Electronic Preprint for Journal of Information Processing Vol.23 No.5

Regular Paper

System-level Design Method for Control Systems with Hardware-implemented Interrupt Handler

Yuki Ando[1,a)] Shinya Honda[1,b)] Hiroaki Takada[1,c)] Masato Edahiro[1,d)]

Received: November 21, 2014, Accepted: May 9, 2015

Abstract: In this paper, we propose a system-level design method for control systems that enables the development of Hardware-implemented interrupt handler. The increasing complexity of control systems has led to a rise in the frequency of interrupts. As a result, the processor load increases, leading to a deterioration in the latency of interrupt processing. To solve these problems, we require dedicated hardware that is activated by an interrupt and can directly access devices during its processing. The proposed method enables control systems with the above dedicated hardware to be developed using a model that abstracts an interrupt, interrupt processing, and communication between the control processing and devices. We have developed a system-level design tool which automatically generates the target implementation from the model. Case studies on a motor control system show that the proposed method reduces the processor load, improves the latency of the interrupt processing, and enables the design space exploration for the control system.

Keywords: hardware/software co-design, design environments, design space exploration, interrupt handling

1. Introduction

Technological advances in system-on-chips, actuators, and input/output hardware (HW) have led to the realization of high-level control systems. In any control system, there is a need for communication between the control processing running on the processor and the devices under control. For example, a typical sensor converts analog data to digital data through an analog-to-digital converter (ADC). The control processing then reads this digital data. As well as the communication between control processing and devices, the control system must also handle any interrupts that are asynchronously notified from the devices. In general, the control system consists of several devices, and the processor handles the frequent interrupts that originate from the devices. The interaction of these interrupts can increase the latency of the interrupt processing. Furthermore, handling the interrupt processing causes an increase in processor load.

One method of improving the latency of interrupt processing and reducing the processor load is Renesas's Event Link Controller (ELC) [15]. ELC is a HW circuit being connected to several interrupt signals. Instead of an interrupt processing running on a processor, ELC directly activates a module (e.g., timer) by an interrupt itself. Then, the module starts its processing without the activation of the processor. Another method involves an HW-implemented interrupt handler (HW-INH). HW-INH is a dedicated system that handles the interrupt processing. To enable this, the HW-INH is activated by the interrupt itself, and often requires access to one or more devices while it is running. An example of HW-INH is Atmel's SleepWalking technology [2]. In this approach, the HW-INH is activated by the interrupt, and then judges whether the processor should be activated. If the processor needs to be activated, the HW-INH sends an activation notification to the processor. Otherwise, the HW-INH accesses the relevant device(s) and handles the interrupt processing. In these cases, the processor load is reduced, because the processor is not activated. Furthermore, it is possible to improve the latency of the interrupt processing, because the HW-INH can handle the interrupt processing of each device in parallel. Therefore, HW-INH is important to enable high-performance control systems.

System-level design methods produce systems consisting of processors and dedicated HW. System-level design first considers the system at a high level of abstraction, without considering the eventual HW and software (SW). The designers then determine a suitable mapping that allocates the processing implementation. The specification of the system is revised by repeatedly changing and evaluating this mapping. By evaluating the system performance at a high level of abstraction, designers hope to avoid the need to step back during the second half of the design process. Thus, system-level design makes the design process more efficient.

A number of system-level design tools have been proposed and developed [7]. Some tools provide a modeling and simulation environment at a system level, although they do not include the ability to generate an implementation of the resulting prototypes. Others provide a unique model description, and offer the ability to generate prototypes on the target architecture using this model. However, these existing tools do not support the design of HW-INH.

[1] Nagoya University, Nagoya, Aichi 464–8603, Japan
[a)] y_ando@ertl.jp
[b)] honda@ertl.jp
[c)] hiro@ertl.jp
[d)] eda@ertl.jp

This paper proposes a design methodology that can describe a control system at a high level of abstraction as a model. In addition, we develop a system-level design tool that can automatically generate a target implementation, including the HW-INH, from this model. The proposed method automatically generates the HW-INH by extending a system-level design tool that we have been developing, named SystemBuilder [9]. Unlike previous system-level design tools including SystemBuilder, the proposed system-level design tool can handle an interrupt, interrupt processing, HW-INH, and communication to devices. By supporting them, the proposed system-level design tool expands the design space. We also present a control system model to realize the system-level design of control systems. Using this control system model, designers can describe interrupts, interrupt processing, and the communication between processing and devices at a high level of abstraction. The interrupt processing in the control system model is implemented as an interrupt handler on the processor, or it is implemented as the HW-INH. The tool automatically generates the target implementation from the control system model. Therefore, it is possible to explore the design space of a control system, and the control system can be efficiently designed.

In the past, designers have been able to handle an event, event processing, HW-INH, and communication to devices by themselves. However, previous system-level design tools including SystemBuilder have not been able to do that. Unlike previous system-level design tools, SystemBuilderCtr can handle an event, event processing, HW-INH, and communication to devices. By supporting them, SystemBuilderCtr expands the design space.

Hence, the contributions of the proposed method are:

- A tool for the design of HW-INH
- A control system model
- An efficient design method for control systems (automatic generation of the target implementation)

This paper is organized as follows. Section 2 introduces the related works. Section 3 indicates the concept behind the proposed method, and Section 4 presents the details of the control system model. In Section 5, we describe how the proposed tool enables automatic system implementation. Section 6 demonstrates the effectiveness of the proposed method through a case study on a motor control system, and Section 7 concludes the 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 as a target architecture.

SCE (System-on-Chip Environment) [4] is a system-level design framework based on the SpecC language [6]. 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 [14] provides modeling and simulation methods and tools for efficient performance evaluation and exploration of heterogeneous embedded multimedia systems. Artemis's design flow starts 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 Environment) [8] 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 [3] 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 [12] provides a simulation platform for a model written 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 the generation of HW-INH. Our system-level design tool, SystemBuilderCtr, is the first system-level design tool which automatically synthesizes the implementation with dedicated HW.

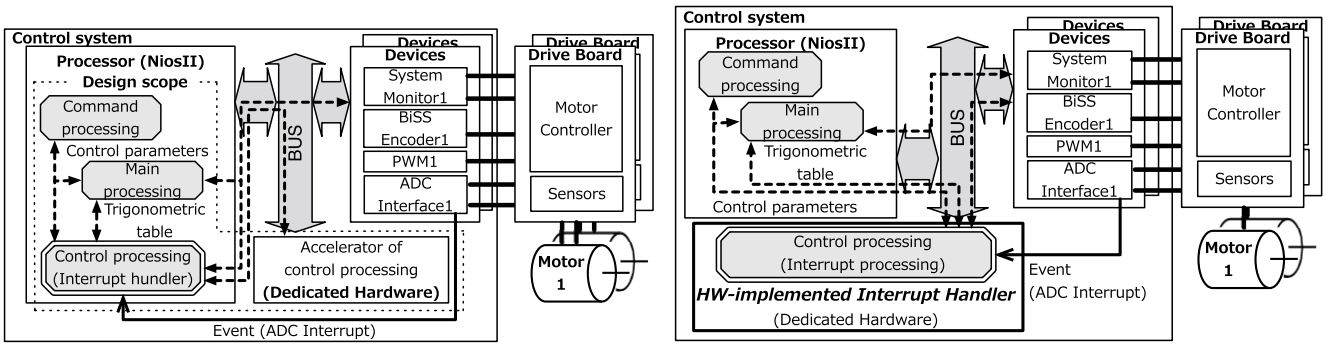
3. System-level Design Tool for Control Systems

This section describes SystemBuilderCtr, a system-level design tool for control systems. This tool is an extension of SystemBuilder [9], which we are in the process of developing.

3.1 Target Control System

Figure 1 (a) shows an example of the target control system (motor control system). This system controls the motor so that it maintains the target rotation rate. The control processing is activated by the ADC interrupt, which notifies the end of ADC. Control parameters are then calculated from sensor values provided by the device(s) to enable the control of the motor. Next, the system controls the motor by writing the calculated parameters to the devices. Note that a dedicated HW which is activated by the processor can accelerate some control processing (e.g., calculation of parameters). The main processing first initializes the trigonometric table and control parameters, and then supervises the motor status. The command processing sets the target rate of rotation depending on commands received from an external system.

The design scope of the proposed method is shown by the dotted lines in Fig. 1 (a), and does not include processors, buses, and devices. The processing of the control system is executed on the processor and the dedicated HW. Devices are HW units that may be designed using the hardware description language (HDL). We do not include them in the design scope, instead assuming that existing devices are to be used. Devices have device registers, which are mapped to the same memory space as the processors



(a) Control processing is partially processed by HW accelerator.

(b) Control processing is processed by HW-INH.

Fig. 1 An example of a target system (motor control).

and the dedicated HW. Hence, the processing can communicate with the devices. When an asynchronous notification (e.g., an interrupt) arises in a device, the processing modules gets the notification from the device. In this paper, the asynchronous notification is described as an "event," and the processing activated by an event is described as "event processing." For example, in the case of sensors, an event is the ADC interrupt that notifies the end of ADC. The event processing is the control processing activated by the ADC interrupt and implemented as an interrupt handler.

3.2 HW-implemented Interrupt Handler (HW-INH)

HW-INH is dedicated hardware that is activated by the interrupt itself, and then handles the interrupt processing. The HW-INH often requires access to one or more devices while it is running. Figure 1 (b) shows an example of the motor control system with the HW-INH. The control processing is implemented as HW-INH that is activated by an ADC interrupt. The HW-INH then handles the control processing, and accesses the devices while it is running.

By implementing interrupt processing through HW-INH, the interrupt processing is not executed on the processor. HW-INH has the potential to reduce the processor load of the control system. In addition, by handling the interrupt processing of each device in parallel, the HW-INH can improve the latency of the interrupt processing. The advantages of HW-INH can be summarized as follows:

- Reduced processor load
- Improved latency of interrupt processing

3.3 Concept of the Proposed Wethod

Designers have been able to handle an event, an event processing, HW-INH, and communication to devices by themselves. However, previous system-level design tools including SystemBuilder have not been able to do that. **Figure 2** illustrates the design of a control system by SystemBuilderCtr, which is an extension of SystemBuilder [9]. Unlike previous system-level design tools, SystemBuilderCtr can handle an event, an event processing, a HW-INH, and communication to devices. By supporting them, SystemBuilderCtr expands the design space.

SystemBuilderCtr takes three inputs:

- Control system model
- Mapping
- Architecture template

The control system model describes the processing and communication in the target system at a high level of abstraction (see Fig. 1 (a)). This model is an extension of the Kahn Process Network [11]. In the model, the processing and communication are indicated as a process and a channel, respectively. All processes are written in the C language, using APIs for channel access. Details of the control system model are explained in Section 4. The architecture template includes the number and type of processors, number and type of memory modules, bus structure, and input/output (IO) module information. In other words, it describes the target architecture. The mapping indicates the allocation of processes to SW or HW. For example, in the 1motor-SP mapping shown in Fig. 2, all processes are implemented as SW. In contrast, 1motor-SP-HW shows a mapping in which the Control1 process is implemented as HW, and the other processes are implemented as SW.

SystemBuilderCtr automatically generates the target implementation from three inputs according to the given mapping. While the implementation is being generated, processes are allocated to the processor and the dedicated HW, and channels are allocated to the memory modules. APIs to access the channels are implemented as drivers, a master interface (IF), and a slave IF. By changing the mapping in this way, SystemBuilderCtr automatically generates the target implementation according to the mapping, as shown in Fig. 2. Without SystemBuilderCtr, designers have to modify a configuration file of real-time operating system (RTOS), a driver description, HDL description of master IFs, and HDL description of slave IFs by themselves to change the allocation of processes. Even if the designers use a high-level synthesis (HLS) [5] tool, they still need to modify a configuration file for HLS tool and HDL description of registers and bus IFs. At least, a few lines of files should be modified. In some cases, hundreds of lines of files should be modified. On the other hand, with SystemBuilderCtr, the designers just need to modify a few lines of mapping information to change the allocation of processes. In this way, the automatic implementation can reduce the number of modified lines during exploration. Thus, designers can efficiently explore different mappings and can easily evaluate and compare various implementations.

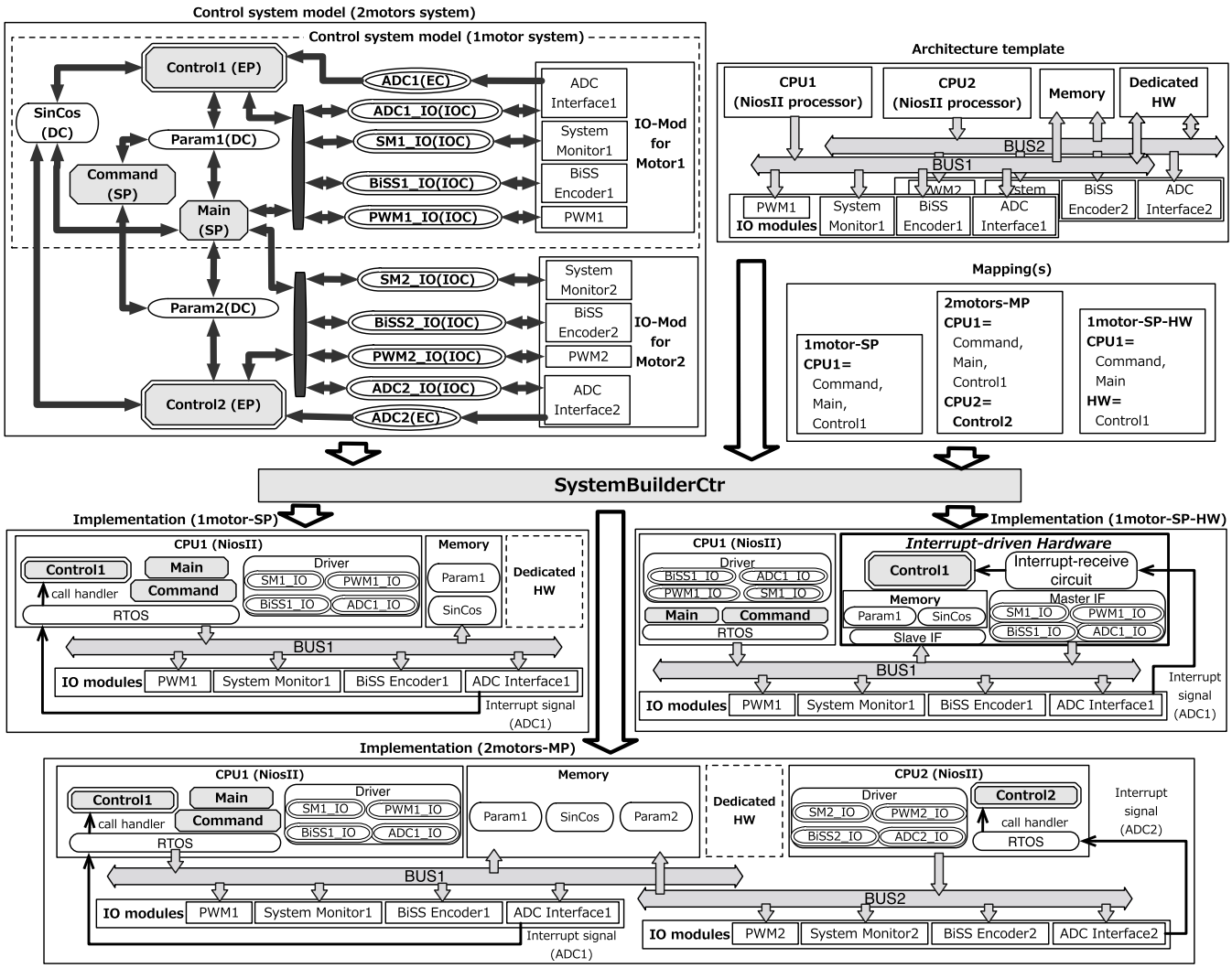


Fig. 2 Diagram showing the workflow of the tool and example implementations.

3.4 Requirements of the Control System Model and Design Tool

To realize the concept described in the previous section, the target control system shown in Fig. 1 (a) must be described as a control system model. Furthermore, we also require a tool that automatically generates the implementation from this model. The requirements needed to realize the concept and tool are listed below.

R1: Ability to describe an event and event processing
R2: Ability to describe communication between the processing and devices
R3: Ability to realize HW-INH
R4: Ability to formulate a model without significant changes in system performance

The target control system shown in Fig. 1 (a) includes the event, event processing, and communication between the processing and the devices. As these items are essential to the control system, the model must be able to describe them. Considering the realization of HW-INH, an event processing should be implemented as HW-INH when it is mapped to HW, while it should be implemented as an interrupt handler when it is mapped to SW. The implementation of a model should not lead to large changes in system performance, because such changes may render the control system unable to control the target.

4. Control System Model

This section describes a control system model that satisfies the requirements outlined in Section 3.4. First, we give a summary of the control system model, and then present a more detailed description.

4.1 Summary of the Control System Model

An example of the control system model is shown in **Fig. 3**. This is a model description of a part of the motor control system shown in Fig. 1 (a). The control system model consists of an IO module (IO-Mod) to represent a device, a process to represent the system processing, and a channel to represent communication among the processes and the IO-Mod. All processes are written in the C language, using APIs for channel access. In the figure, examples of C description with APIs for channel access are described.

We consider a standard process (SP) and an event process (EP). SPs realize all system processing except event processing. An EP realizes an event processing. **Table 1** lists the characteristics of these processes. All processes can be mapped to either SW or HW. If mapped to SW, SPs are implemented as tasks, whereas

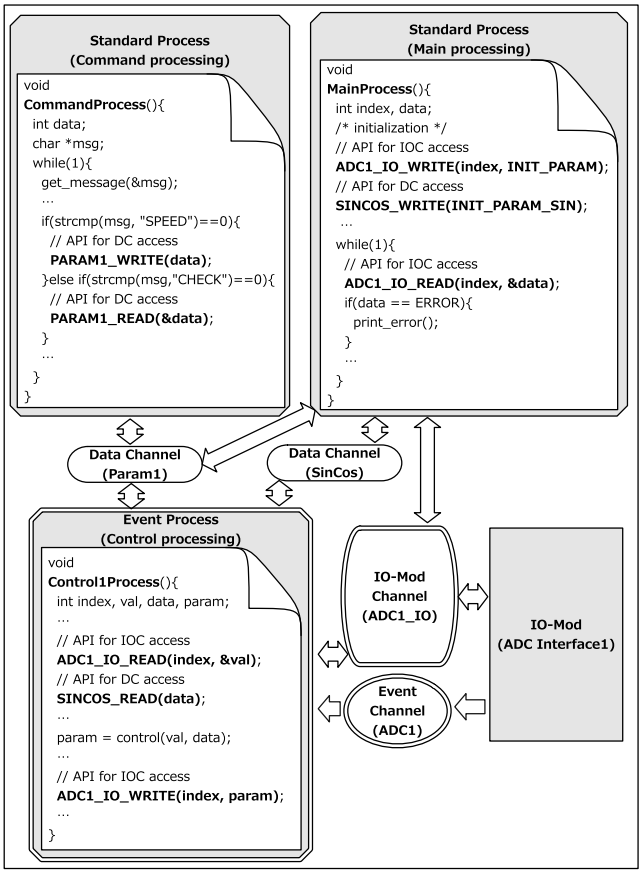


Fig. 3 An example of a control system model.

Table 1 Summary of processes.

Type of process	Standard process	Event process
Available mapping	SW/HW	SW/HW
Implementation of SW	Task	Interrupt handler
Use of DC	Allowed	Partially banned
Use of EC / IOC	Allowed	Allowed
Wake-up channel	DC	EC

EPs are implemented as an interrupt handler. Furthermore, the HW-INH can be realized by mapping the event process to HW.

There are three channel types: data channels (DCs), event channels (ECs), and IO-Mod channels (IOCs). DCs allow communication among processes, and ECs carry the notification of an event from an IO-Mod to an event process. IOCs realize communication between the IO-Mod and the processes. Basically, all processes can handle all channel types. However, EPs cannot use part of the DC, because the DC has an ability to terminate the processing. Channels are also used to wake up processes. For instance, DCs and ECs can be used to wake up the SPs and EPs, respectively.

The control system model is an extension of the model of SystemBuilder [9]. However, the control system model differs from the model of SystemBuilder. The control system model has the following new elements which do not exist in the model of SystemBuilder.

- event channel
- event process
- IO-Mod channel

In addition, the control system model satisfies requirements R1, R2, and R3, which are described in Section 3.4. The control system model satisfies R1, because an event and an event processing are described as an EC and an EP, respectively. Because an IOC describes the communication between the processes and devices, the control system model satisfies R2. Furthermore, the event process can be mapped to HW as shown in Table 1. This realizes the HW-INH, and the control system model satisfies R3. To evaluate R4, system performances should be measured. Thus, R4 is evaluated in Section 6.1. The following subsection gives a detailed description of the processes and channels.

4.2 Standard Processes and Data Channels

SPs describe all system processing, except for event processing. For example, in the control system model, the Command and Main processes are SPs because they are not event processing.

DCs transfer data among processes, and can be one of the following four types.

- Blocking channel
- Non-blocking channel
- Memory channel
- Exclusive access object

Blocking channels realize synchronous communication with first-in first-out buffers. They are used to wake up SPs, and mainly realize synchronization between two SPs. Non-blocking channels and memory channels realize asynchronous communication, transferring single data points and sets of data, respectively. Exclusive access objects are used to realize exclusive access control among the processes.

4.3 Event Processes and Event Channels

EPs describe the event processing. For example, in the control system model, the Control1 is EP. The event processing cannot be terminated during its execution. Thus, the EPs are distinguished from the standard processes which can be terminated by the blocking channels. For the same reason, the EPs cannot be activated by the blocking channels. Instead of the blocking channels, the EPs are woken up by the EC which is described below.

An EC carries the notification of an event from the IO-Mod to the event process. The ADC interrupt is an example of an EC. ECs connect a single event and a single event process. If the event occurs in an IO-Mod, they carry a wake-up signal to the relevant event process.

In this way, the event and the event processing are described as a channel and a process in the control system model. In addition, EPs can be implemented as HW-INH when they are mapped to HW.

4.4 IO-Mod and IO-Mod Channel

The IO-Mod indicates the device. For example, the ADC Interface is an IO-Mod. It translates analog data from the sensors into digital data, and stores the translated data in the device register. The processor and dedicated HW can access the device register via the bus.

The IOC realizes the transfer of data between the IO-Mod and the processes. For example, the API to read data provided by the IOC is called to read data from the ADC Interface. This API

is implemented to read data in the device register inside the IO-Mod. A single IOC is needed for each IO-Mod, and each IOC can be connected to multiple processes. The control system model can describe communication between the processes and the devices through the IOC.

5. Automatic Implementation from the Control System Model

Figure 2 shows an example of the automatic generation of an implementation from the control system model. In the case that an SP is mapped to SW, it is implemented as a task in a RTOS [16]. If the SP is mapped to HW, it is translated to HDL using HLS, and is implemented on dedicated HW (the figure does not show this part). Our tool supports a HLS tool named eXCite [10]. In the case that an EP is mapped to SW, it is implemented as an interrupt handler in the RTOS, and is called when the RTOS receives a relevant interrupt. When the EP is mapped to HW, it is translated to HDL using HLS tool, and implemented on the dedicated HW. In this case, because an interrupt-receive circuit is also implemented, the EP is implemented as HW-INH that is activated by the notification of a relevant interrupt.

The DCs are allocated to the memory. In the case that the process connected to the DC is mapped to HW, the slave IF is implemented. If the process connected to the EC is mapped to SW, the EC is registered to the RTOS as an activation condition of the interrupt handler. However, if this process is mapped to HW, the EC is converted to an interrupt-receive circuit. If the process connected to the IOC is mapped to SW, it is implemented as a driver on the processor. When such a process is mapped to HW, the IOC is converted to a master IF.

6. Case Studies on a Motor Control System

This section describes case studies using the motor control system [13]. The motor control system controls up to two motors as shown in Fig. 1 (a). The motor control system is implemented on DE2 board having an Altera's FPGA which runs at 100 MHz [1]. In the FPGA fabric, there are up to two NiosII soft-core processors that are connected to the devices through the Avalon buses. The ADC Interfaces raise an interrupt signal with a period of 62.5 μs (16 KHz). The control processing is executed by the ADC interrupt. The control parameters calculated by the control processing are then transferred to the drive board by writing them to the devices. Finally, the drive board rotates the motor depending on the setup control parameters.

Figure 2 shows the control system model of 2motors system which controls two motors. The control system model for 1motor system which controls one motor is shown inside the dotted lines. This figure also shows three examples of implementation that were automatically generated by SystemBuilderCtr. The control system model of 2motors system consists of the following four processes.

- Command (SP): corresponding to the command processing
- Main (SP): corresponding to the main processing
- Control1 (EP): corresponding to the control processing for motor1
- Control2 (EP): corresponding to the control processing for motor2

Table 2 Mapping patterns.

Mapping	Main	Command	Control1	Control2
1motor-SP	CPU1	CPU1	CPU1	—
1motor-SP-HW	CPU1	CPU1	HW	—
2motors-SP	CPU1	CPU1	CPU1	CPU1
2motors-MP	CPU1	CPU1	CPU1	CPU2
2motors-SP-MIX	CPU1	CPU1	CPU1	HW
2motors-SP-HW	CPU1	CPU1	HW	HW

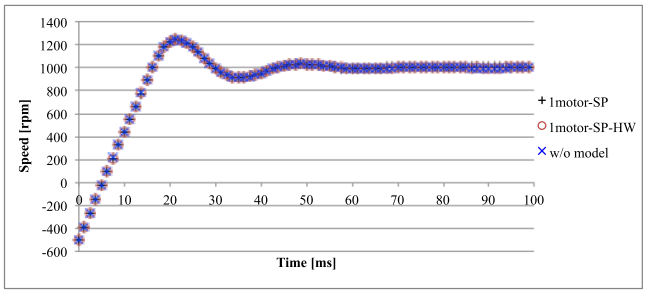


Fig. 4 Quality of speed control.

The Main process is connected to the Param1, Param2, and SinCos DCs. The Param1 DC and the Param2 DC retain the control parameters of motor1 and motor2, respectively. The SinCos DC holds the trigonometric table value. The Command process is connected to the Param1 and Param2 DCs. The Control1 process and the Control2 process are connected to the Param1 DC and the Param2 DC, respectively, and they are connected to the SinCos DC. The Control1 process is connected to four IOCs (SM1_IO, BiSS1_IO, PWM1_IO, ADC1_IO), and the Control2 process is connected to four IOCs (SM2_IO, BiSS2_IO, PWM2_IO, ADC2_IO). In addition, the Control1 process and the Control2 process are connected to the ADC1 EC and the ADC2 EC, respectively. Using SystemBuilderCtr, six implementations (see **Table 2**) were automatically generated from the above control system models. In the following sections, all six implementations are compared.

6.1 Evaluation of Requirements

We show that the proposed method and this case study satisfy the requirements described in Section 3.4. Because the control system model of 2motors system includes ECs and event processes, this case study satisfies R1. This case study satisfies R2 because the model includes up to eight IOCs. The implementation of 1motor-SP-HW in Fig. 2 has a HW-INH which is automatically generated by SystemBuilderCtr. Thus, SystemBuilderCtr satisfies R3.

We have measured the system performance in terms of control quality in order to evaluate requirement R4. **Figure 4** shows the change in the rate of rotation of the motor when the target rate of rotation is changed from −500 rotations per minute (rpm) to +1,000 rpm. The figure shows the result of three implementations of 1motor system. 1motor-SP and 1motor-SP-HW were implemented from the control system model by SystemBuilderCtr. The one named "w/o model" was implemented as SW without the control system model. Because of a limitation of the drive board, we only measured the rate of rotation on 1motor system. Note that all implementations could rotate the motor correctly.

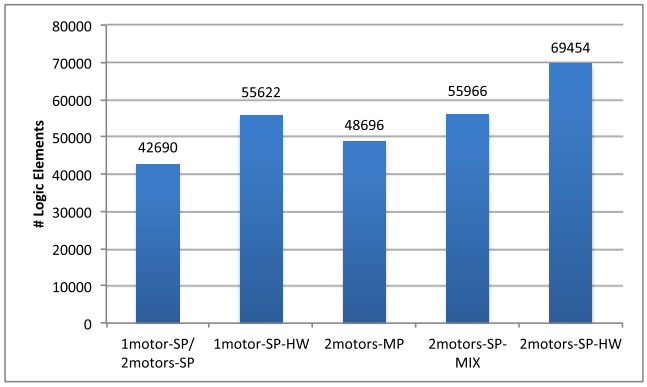


Fig. 5 Comparison of the number of logic elements.

From the figure, we can confirm that the rate of rotation converges. The root-mean-square error between the rate of rotation produced by 1motor-SP and the rate of rotation produced by w/o model is 2.4%, and that for 1motor-SP-HW is 2.4%. These results indicate that the system performance is not changed by the system modeling approach. The control system model and SystemBuilderCtr satisfy R4. Therefore, the proposed method and this case study satisfy the requirements R1, R2, R3, and R4.

6.2 Comparison of the Number of Logic Elements

Figure 5 shows the number of logic blocks of Altera's Cyclone IV FPGA (logic elements: LEs) for six implementations, which includes CPUs, buses, motor control devices, slave IFs, master IFs, and dedicated HW. At least, a CPU, a bus, and motor control devices are common to six implementations. In detail, a single processor and bus need about 6,000 LEs. A HW-INH of the Control1/2 process needs about 13,000 LEs, and motor control devices need about 37,000 LEs.

1motor-SP and 2motors-SP use the same target implementation which has a CPU and a bus. 2motors-SP only needs a bus because the Control1 and Control2 processes run on the CPU in sequence. 2motors-MP has two CPUs and two buses. Because the Control1 and Control2 processes run on different CPU, two buses are required to avoid a bus collision. In the result, 2motors-MP has more LEs than 2motors-SP.

The number of LEs for 1motor-SP-HW and 2motors-SP-MIX is almost the same. 1motor-SP-HW has a single bus because it only controls a motor. 2motors-SP-MIX has two buses because a CPU controls a motor and a dedicated HW controls another one in parallel. Thus, 2motors-SP-MIX slightly has larger usage of LEs than 1motor-SP-HW. 2motors-SP-HW has the largest usage of LEs among the six implementations because it has two dedicated HWs (HW-INH).

Adding an HW-INH increases the usage of LEs. On the other hard, the HW-INH brings some advantages which are discussed in coming sections. Therefore, the designer has to consider the trade-offs between the usage of LEs and the advantages of HW-INH.

6.3 Reduction of Processor Load by HW-INH

We measured the processor load by the Control1 and Control2 processes. The Command process basically waits commands

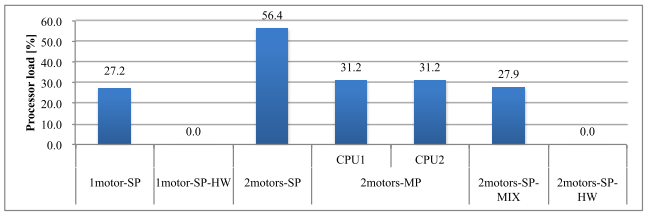


Fig. 6 Comparison of the processor load.

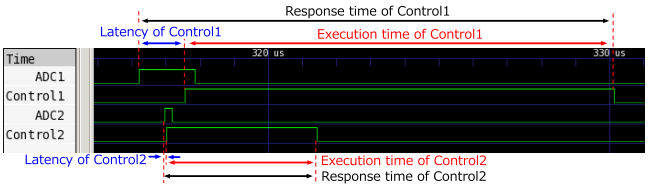


Fig. 7 An example of the execution time and latency of the Control1 and Control2 processes.

from outside of the system. The processor load of the Main process depends on cycle of Main process's supervision function. Since we focused on unveiling that the HW-INH reduces the processor load, the Main and Command processes were not executed during this measurement. **Figure 6** shows the processor load of the six implementations.

In the implementations of 1motor system, the processor load for 1motor-SP was 27.2%, while that for 1motor-SP-HW was 0.0%. In 1motor-SP-HW, the Control1 process is implemented as HW-INH, and therefore its processing is offloaded from the processor. As a result, the processor load is reduced to 0.0%. In other words, the processor did not execute any processing.

The processor load for 2motors-SP was 56.4%, which is about double that of 1motor-SP. This is because CPU1 runs the Control1 and Control2 processes in sequence. The processor load for 2motors-SP-MIX was 27.9%, which is almost the same as 1motor-SP. The Control2 process is implemented as HW-INH, and therefore its processing is offloaded from the processor. As a result, CPU1 only runs the Control1 and its processor load became the same as 1motor-SP. The processor loads of CPU1 and CPU2 for 2motors-MP were both 31.2% because CPU1 runs the Control1 process and CPU2 runs the Control2 process. Because this implementation uses RTOS for multi-processors, the overhead of interrupt handling is larger than that for single-processor. As a result, the processor loads of 2motors-MP were larger than that of 1motor-SP. The processor load for 2motors-SP-HW was 0.0% because the Control1 and Control2 processes are implemented as HW-INH, and therefore their processing is offloaded from the processor.

By implementing the HW-INH, no interrupt processing was executed on the processor. Offloading the interrupt processing has a prospect to keep the processor in a standby mode. Therefore, the HW-INH has a possibility to reduce the processor load of the control system.

6.4 Confirmation of the Real-time Requirements

We measured the execution time, latency, and response time of the Control1 and Control2 processes as shown in **Fig. 7**. In this figure, ADC1 and ADC2 indicate interrupt signals from ADCs,

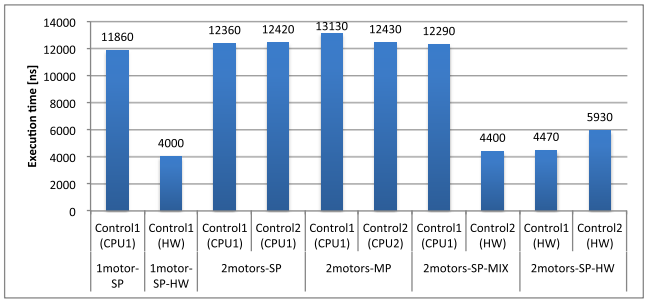


Fig. 8 Comparison of execution time of the Control1/Control2 processes.

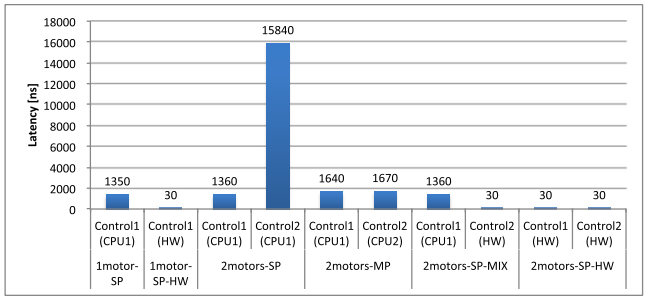


Fig. 9 Comparison of latency of the Control1/Control2 processes.

and Control1 and Control2 indicate the execution of the Control1 process and the Control2 process, respectively. The execution time is the time from the activation of the Control1 (Control2) process to the end of its execution. The latency is the time from the occurrence of an ADC interrupt to the activation of the Control1 (Control2) process.

Figure 8 shows the execution time of the Control1 and Control2 processes. For the 1motor system, the execution time of the Contorl1 process in 1motor-SP was 11,860 ns, and that in 1motor-SP-HW was 4,000 ns. For the 2motors system, the execution time of the Control process which is implemented as a CPU was approximately 12,300 ns. On the other hand, the execution time of the Control process which is implemented as HW was approximately 4,400 ns. Since the Control1 and Control2 processes share a SinCos DC, their execution times are longer than those of 1motor system. By implementing the Control process as HW, the execution time could be reduced by a factor of 2.0 at least.

Figure 9 shows the latency of the Control1 and Control2 processes. The latencies of the Control1 process in 1motor-SP, 2motors-SP, and 2motors-SP-MIX were about 1,350 ns, which is handled by RTOS for single-processor. On the other hand, the latencies of the Control processes in 2motors-MP were about 1,640 ns. Because of the overhead of RTOS for multi-processors, the latencies of the Control processes in 2motors-MP were longer than those handled by RTOS for single-processor. All latencies implemented as HW were 30 ns. At a frequency of 100 MHz, 30 ns is equivalent to 3 cycles.

Figure 10 shows the response time of the Control1 and Control2 processes. The response time of the Control2 process in 2motors-SP was the longest because both Control1 and Control2 were run on CPU1. Because its response time (28.26 μs) was shorter than the interrupt period (62.5 μs), all implementations satisfy the real-time requirement. Additionally, the response times which were implemented as HW were improved compared

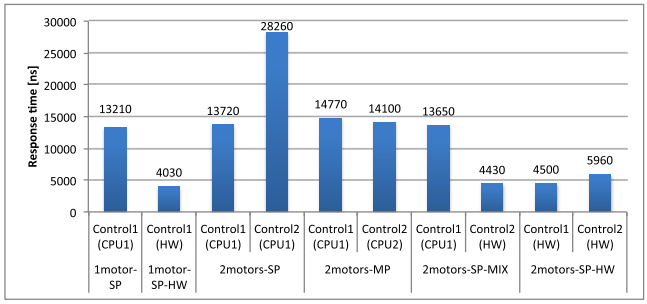


Fig. 10 Comparison of response time of the Control1/Control2 processes.

Table 3 The number of modified lines of files.

No.	w/o SysCtr				w/ SysCtr
	RTOS	APIs	HLS	HDL	Mapping
1 (1motor-SP)	—	—	—	—	—
2 (1motor-SP-HW)	2	18	35	1,021	2
3 (2motors-SP)	2	18	—	—	2
4 (2motors-MP)	8	—	—	—	2
5 (2motors-SP-MIX)	2	18	—	—	2
6 (2motors-SP-HW)	2	—	—	126	2
total	1,252				10

with those implemented as SW. As shown in Figs. 8 and 9, HW-INH reduced the execution time and the latency. This resulted in a reduction of the response time as well. From this point, the HW-INH is effective to improve the real-time performance.

By implementing the Control process as HW-INH, the latency was reduced to no less than 3 cycles, which is 1/45 of the latency of the interrupt handler. Shorter latencies are preferable in the design of control systems such as the motor control system. In addition, the HW-INH reduced the response time and improved the real-time performance. Thus, the HW-INH provides better real-time characteristics in terms of:

- A reduction in latency
- Faster execution of interrupt processing
- Shorter response time of interrupt processing

6.5 Efficiency of Exploration by Automatic Generation

Without SystemBuilderCtr, the designers have to modify several files by themselves to change the allocation of processes as decribed in Section 3.3. On the other hand, the designers just need to modify a few lines of mapping information to change the allocation of processes by using SystemBuilderCtr. To show the efficiency of the exploration with SystemBuilderCtr, we compared the number of modified lines of files between the implementations without SystemBuilderCtr and the implementations with SystemBuilderCtr.

Table 3 lists the number of modified lines of files during the exploration of six mappings. The six mappings were implemented in ascending order in the table. The column named "w/o SysCtr" indicates the number of modified lines of files without SytemBuilderCtr. The columns named RTOS, APIs, HLS, and HDL indicate a configuration file of RTOS, C code of the channel access APIs, a configuration file of HLS, and a HDL file of channels, respectively. In column "w/ SysCtr" indicates the number of modified lines of mapping information inputted to SytemBuilderCtr.

Since 1motor-SP is the first implementation, there is no modified file. The modification of RTOS file was a registration and

unregistration of an interrupt handler and task. When the allocation of processes was changed, each process required a modification of two lines in the RTOS file. The modifications of APIs were a read and write implementation and a configuration of base address. Each channel implementation required a modification of three lines in the file. Thus, 18 lines were modified for 6 channels when the channel implementations were changed. The modification of HLS was a configuration file for the HLS tool. Since a HW module generated by the HLS tool was reusable, 35 lines were modified at the first use of the HLS tool. The modification of HDL includes IF's circuits and connections between channels and HW modules. More than 1,000 lines were modified in the HDL file at the first use of dedicated HW (1motor-SP-HW). Since most of the description in the HDL file was reusable, no line was modified on 2motors-SP-MIX and 126 lines were modified on 2motors-SP-HW. Without SystemBuilderCtr, 1,252 lines in total were modified in four files.

With SystemBuilderCtr, two lines were modified in mapping information (CPU1=, CPU2=, or HW=) to change the allocation of processes as shown in Fig. 2. In total, only 10 lines of mapping information were modified during the exploration with SystemBuilderCtr. Comparing with the exploration without SystemBuilderCtr, the number of modified lines was reduced to 1/120. In addition, the designers do not need expert knowledge (e.g., a configuration file of RTOS and a HDL description) to explore different mappings. Especially, SystemBuilderCtr automatically generates HDL descriptions that HLS tools cannot generate. SystemBuilderCtr is very effective to change the allocation of processes from SW to HW because the designers had to change more than 1,000 lines of HDL without SystemBuilderCtr (see 1motor-SP-HW in Table 3). Therefore, SystemBiulderCtr can realize more efficient design space exploration by reducing the number of lines which should be modified.

Table 4 shows the time to implement the system from the models. SysCtr, SW compile, eXCite and QuartusII indicate a time to generate files by SystemBiulderCtr, a time to compile RTOS and tasks for processor, a time to generate HDL of processes by eXCite HLS tool, and a time to synthesize FPGA logics by QuartusII [1], respectively. For all implementations, we took less than one second for SysCtr and less than 15 seconds for SW compile. Implementations which have a process(es) mapped to HW additionally took time to generate dedicated HW (as shown eXCite and QuartusII), which are time consuming steps.

Because we only changed the mapping information and the implementation of channels and IFs which have traditionally required expert knowledge were automatically generated by SystemBiulderCtr, we only took one and a half hours to realize six implementations. Therefore, not only expert designers but also novice designers can efficiently explore design space of the control systems by our method.

Table 4 Time to implement the system from the model [seconds].

Mapping	SysCtr	SW compile	eXCite	QuartusII
1motor-SP	$\ll 1$	9.9	—	—
1motor-SP-HW	$\ll 1$	9.7	47	1,463
2motors-SP	$\ll 1$	11.1	—	—
2motors-MP	$\ll 1$	14.0	—	—
2motors-SP-MIX	$\ll 1$	11.0	47	1,506
2motors-SP-HW	$\ll 1$	10.8	95	2,092

7. Conclusion

We have presented a system-level design tool for control systems that enables the development of HW-INH that is activated by an interrupt. We also described a control system model that abstracts an interrupt, interrupt processing, and communication between control processing and devices. The proposed method automatically generates HW-INH from this control system model. Case studies using a motor control system demonstrated that our approach improves the efficiency of the design space exploration of control system design, and the control quality of the automatically generated implementation was found to be the same as that of a conventionally designed implementation. The HW-INH was shown to reduce the processor load, to shorten the execution time of interrupt processing, and to improve the latency of interrupt processing. Furthermore, our method enables designers to efficiently explore the design space of the control system.

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

References

[1] Altera Corporation (online), available from ⟨http://www.altera.com/⟩ (accessed 2014-10-27).
[2] Atmel Corporation (online), available from ⟨http://www.atmel.com⟩ (accessed 2014-10-27).
[3] Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C. and Sangiovanni-Vincentelli, A.: Metropolis: An Integrated Electronic System Design Environment, *Computer*, Vol.36, No.4, pp.45–52 (online), DOI: http://dx.doi.org/10.1109/MC.2003.1193228 (2003).
[4] 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 J. Embedded Syst.*, Vol.2008, pp.1–13 (online), DOI: http://dx.doi.org/10.1155/2008/647953 (2008).
[5] Gajski, D.D., Dutt, N.D., Wu, A.C.-H. and Lin, S.Y.-L.: *High-level Synthesis: Introduction to Chip and System Design*, Kluwer Academic, Norwell, MA, USA (1992).
[6] Gajski, D.D., Zhu, J., Dömer, R. and Gerstlauer, A.: *SpecC: Specification language and design methodology*, Kluwer Academic (2000).
[7] Gerstlauer, A., Haubelt, C., Pimentel, A.D., Stefanov, T.P., Gajski, D.D. and Teich, J.: Electronic system-level synthesis methodologies, *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, Vol.28, No.10, pp.1517–1530 (online), DOI: http://dx.doi.org/10.1109/TCAD.2009.2026356 (2009).
[8] 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. Des. Autom. Electron. Syst.*, Vol.12, No.3, pp.1–25 (online), DOI: http://doi.acm.org/10.1145/1255456.1255461 (2007).
[9] Honda, S., Tomiyama, H. and Takada, H.: RTOS and Codesign Toolkit for Multiprocessor Systems-on-Chip, *Design Automation Conference, ASP-DAC '07, Asia and South Pacific*, pp.336–341 (online), DOI: 10.1109/ASPDAC.2007.358008 (2007).
[10] Y Explorations Inc.: eXCite (online), available from ⟨http://www.yxi.com/⟩ (accessed 2014-10-27).
[11] Kahn, G.: The semantics of a simple language for parallel programming, *Proc. IFIP Congress 74*, pp.471–475 (1974).
[12] Mahadevan, S., Virk, K. and Madsen, J.: ARTS: A SystemC-based framework for multiprocessor systems-on-chip modelling, *Design Automation for Embedded Systems*, Vol.11, No.4, pp.285–311 (2007).
[13] Multiaxis Motor Control Development Board (online), available from ⟨http://www.altera.com/products/devkits/altera/kit-multi-axis-motor-control.html⟩ (accessed 2014-10-27).
[14] Pimentel, A.D.: The Artemis workbench for system-level performance evaluation of embedded systems, *International Journal of Embedded Systems*, Vol.3, pp.181–196 (2008).
[15] Renesas Electronics Corporation (online), available from ⟨http://www.

renesas.com/index.jsp⟩ (accessed 2014-10-27).
[16] TOPPERS Project (online), available from ⟨http://www.toppers.jp/en/index.html⟩ (accessed 2014-10-27).

Yuki Ando received his Ph.D. degree in Information Science from Nagoya University in 2014. Currently he is a researcher at the Center for Embedded Computing systems, 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 ACM, IEEE, IEICE, and JSSST.

Hiroaki Takada is a professor at the 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 the University of Tokyo in 1996. He was a research associate at the 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.