model description consisting of functionalities and communications. Then, it automatically generates the target implementation including the communication among the different OSs. Since typical Hetero-MPs have inter-processor interrupts and shared memories, the communications among different OSs are realized using them. The case study demonstrates that the system communicates among the different OSs by automatically generated inter-heterogeneous-processor communications.

This paper is organized as follows. Section 2 mentions the Hetero-MP systems on PSoCs. Section 3 explains our communication synthesis tool. Section 4 presents a case study with our synthesis tool, and Section 5 concludes this paper.

[1] Nagoya University, Nagoya, Aichi 464–8603, Japan
[a)] y_ando@ertl.jp

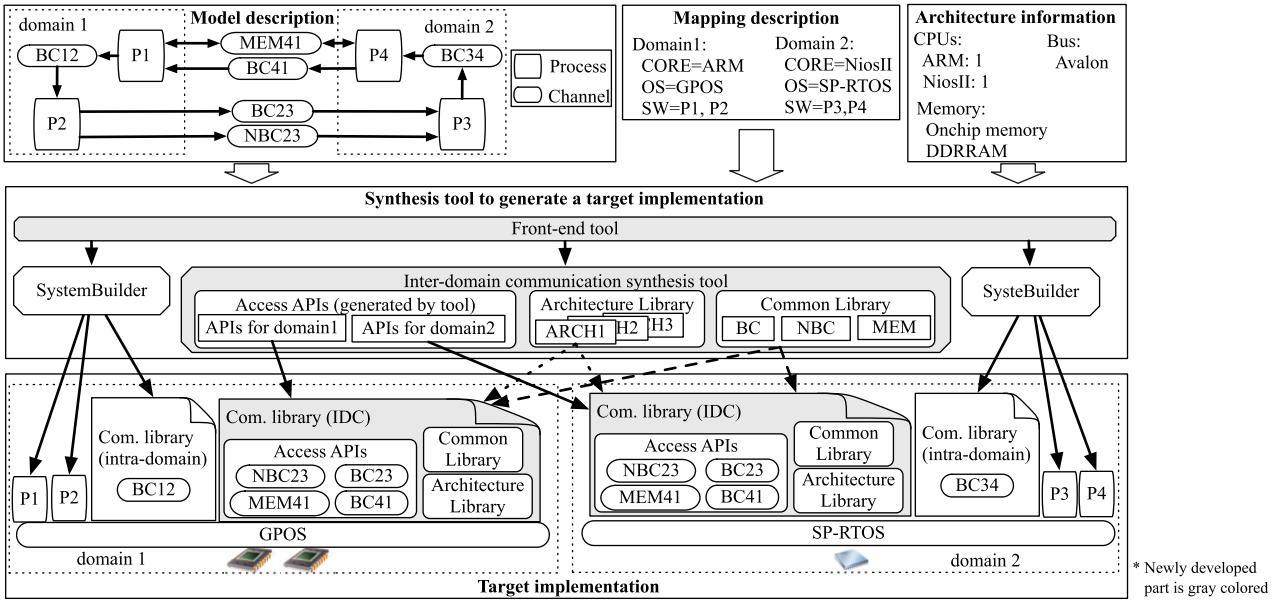

Fig. 2 An automatic communication synthesis tool and an example of target implementation.

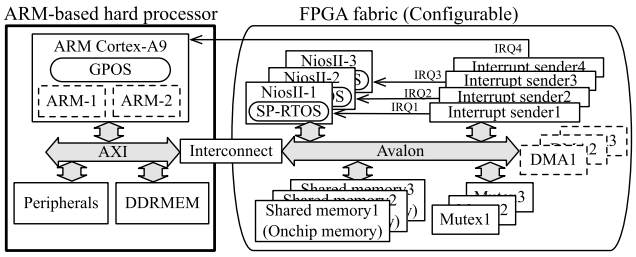

Fig. 1 An example of a Hetero-MP on Altera SoC.

2. Hetero-MP Systems on PSoCs

2.1 The Definition of Hetero-MP System

In this paper, the definition of Hetero-MP system is a system that does not share a single OS's execution binary. For example, a typical Hetero-MP system has several different processors, and it runs different OS on each processor. A system that has same processors but runs different OSs is also a Hetero-MP system in our definition. Furthermore, a system that runs the same OSs but does not share a single OS's execution binary is also a Hetero-MP system. On the other hand, a system using ARM's BIGlittle processor is a Homo-MP system because it shares a single OS's execution binary.

2.2 The Target Architecture of Hetero-MP on Altera SoC

Figure 1 shows the target architecture of Hetero-MP on Altera SoC [1]. Altera SoC integrates an ARM-based hard processor system and an FPGA fabric in a single chip, and they are connected using a high-bandwidth interconnect. The ARM-based hard processor system consists of an ARM Cortex-A9 MPCore processor, peripherals, and DDRMEM, which are connected by AXI bus. The configuration of ARM-based hard processor system is fixed. On the other hand, the FPGA fabric is configurable.

In the FPGA fabric, several NiosII soft processors are connected to interrupt senders, shared memories, mutexes, and other IPs (e.g., DMAs) by the Avalon bus. An interrupt sender is used to raise inter-processor interrupt from other processors. The interrupt numbers of NiosII-1, NiosII-2, NiosII-3, and ARM processor are assigned to 1, 2, 3, and 4, respectively. The shared memories in the FPGA fabric are used in order to communicate among the ARM processor and NiosII soft processors. In addition, mutexes are used to ensure the consistency of data in the shared memory. Since the FPGA fabric is configurable, IPs such as DMA can be optionally allocated.

In this work, the FPGA fabric is assumed to include following elements to realize the communication among the ARM processor and NiosII soft processors.

- A shared memory
- An interrupt sender for each processor
- A mutex

Note that each processor should receive an inter-processor interrupt from other processors by the interrupt sender. This is a suitable assumption for PSoCs because a shared memory, interrupt senders and a mutex can be integrated in the FPGA fabric.

3. Communication Synthesis for Hetero-MP

In this section, we introduce a model description and an automatic communication synthesis tool. **Figure 2** shows an automatic communication synthesis tool and an example of target implementation. Our synthesis tool takes three inputs, a model description, a mapping description, and architecture information. Then, the tool automatically generates a target implementation of the model depending on the mapping information.

Before the explanation of the tool, a "domain" is defined in following section. Then, the details of the tool is described from Section 3.2.

3.1 The Definition of Domain

A domain has to satisfy the following conditions:

- A domain includes at least one processor
- A domain has only a single OS (including BareMetal)

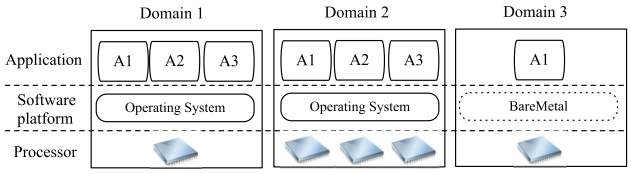

Fig. 3 An example of domains.

- A domain has an unique ID (this is called domain ID)
- A processor does not belong to several domains

Note that a Hetero-MP system is redefined as a system consisting of several domains.

An example of domains is shown in **Fig. 3**. All domains in the figure satisfy the conditions described above. The concept of domain is needed because of Domain 2 in the figure. In domain 2, there are three same processors, and a single OS runs on them. Among the processors in domains 2, there must be inter-processor communications. Such inter-processor communications should be handled by communication services provided by the OS running on three processors because the OS has an ability to efficiently realize such communications. Thus, our method supports only the communications among domains but does not support those inside a domain. Hereafter, the inter-heterogeneous-processor communication among domains is described as "inter-domain communication (IDC)."

3.2 Model Description

The model description is the same as the model description used by SystemBuilder [7], which is related to Kahn Process Network [8]. The model description consists of processes and channels as shown in Fig. 2. Processes represent functionalities of the system. Processes are written in C language with access APIs which are interfaces to channels. Channels represent communications among processes. There are three types of channels:

- Blocking channel (BC)
- Non-Blocking channel (NBC)
- Memory channel (MEM)

BC is a synchronous channel used to synchronize processes. BC behaves as a FIFO buffer. Thus, a sender process has to wait until the buffer has some spaces while a receiver process has to wait until the buffer has data. NBC and MEM are asynchronous channels used to transfer data among processes. NBC and MEM are a shared variable and a shared array, respectively. All processes can access NBC and MEM anytime without being blocked.

These three types of channels are basic communication. Thus, by realizing them in the shared memory, any type of IDC can be realized by the combination of them. Depending on the designers' decisions, processes and channels are allocated to processors and memories, respectively. Since the implementation of the processes and channels is not determined until their allocation are decided, the model is suitable to describe Hetero-MP systems.

3.3 Mapping Description and Architecture Information

A mapping description indicates the type of processors and type of OS for each domain by CORE directive and OS directive, respectively. This also indicates the allocation of processes to the domains by SW directive.

Architecture information describes the architecture of target Hetero-MP. It includes the type and number of processors, the type of bus, and the type and number of memories, and so on. By changing these two, the designers can configure the Hetero-MP systems in PSoCs.

3.4 Synthesis Tool

The synthesis tool takes a model description, a mapping information, and architecture information, and it automatically generates the target implementation. The synthesis tool consists of a front-end tool, SystemBuilder, and IDC synthesis tool. The front-end tool takes the three inputs. Then, it generates files for SystemBuilder and IDC synthesis tool.

3.4.1 SystemBuilder

SystemBuilder is a system-level design toolkit which supports to generate the implementation of intra-domain system. It takes a model description, a mapping description, and architecture information. Then, it automatically generates a target implementation including software, hardware, and Com. library of intra-domain communication among the processes. Since SystemBuilder only supports intra-domain systems, it only generates Com. libraries for BC12 and BC34 (see Fig. 2). SystemBuilder cannot generate the Com. library of IDC.

3.4.2 IDC Synthesis Tool

The IDC synthesis tool automatically generates the Com. library of IDC. In particular, the tool generates the Com. library for MEM41, BC41, BC23, and NBC23 (see Fig. 2). The Com. library of IDC consists of following three elements.

- Access APIs (generated by tool)
- Architecture Library (static)
- Common Library (static)

The tool generates access APIs of channels in order to access data in the shared memory. In addition, the tool automatically allocates the data structure of channels among the domains to the shared memory as shown in **Fig. 4**. For example, a writing access API of MEM takes data and an index to access. The API calculates the absolute address using the allocated base address and index. Then, it stores the data to the shared memory indicated by the absolute address.

Architecture Library includes architecture dependent algorithms and configurations. Examples of such algorithm are a way to raise an inter-processor interrupt and a way to access mutex. In addition, an example of such configuration is the address of the shared memory. An architecture library is prepared for each architecture (ARCH1, ARCH2, and ARCH3 in Fig. 2).

Common Library includes the implementation of NBC, MEM, and BC channels. The implementation of NBC and MEM is simple because they provide the function to access data in the shared memory without blocking. On the other hand, the implementation of BC is complex because BC provides a function to send and receive a message in a manner of FIFO.

The data structures of BCs are allocated to the shared memory. Figure 4 shows an example of the data structure of BCs generated from the model description in Fig. 2. A data structure of BC consists of Data, SendWaitQue, and RecvWaitQue. In addition,

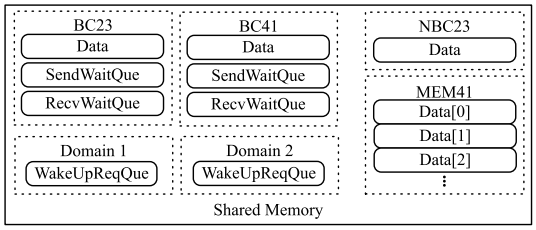

Fig. 4 The data structure of shared memory.

each domain has a WakeUpReqQue in the shared memory which is shared by BCs related to the domain. A mutex is used to ensure the consistency of data in the shared memory.

Data is a region to save the messages in FIFO manner. The size of Data is defined in the model description, and it should be greater than or equal to zero. If the size of Data is zero, BC behaves as a handshake communication.

SendWaitQue keeps the information of processes waiting for sending a message while RecvWaitQue keeps those waiting for receiving a message. Information of processes includes process ID and domain ID. The information is kept in FIFO manner in each queue.

WakeUpReqQue keeps the process ID in FIFO manner in order to wake up the waiting processes of related domain. Since each domain has its own WakeUpReqQue, each domain can wake up the waiting processes according to the related WakeUpReqQue when the domain receives an interrupt. The combination of SendWaitQue/RecvWaitQue and WakeUpReqQue realizes IDC of BC.

3.4.3 Target Implementation

Currently, the synthesis tool supports Altera SoC. Figure 2 shows an example of target implementation. In this example, processes P1 and P2 run on GPOS, and processes P3 and P4 run on SP-RTOS. With the synthesis tool, the designers can get different implementations by only changing the mapping description. Furthermore, the designers can implement the system on different architectures by using different architecture information. Therefore, they can efficiently evaluate and compare the performances of different implementations of Hetero-MP systems.

4. Experimental Results

4.1 Target Application

This section shows a case study to design an AES encryption and decryption application (AES system). We used Altera SoC [1] as a target which has three NiosII soft processors as shown in Fig. 1. We used three OSs, TOPPERS/ASP, TOPPERS/FMP [9], and Linux. TOPPERS/ASP and TOPPERS/FMP are SP-RTOS and MP-RTOS, respectively, and Linux is GPOS.

Figure 5 shows the model description of AES system. AES system consists of three processes named TOP, ENC, and DEC. Process TOP sets the data to encrypt, the keys for encryption and decryption, and a type of data. Then, it starts the process ENC which encrypts the data and sends encrypted data to process DEC. After that, process ENC starts the process DEC which decrypts the encrypted data. Finally, process DEC sends decrypted data to process TOP. In this case study, we allocated ENC and DEC into the same domain in order to make the system simple. Thus, channels named DEC_DATA and DEC_ST are not IDC.

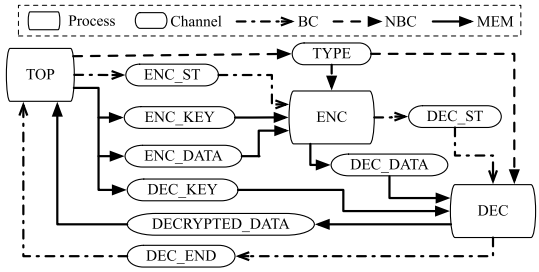

Fig. 5 Model description of AES encryption and decryption application.

Table 1 Execution time of AES system on NiosII processors.

pattern	NiosII-1	NiosII-2	NiosII-3	time(s)
P1-1	–	D1:SP-RTOS SW=TOP	D2:SP-RTOS SW=ENC,DEC	6.470
P1-2	D1:MP-RTOS SW=TOP		D2:SP-RTOS SW=ENC,DEC	6.479
P1-3	D1:MP-RTOS SW=ENC,DEC		D2:SP-RTOS SW=TOP	6.559

Table 2 Execution time of AES system on ARM multicore processors with a NiosII processor.

pattern	ARM-1	ARM-2	NiosII	time(s)
P2-1	D1:MP-RTOS SW=TOP		D2:SP-RTOS SW=ENC,DEC	2.182
P2-2	D1:MP-RTOS SW=ENC,DEC		D2:SP-RTOS SW=TOP	0.258
P2-3	D1:GPOS SW=TOP		D2:SP-RTOS SW=ENC,DEC	2.197
P2-4	D1:GPOS SW=ENC,DEC		D2:SP-RTOS SW=TOP	0.717

4.2 Evaluation of Execution Time

We generated three patterns of AES system on three NiosII processors as shown in **Table 1**. In the table, D1 and D2 indicate domain 1 and domain 2, respectively, and SW indicates processes allocated to the domain. We only changed the mapping description to generate three patterns. All three patterns were correctly executed. Since all processors are NiosII, the execution times of three patterns are almost same.

We also generated four patterns of AES system on an ARM dual-core processor with a NiosII processor as shown in **Table 2**. Even ARM processor was used, we only needed to change the mapping description in order to generate these patterns. All four patterns were correctly executed. Since ARM processor runs much faster than NiosII does, the execution time of P2-2 and P2-4 were faster than those of P2-1 and P2-3.

4.3 The Design Efficiency

Using our method, about 30 lines of mapping description was only changed to replace the target architectures. The synthesis tool automatically generated the Com. library of IDC which consists of following files for each domain.

- Common and architecture library: about 400 lines of C code
- Access APIs: about 80 lines of C code

Without the synthesis tool, the designers had to implement the IDC. Even they can re-implement the IDC from a previous implementation, they still need to change about 80 lines of C code on this case study. This costs longer time than our method does. In fact, we needed less than one hour to generate and evaluate above implementations. With that, designers can efficiently de-

sign a system on Hetero-MP SoCs with our method.

In addition, our method has an advantage to replace the target architecture. Changing the mapping information is only needed to replace the target architecture. As shown in Section 4.2, the designer replaced the architecture without rewriting the model description in this case study. Therefore, our method is effective in order to compare the several Hetero-MP SoCs in short time.

5. Conclusion

This paper introduces an automatic communication synthesis technique for Hetero-MP systems in PSoCs. We focus on that typical Hetero-MP has inter-processor interrupts and a shared memory. With these two elements, we propose an implementation of IDC which is essential for Hetero-HP systems. To make the design efficiency better, we developed a tool which automatically generates the IDC for the target system. The case study shows that our method increases the design efficiency by the automatic synthesis of inter-heterogeneous-processor communication implementation.

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

References

[1] Altera Corporation: available from ⟨https://www.altera.com⟩ (accessed 2015-5-1).
[2] Xilinx Inc.: available from ⟨http://www.xilinx.com⟩ (accessed 2015-5-1).
[3] ARM Ltd.: available from ⟨http://www.arm.com⟩ (accessed 2015-5-1).
[4] The Multicore Association: available from ⟨http://www.multicore-association.org/index.php⟩ (accessed 2015-5-1).
[5] Meakin, B. and Gopalakrishnan, G.: Hardware Design, Synthesis, and Verification of a Multicore Communications API, *SRC Techcon* (2009).
[6] Matilainen, L., Salminen, E., Hamalainen, T. and Hannikainen, M.: Multicore Communications API (MCAPI) implementation on an FPGA multiprocessor, *2011 International Conference on Embedded Computer Systems* (*SAMOS*), pp.286–293 (online), DOI: 10.1109/SAMOS.2011.6045473 (2011).
[7] Honda, S., Tomiyama, H. and Takada, H.: RTOS and Codesign Toolkit for Multiprocessor Systems-on-Chip, *Design Automation Conference, 2007. ASP-DAC '07, Asia and South Pacific*, pp.336–341 (online), DOI: 10.1109/ASPDAC.2007.358008 (2007).
[8] Kahn, G.: The semantics of a simple language for parallel programming, *Proc. IFIP Congress 74*, pp.471–475 (1974).
[9] TOPPERS Project: available from ⟨http://toppers.jp/en/index.html⟩ (accessed 2015-5-1).

Yuki Ando received his Ph.D. degree in Information Science from Nagoya University in 2014. Currently he is a researcher at Center for Embedded Computing Systems, Nagoya University. His research interests include system-level design and embedded systems.

Yukihito Ishida received his B.E. degree in information engineering and M.S. degree in Information Science from Nagoya University in 2012, and 2014, respectively.

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 ACM, IEEE, IEICE, and JSSST.

Hiroaki Takada is a professor at Institute of Innovation for Future Society, Nagoya University. He is also a professor and the Executive Director of the Center for Embedded Computing Systems (NCES), the Graduate School of Information Science, Nagoya University. 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, IEICE, JSSST, and JSAE.

Masato Edahiro is a professor at the Department of Information Engineering, the Graduate School of Information Science, Nagoya University. He received his Ph.D. degree in Computer Science from Princeton University in 1999. He joined NEC Corporation in 1985, had worked in its research center for 26 years, and moved to Nagoya University in 2011. His research topics include graph and network algorithms and software for multi- and many-core processors. He is a member of IEEE, IEICE, ORSJ.

(Recommended by Associate Editor: *Takeshi Kumaki*)