PAPER Special Section on Reconfigurable Systems

FPGA Components for Integrating FPGAs into Robot Systems

Takeshi OHKAWA[†a)], ***Member***, **Kazushi YAMASHINA**[†], **Hitomi KIMURA**[†], ***Nonmembers***, **Kanemitsu OOTSU**[†], ***and*** **Takashi YOKOTA**[†], ***Members***

SUMMARY A component-oriented FPGA design platform is proposed for robot system integration. FPGAs are known to be a power-efficient hardware platform, but the development cost of FPGA-based systems is currently too high to integrate them into robot systems. To solve this problem, we propose an FPGA component that allows FPGA devices to be easily integrated into robot systems based on the Robot Operating System (ROS). ROS-compliant FPGA components offer a seamless interface between the FPGA hardware and software running on the CPU. Two experiments were conducted using the proposed components. For the first experiment, the results show that the execution time of an FPGA component for image processing was 1.7 times faster than that of the original software-based component and was 2.51 times more power efficient than an ordinary PC processor, despite substantial communication overhead. The second experiment showed that an FPGA component for sensor fusion was able to process multiple sensor inputs efficiently and with very low latency via parallel processing.

key words: *FPGA, component-oriented development, ROS, ROS-compliant FPGA component, robot*

1. Introduction

Autonomous mobile robots with high intelligence [1], [2] are expected to be used for nursing care of handicapped and elderly people, disaster relief, and so on. They need to be able to recognize their environment independently using deep learning neural networks, simultaneous localization and mapping (SLAM), and so on. They also need to perform these intelligent processing tasks while consuming limited power, as they run on batteries. These robots must therefore perform intelligent image processing and other calculations using enormous amounts of sensor data in real-time with low power consumption. A high-performance, low-power processing platform is greatly needed for such robots.

To satisfy these processing platform requirements for robots, we focus on field-programmable gate arrays (FPGAs). An FPGA is a processing platform that can be used to create specialized circuits for parallel and high-performance processing with low power requirements [3], especially in fields such as image [4] and network packet processing [5].

Robotics depends on a wide range of expertise [6], not only in the logical design of hardware (HW) and software (SW) but also in electronics, mechanics, and so on. The focus of robot developers is not therefore on the design of the FPGA itself but on integrating the FPGA as a component. Currently, integrating an FPGA into a robot system is not easy because it requires not only the design of the FPGA HW but also of SW for controlling it. FPGAs should ideally be much more easy to use in robot development.

In the context of developing computer systems, reusing FPGA intellectual property (IP) modules requires providing either the source code (in hardware description language (HDL), C language for high-level synthesis (HLS), and so on) or a black-box netlist. However, the cost of integrating IP provided in a source code or netlist form is very high for robot development. Therefore, FPGA IP components for robot development should be more loosely coupled.

The Robot Operating System (ROS) [7] has been proposed as a software platform for component-oriented development of robots. ROS provides communication libraries and a build system for robotic application software. It is widely used by robot engineers all over the world for prototyping novel robotic systems and is becoming a mainstream platform for robotic application software development.

In this paper, we propose an ROS-compliant FPGA component, namely, a component-based design principle for easy integration of FPGAs into robot systems. ROS-compliant FPGA components can achieve easy FPGA integration and high-performance, low-power processing. The contributions of this paper are as follows:

- Proposal of ROS-compliant FPGA components for easy integration of FPGAs into robot systems
- Presentation of detailed component implementations
- Evaluation of these components both in terms of processing latency and power efficiency.

Compared with our initial proposal [8], this paper discusses the background and design concept in significantly more detail.

The structure of this paper is as follows. First, we discuss the issues in robot system development in Sect. 2. The design of our ROS-compliant FPGA components is described in Sect. 3. Section 4 explains the implementation of these components in detail. In Sect. 5, we consider two case studies to highlight the benefits of the proposed components and evaluate them in terms of processing latency and power efficiency. Section 6 discusses our proposal in more detail. Then, we survey previous research and highlight

Manuscript received May 2, 2017.
Manuscript revised September 8, 2017.
Manuscript publicized November 17, 2017.
[†]The authors are with Utsunomiya University, Utsunomiya-shi, 321–8585 Japan.
a) E-mail: ohkawa@is.utsunomiya-u.ac.jp
DOI: 10.1587/transinf.2017RCP0011

our proposal's importance in Sect. 7. Finally, the paper's conclusions are presented in Sect. 8.

2. Issues in Robot Software System Development

In this section, the issues in robot software system development are discussed in order to clarify the need to introduce FPGAs into robot software systems.

2.1 Component-Oriented Development of Robot Software Systems

The requirements for robot software system development are generally quite diverse [6]. For example, in the initial stages of research and development, functionality, performance, and power consumption are important elements. As development progresses and reaches a commercial level, the system design needs to fulfill the requirements of cost, reliability, manageability, and so on. In any case, the most important aspect of robot development is realizing the necessary functions. Particularly for cutting-edge robot development, the latest research in fields such as AI, computer science, and control theory should be integrated, and the fastest way to achieve this is with reusable components.

For example, in the robot vision field, AI technologies are expected to be used together with small, light-weight image sensors with high resolution and sensitivity, and with deep learning neural networks. Here each module should be a connectable component. Such components will inevitably be reusable and updateable to meet any requirement.

The performance of software is as important as its functionality. For real-time systems, including robot software, functionality innovations are intimately related to performance improvements. For example, in movement tracking for robot vision, the number of frames that can be processed per second has a direct impact on potential next-generation applications. Furthermore, power consumption, processing performance, and battery life all limit the size and weight of systems. The functionality, performance, and power consumption requirements all influence each other.

Architecture exploration involves searching for tradeoffs among functionality, performance, and power consumption. In architecture exploration, it is not realistic to implement a complete robot system because of the high cost and long time required for implementation. As a result, model simulations are usually carried out instead.

The models used for exploration should generate, or at least be closely related to, runnable software implementations for an existing processing platform. Any difference between the model and the implementation leads to a long turn-around time and reduces the quality, reliability, operability, and reusability of the system.

Component-oriented development [9] is a straightforward approach to directly connecting a model with its implementation. Here, a "component" is a module that can interface with other components and which can be replaced by another component with the same interface.

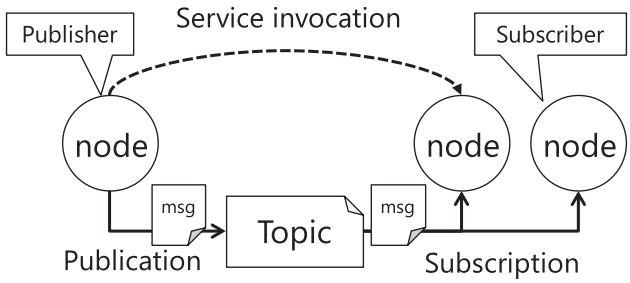


Fig. 1 Conceptual overview of an ROS system

Component reusability is one of the main benefits of component-oriented development. Using a well-designed component library enables rapid system prototyping.

In other words, realizing this approach, if it enables the performance requirements to be met while reducing power consumption, will enable significant progress in next-generation robot system applications.

2.2 Robot Operating System (ROS)

To incorporate FPGAs into robot systems, we chose the Robot Operating System (ROS) [7] as a framework for robot software system development. This is a software platform for component-oriented development of robotic application software on Linux or Microsoft Windows. It is an open-source project managed by the Open Source Robotics Foundation (OSRF) [27] and provides communication libraries and a build system for robotic application software. ROS mainly runs on the Ubuntu Linux operating system. Each ROS software component corresponds to a software package in the ROS system [7], and each package provides a different function, such as an image processing filter, camera input, or motor control. Many open-source, reusable software packages are available on the official ROS website [27], and ROS is becoming mainstream in robot development [10].

The primary model for data communication among ROS software components is publish/subscribe messaging (Fig. 1). Service invocation can also be used to control node status. Each software component, called a *node*, communicates by sending and/or receiving messages (data) through communication channels called *topics*. Nodes have two possible roles: *publisher* and *subscriber*. Publishers can publish messages to any topic, and the messages are queued in the topic. Subscribers, in turn, can subscribe to any topic: when a topic is updated, all subscribers receive a message from the topic and execute a callback task.

In ROS, nodes need not know which peer nodes they are communicating with. Since they are loosely connected, components can easily be added to or removed from the system. This loose binding among components improves productivity in designing and debugging robot software.

2.3 Using FPGAs in an ROS System

Normal ROS systems are purely implemented in software and do not use FPGAs. ROS nodes are written either in C++ or Python. When data input, processing, or output is done using these programming languages, FPGAs can be used in an ROS system in the same way as normal software. However, there are too many degrees of freedom in the above-mentioned development method to directly use FPGAs in the ROS package distribution scheme.

The advantage of using ROS is that there are several thousand packages available [27], and any package released by any other developer can be used on any PC. On the other hand, with FPGAs, the operating environments of the FPGA developer and FPGA user are not always the same. Generally speaking, ROS nodes (including FPGA nodes) designed for one environment cannot be used in any other environment. This portability problem with ROS packages, including FPGA modules, must be resolved.

With FPGA systems, it is generally left to developers to work out how to connect software with the FPGA and share data. That is, the frameworks for accessing data, shared between software and FPGA, and communication, e.g., using PCI express or Ethernet, are different for each environment. In the future, component developers and users should share a suitable software and communication framework.

2.4 Requirements for FPGA Components for Robots

The requirements for FPGA components for robots can be summarized as follows:

A) Seamless connection between the FPGA and the primary robot software platform (i.e., ROS)
B) Can be deployed within the ROS package scheme (portability)
C) No need to use FPGA design tools
D) Good performance with low power consumption.

3. Design of ROS-Compliant FPGA Components

In order to make the introduction of FPGAs into robot development easier, we propose the concept of an "ROS-compliant FPGA component" that meets the above FPGA component requirements for robots. In this section, we give a conceptual overview, discuss the requirements, and describe the structure of the components.

3.1 Conceptual Overview

An "ROS-compliant FPGA component" can be defined as *an ROS component made from an FPGA whose functionality is equivalent to that implemented in software*. That is, the message types and data formats used in ROS-compliant FPGA components are equivalent to those implemented in normal ROS software.

The goal with ROS-compliant FPGA components is to replace pure-software ROS components with functionally equivalent FPGA-accelerated components. Therefore, an FPGA integrated into a robotic system must have equivalent functionality and must send/receive ROS messages using the proper ROS protocol, imitating the original pure-software ROS component. As long as the ROS messages sent/received to/from other ROS components in the system are equivalent, users of ROS-compliant FPGA components can obtain better performance, compared with the original pure-software ROS component, while maintaining the same functionality.

Next, we clarify the meaning of "ROS-compliant." The word "compliant" generally means that something has been manufactured or produced in accordance with a specified body of rules. It is therefore natural to state that a component is "ROS-compliant" if it has been manufactured or produced in accordance with the rules given by the official ROS provider. However, ROS is an open-source project and does not provide detailed specifications or conformance tests, and we only have software implementations and their documentation. As discussed above, the goal with ROS-compliant FPGA components is to realize functionally equivalent ROS components that can send/receive ROS messages according to the proper ROS protocol. Therefore, in this paper, we define "ROS-compliant" components to be components that send/receive ROS messages using the proper ROS protocol. An overview of the proper ROS message protocol is given on the ROS/Technical Overview page of the ROS wiki [28].

3.2 Satisfying the FPGA Component Requirements

The four FPGA component requirements are listed in Sect. 2.4. The mapping between FPGA signal semantics and ROS messages needed to satisfy Requirement A is discussed in Sect. 3.3, and the structure used to realize this mapping is described in Sect. 3.4.

It is possible to partly satisfy Requirements B and C using compiled FPGA configuration data. However, when using external pins, the bitstream must then be re-complied using FPGA design tools for the external pin assignment of the user's board.

Requirement D can be satisfied by reducing both the processing and communication times for the FPGA component. User application logic (for example, image processing) is responsible for the processing time, and the framework of the FPGA component is responsible for the communication time. Communication latency is discussed in Sect. 3.4.

Finally, the overall performance, including processing and communication, should be verified via case studies on a number of example designs. In this paper, two examples are presented in Sect. 5.

3.3 Mapping between FPGA Signals and ROS Messages

In order to achieve seamless connection between the FPGA

Table 1 Some built-in ROS message types (taken from [29])

Primitive type	*Serialization*	*C++*	*Python*
bool	unsigned 8-bit int	uint8_t (2)	bool
int8	signed 8-bit int	int8_t	int
int32	signed 32-bit int	int32_t	int
float64	64-bit IEEE float	double	float
string	ascii string (4)	std::string	str

```
int8    a
int8    b
int32   result
```

(a) ROS message

```
module calc(
        input [7:0] a,
        input [7:0] b,
        output reg [31:0] result,
        input clk,
        input reset
);
always@(posedge clk)
  if(reset) result <= 0;
    else result = a + b * 16'h1234;
  endif
endmodule
```

(b) Verilog-HDL module

Fig. 2 Example mapping between an ROS message (a) and the corresponding Verilog HDL module (b)

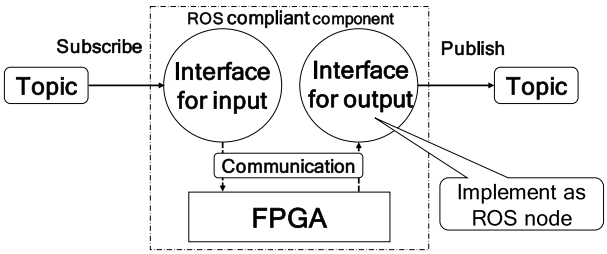


Fig. 3 Structure of ROS-compliant FPGA components

and ROS, it is necessary to set up a mapping between FPGA signals and ROS messages. ROS defines message types in an implementation-neutral fashion because ROS nodes can be implemented in C++ or Python.

ROS message types are defined in [29], together with mappings to C++ and Python. These types can either be built-in primitive types (as shown in Table 1), arrays of such types, or combinational types defined in message files (.msg) in packages. Interface signals from HDL modules written in Verilog HDL (VHDL) can be mapped according to the bit width of each signal; an example mapping is shown in Fig. 2. In the case of HLS from C/C++, the mapping can be more straightforward, i.e., the same mapping can be used for both.

3.4 Structure of ROS-Compliant FPGA Components

Figure 3 shows the proposed ROS-compliant FPGA component structure, which realizes a semantic mapping between FPGA signals and ROS messages. On the basis of the requirements discussed in the previous section, components must implement the following functions:

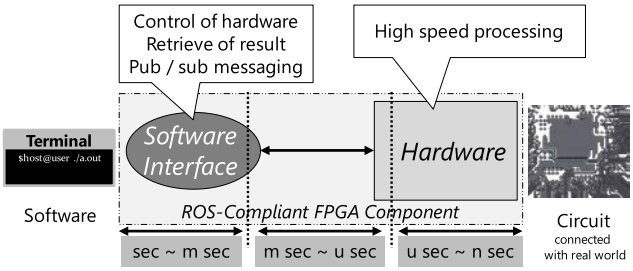


Fig. 4 Expected component latency range

- Encapsulation of FPGA circuits
- Interface between ROS software and FPGA circuits
- Interface for subscribing/publishing to a topic.

The FPGA performs all accelerated processing, and each component has two interfaces. The first one is an input interface that subscribes to topics to receive input data and is responsible for formatting the data so that it is suitable for FPGA processing and sending the formatted data to the FPGA. The second one is an output interface that receives processing results from the FPGA and is responsible for reformatting these results so that they are suitable for the ROS system and publishing them to the relevant topic.

Developers can implement any message mapping procedure they wish in these two interfaces. For example, in image processing, there are many different image data arrangements (i.e., pixel formats), e.g., RGB, BGR, or grayscale. FPGA image processing circuits could be implemented for any one of these pixel formats. In addition, ROS has a standard image type (defined in "sensor_msgs/Image.msg"). In many cases, it is therefore necessary to convert between the ROS and FPGA formats, although the conversion can either be implemented in the FPGA or in software.

3.5 Expected Latency Range in Components

The goal of introducing FPGAs into ROS systems is to accelerate processing. If the user application logic (e.g., image processing) processing time is very short, the communication latency (i.e., overhead) introduced by componentization may make the total latency worse. However, there are inevitable communication delays in any robot system, introduced by the need to have components at physically remote locations.

As shown in Fig. 4, the communication latency between remote ROS nodes ranges from several milliseconds to a second, for publish/subscribe communication among arbitrary ROS nodes in the system. Topic communication among ROS nodes is normally done over TCP/IP sockets, so real-time communication is not expected for remotely located ROS nodes. The ROS's main benefit is easy component integration for rapid prototyping; real-time communication between ROS nodes is not expected.

On the other hand, processing latencies inside components can be reduced to microseconds or nanoseconds,

owing to the programmability of FPGAs and their ability to realize arbitrary digital circuits operating at more than 100 MHz. The latency between ROS software and FPGA hardware can thus be in the range of microseconds to milliseconds.

Developers of ROS-compliant FPGA components should take the assumed latency range into account when allocating tasks to software and FPGA components.

4. Implementation of ROS-Compliant FPGA Components

This section discusses the implementation of the proposed ROS-compliant FPGA components using programmable SoCs in detail.

4.1 Target Platform

To satisfy the requirements summarized in Sect. 2.3, we focus on programmable SoCs to implement the proposed components. Programmable SoCs offer large-scale integration of an FPGA and a microprocessor on a single chip, wired so that the FPGA can be directly accessed through a bus. Thus, developers and users can both use the same method for accessing the FPGA via software.

Many recent programmable SoC products, such as the Zynq-7000 platform [30] from Xilinx, use ARM [31] processors. ROS mainly runs on Linux and can thus easily be used on ARM processors. Programmable SoCs therefore provide a good environment for ROS software to handle FPGAs. For the above-mentioned reasons, we investigate programmable SoCs as a way of using FPGAs as part of ROS systems. Portability among different FPGA vendors should, however, be considered. For example, Intel's SoC [32] is an alternative to the Xilinx Zynq.

For Requirement B in particular, ROS packages that use a programmable SoC make it possible to encapsulate the HW and SW in one package, since the communication channels between the SW and HW are fixed once the programmable SoC has been specified.

4.2 Example HW/SW Interface Implementation

In this study, Xillinux [33] is used to communicate between the FPGA logic and the ARM processor. The Xillinux platform integrates Linux (Ubuntu) and FPGA design for the Zynq-7000 series and is developed by Xillybus Ltd. User applications on Xillinux can access FPGA logic through specific device files (e.g., /dev/xillybus_write_32). Figure 5 shows the mechanism for communicating between user-defined hardware logic and the ARM processor in Xillinux. Software has FIFO access via reading/writing data from/to device files, and the FPGA reads/writes data from/to the FIFO buffer by controlling the read-/write-enable port at any given time.

To realize ROS-compliant FPGA components, HW/SW interfaces are implemented using the input/output FIFO

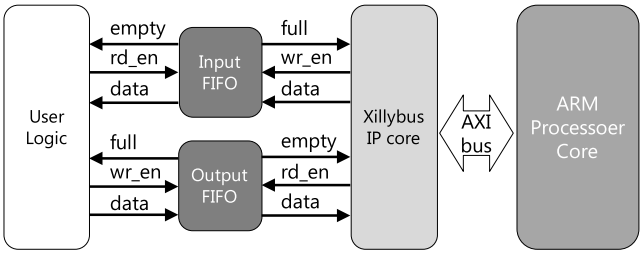


Fig. 5 Mechanism for communication between the ARM processor and an FPGA in Xillinux

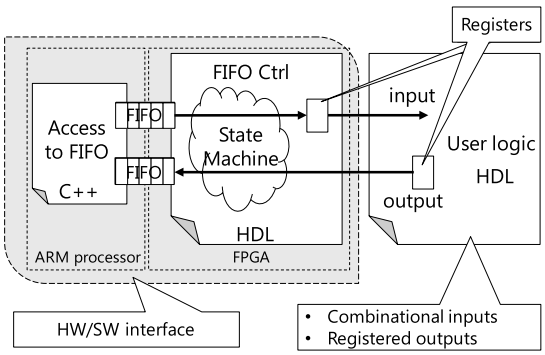


Fig. 6 HW/SW interface for ROS-compliant FPGA components using Xillinux on programmable SoCs

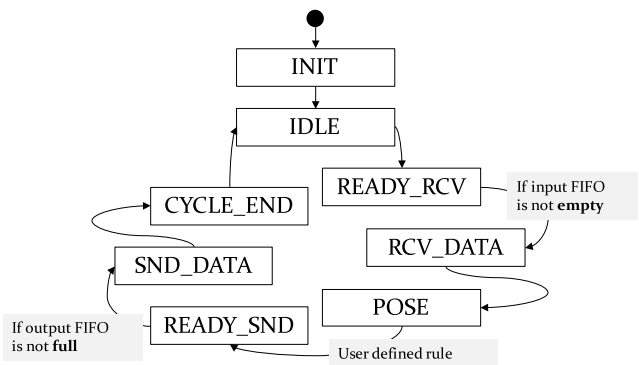


Fig. 7 Example state machine used for FIFO access (exclusive input and output access)

buffers of the Xillinux platform as communication channels, as shown in Fig. 6. Essentially, the user logic HDL input is assumed to be combinational, and the output is assumed to be registered. This assumption is useful in minimizing the effect of signal latency between the FIFO control circuit and the user logic. Figure 7 shows an example state machine used for FIFO access. Here, the RCV_DATA/SND_DATA states are repeated until all necessary data has been received/sent, and POSE waits until processing is complete. In this simple example, the FIFO input states are separated from the FIFO output states, which means that the input and output are exclusive.

With respect to portability, Xillinux has also been released for the Intel Cyclone V, although this port is no longer maintained. This means that an abstract HW/SW interface

layer (Linux device driver for the FPGA FIFO buffer) can be provided by Xillinux even when a different FPGA vendor is used. The HW/SW interface for ROS-compliant components is therefore portable in principle, as are components implemented using it.

4.3 Discussion

The implementation discussed above is somewhat naive. To achieve better FPGA performance, it is better to use stream processing to exploit pipeline parallelism [3]. For stream processing, control of the input and output FIFO buffers should be separated, and synchronization signals, indicating whether to start/stop processing, should be provided by user logic. Therefore, some rules must be added to the user HDL logic module. Sometimes, it may be straightforward to modify the user logic: when using HLS, synthesized modules often include synchronization signals that can be used to control the HW/SW interface.

To improve performance, shared memory access is important between software on the ARM processor and user logic in hardware. This can be implemented in user logic, independent of the proposed component framework.

5. Evaluation

In this section, we evaluate the performance of the proposed ROS-compliant FPGA components using two types of user logic: an image processing filter for connected-component labeling and sensor input processing to estimate attitude angle by sensor fusion.

5.1 Evaluation 1: Image Processing Filter Node

Here, we describe an implementation of image labeling based on an ROS-compliant FPGA component to investigate the issues involved in accelerating image processing in robot vision systems. We gave a partial introduction to this topic in [8].

Image labeling assigns numeric labels to groups of white pixels in binary images. Such labeling is used to measure areas, angles, and target lengths in many robotic systems, and an example labeling result is shown in Fig. 8. To produce correct results, labeling requires two steps because a simple raster scan may split a single region into several regions, a problem that is easily resolved by the second step.

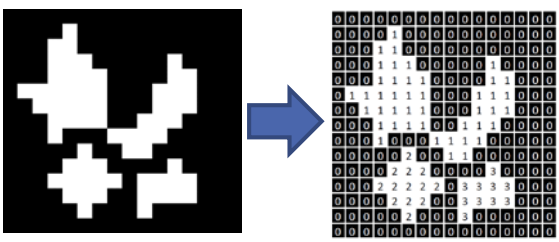


Fig. 8 Connected-component labeling of an image

This paper, however, focuses on the first step of labeling, which tends to be time-consuming in software.

Figure 10 shows a hardware block diagram. The processing target is a full HD image (1920×1080 pixels, about 2 MB). In this case, labeling is carried out line-by-line, since the block RAM of the FPGA is too small to store the entire HD image. In order to maximize the processing efficiency of the FPGA hardware implementation, we designed it to label pixels using modular arithmetic.

Table 2 shows the roles of each of the five hardware modules. First, *memory_img* stores a line of the input image. Immediately after that, the pixel data is sent to *label_generator*, which carries out labeling pixel by pixel. The labeling algorithm needs the pixel data and label numbers from the previous line and previous pixel, so two memories, *label_data0* and *label_data1*, are used. Specifically, *label_generator* writes the result for the current line to *label_data1* while reading from *label_data0*. For the next line, *label_generator* then reads from *label_data1* and writes to *label_data0*.

The labeling hardware labels pixels using modular arithmetic. Figure 9 shows a circuit diagram for *label_generator*. There are four 8-bit inputs: one, called "*New Pixel*," for the input image's pixel data and the others for previous label numbers. In addition, there is a single 8-bit output, called "*Output Label*," for the label numbers.

If the "*Reference Labels*" are all 0, the circuit outputs the previous label number incremented by 1, which is stored in the "*Current Label*" register. On the other hand, if there are any non-zero numbers in "*Reference Labels*," the minimum of these is output as the "*Output Label*" result.

The message format for this task includes the following fields:

- *int32 frame_ID*: frame number (32-bit integer)
- *int16 width*: image width (16-bit integer)
- *int16 height*: image height (16-bit integer)
- *int32[] pixels*: image pixels (32-bit integer array).

The structure of the implemented component is illustrated in Fig. 11. The labeling circuit is implemented using the FPGA part (PL) of the Zynq-7020, and the ROS topic communication is done by the ARM part (PS). These are

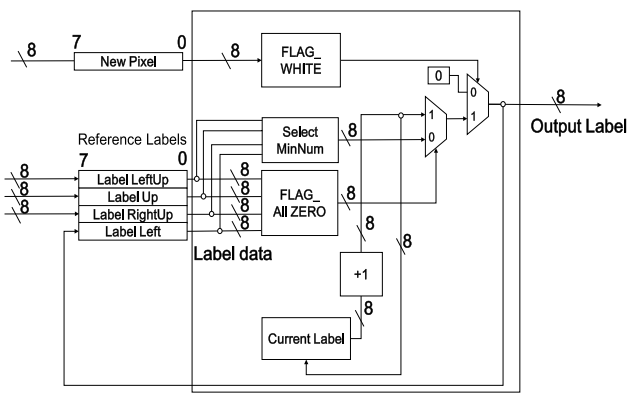


Fig. 9 Circuit for *label_generator*

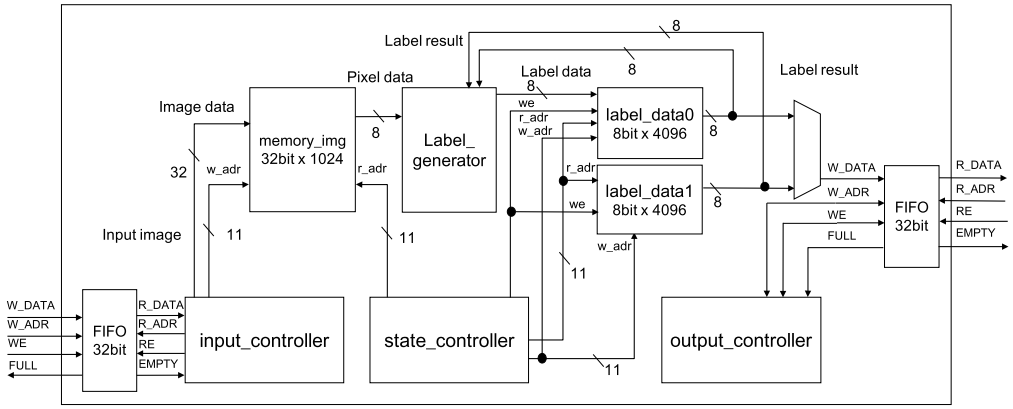


Fig. 10 Circuit block diagram for the image labeling task

Table 2 Module functions

Module	*Function*
memory_img	Save an input image line
label_generator	Label pixels
label_data0	Save a result line (label number)
label_data1	Save a result line (label number)
FIFO buffer	Buffer for input and output

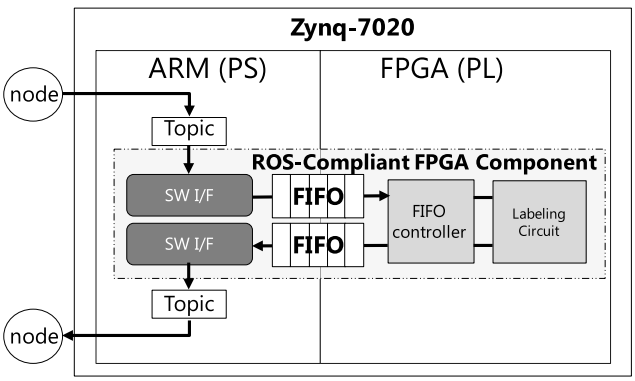


Fig. 11 ROS-compliant FPGA image processing component using a programmable SoC (Xilinx Zynq-7020)

Table 3 Hardware resource utilization (Zynq-7020)

Resource	*Utilization*
Slice Registers	4,123/106,400 (3%)
Slice LUTs	4,114/53,200 (7%)
RAM B36E1	3/140 (2%)
RAM B18E1	11/280 (3%)

connected via a FIFO buffer provided by Xillinux. Table 3 shows the hardware utilization of the FPGA.

The component was evaluated under the following three conditions:

(1) PC (SW only: PC)
(2) Software only (SW only: ARM)
(3) ROS-compliant FPGA component (ARM + FPGA).

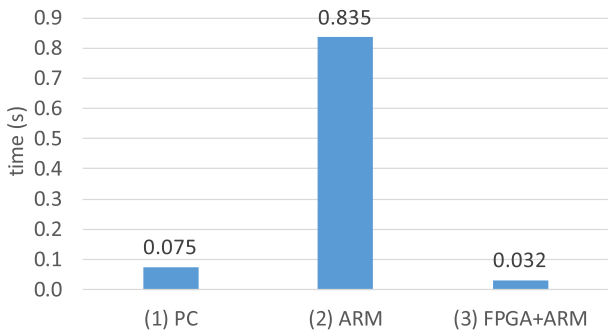


Fig. 12 Processing time for connected-component labeling (measured by gettimeofday() in software, via FIFO communication)

The environment for Condition 1 was an ordinary PC equipped with an Intel Core i7 870 @ 2.93 GHz with 16 GB of RAM running Ubuntu Linux 12.04 LTS. The environment for Conditions 2 and 3 was a ZC7Z020 (Zynq-7020, Xilinx Ltd.) on Zedboard. The ZC7Z020 is a programmable SoC equipped with an ARM Cortex-A9 @ 666 MHz and an Artix-7 FPGA on a single chip. The OS was Ubuntu 12.04 LTS (Xillinux-1.2-eval), and the operating frequency of the labeling hardware for Condition 3 was 100 MHz.

Figure 12 shows the average measured processing time for the labeling task. The input image resolution was 1920×1080 pixels, and the measurements were made using the standard C library function gettimeofday() in software and repeated 10 times. The processing time for Condition 2 (ARM only) was 835 ms, which was 11.1 times slower than the time for Condition 1 (PC). Both Conditions 1 and 2 are measures of pure-software performance. The slowdown ratio was much larger than the ratio of the CPU clock frequencies (2.93 GHz/666 MHz = 4.4 times), possibly owing to other factors (e.g., memory speed) reducing processing performance.

For Condition 3 (FPGA + ARM), the measured latency included the communication between the FPGA and ARM CPU through the FIFO buffers. In this case, the average was 32 ms per frame, with minimum and maximum times of 28 and 35 ms, respectively. The processing time for the FPGA

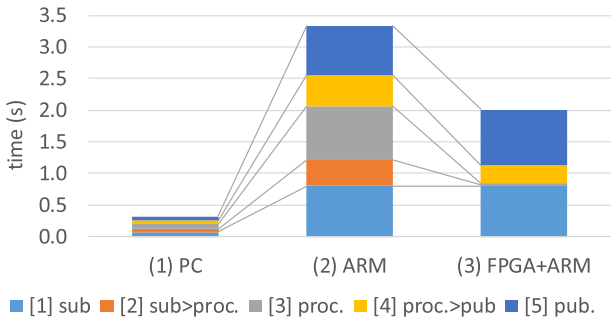


Fig. 13 Latency analysis for the ROS components

component was thus 26 times faster than that for software running on the ARM processor and even 2.3 times faster than that for the PC.

Ideally, the number of clock cycles used for processing should be 1,920 clocks per line. Owing to the overhead of changing the line, however, the period was actually 2,400 clocks, which means that each frame (1,080 lines) required 2.6 M clocks, giving a processing time per frame of 26 ms for the FPGA (1 clock is 10 ns at 100 MHz). The difference between the measured processing time (32 ms) and this calculated value (26 ms) is due to the communication time between the ARM processor and the FPGA. Even including the communication time in the processing time, the FPGA component still performed faster than the pure-software implementation.

Figure 13 shows an analysis of the measured times for the different components, broken down into the following segments:

[1] ROS node communication (Subscribe)
[2] Time from receiving data to starting processing
[3] Connected-component labeling processing
[4] Time from completing processing to publishing
[5] ROS node communication (Publish).

For Condition 3, the execution time was 1.99 s, which was about 1.7 times faster than software only with the ARM processor (3.33 s). For Conditions 2 and 3, ROS node communication consumes much of the execution time.

To investigate the feasibility of introducing FPGAs into robots to reduce power consumption, the power consumption and efficiency of the FPGA were estimated. The power supplied to the labeling hardware of the FPGA component was estimated using the XPower Analyzer included in the ISE Design Suite. The estimated total power draw was 0.33 W, composed of 0.20 W of dynamic power and 0.13 W of idle power. The measured power reported for the Zedboard is 6.1 W [11], and the power consumption of a high-performance processor is approximately 90 W [12] (Intel Core i7). The power consumption of the proposed ROS-compliant FPGA component is therefore much lower than that of a PC.

The estimated power efficiency of the FPGA component is shown in Table 4. On the basis of the measured execution times for the three conditions (PC, ARM, and FPGA + ARM), processing throughputs (frames per second) were calculated. The relative power efficiency was also calculated for ARM and FPGA + ARM, normalized to the PC value, and the resulting power efficiency for FPGA + ARM was 2.51. Compared with the improvement in processing time for connected-component labeling shown in Fig. 12, the power efficiency is not as high, owing to the communication delay caused by ROS messaging.

Table 4 Estimated power efficiency (throughput/power) of the ROS-compliant FPGA component

Environment	*Execution time (s)*	*Throughput (fps)*	*Power (W)*	*Normalized power efficiency (PC = 1)*
(1) PC	0.32	3.12	90	1.00
(2) ARM	3.33	0.30	6.1	1.58
(3) FPGA + ARM	1.99	0.50	6.4	2.51

Table 5 Robot Power consumption

Robot	*Battery power (W)*
Kobuki [33]	22
iRobot Roomba [35]	30
Husky A200 [36]	400

Table 5 shows the absolute power consumed by several household robots, all of which are at least 22 W, compared with the 6.4 W consumed by the FPGA component. The component can therefore be expected to reduce power consumption, which is important if the robots are to operate for as long as possible.

In summary, this experiment has shown that the communication latency experienced by the proposed ROS-compliant FPGA component is very large. However, it can still contribute to improving the performance of robots while maintaining low power usage.

5.2 Evaluation 2: Sensor Input Node

In order to explore another application of FPGA components in robot systems, an ROS-compliant FPGA component that combines (fuses) input from multiple sensors was evaluated. We gave a partial introduction to this topic in [13], [14].

Sensor fusion [15] is a method for obtaining highly reliable information by combining input from multiple sensors. Robot systems depend on sensor inputs to recognize their environment independently, and sensor fusion is very important for obtaining reliable measurement values.

First, we focus on an open-source implementation [37] of sensor fusion processing using the Mahony algorithm [16] for attitude angle estimation. An overview of the target hardware is given in Table 6. The target software was written in C++ and compiled in the ARMmbed [38]

Table 6 Target environment for sensor fusion processing

Processor	ARMmbed ARM STM32F401RE Nucleo board (ARM @ 84 MHz)
Motion sensor	MPU9250, InvenSense, Inc. (I/F: I2C, clock freq. 400 kHz)
Sensor fusion algorithm	Mahony [16] sensor fusion filter using an accelerometer, gyro, and magnetic compass

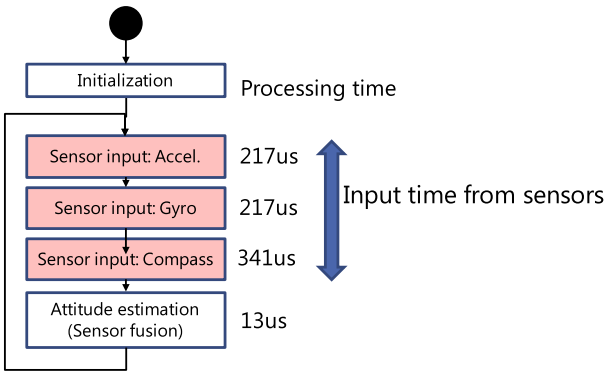


Fig. 14 Measured processing times for the sensor fusion software

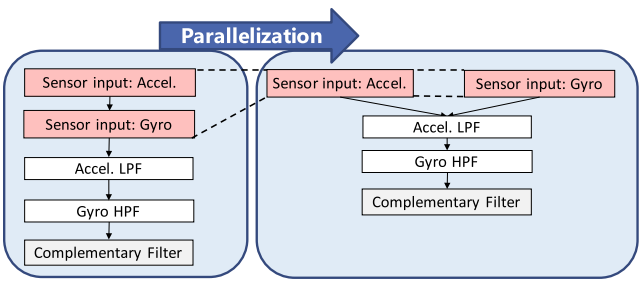


Fig. 15 Parallelization of sensor inputs

environment.

The measured processing times for the software implementation are given in Fig. 14, showing that sensor input processing occupied 98% (217 + 217 + 341 = 775 μs) of the entire main loop (788 μs). The majority of this time is consumed by communication (the I2C I/F at 400 kHz in this case). If this communication could be parallelized, the input time could therefore be reduced drastically.

In evaluating the FPGA component, a simpler complementary filter was applied to the data to obtain stable attitude angle values using two sensors operating on different principles, namely, a gyro sensor and an acceleration sensor.

The concept behind sensor input parallelization is shown in Fig. 15. FPGAs can be programmed to realize arbitrary digital circuits, and by implementing two dedicated sensor input circuit modules, sensor input processing can be fully parallelized.

Figure 16 shows the design of the sensor fusion component for attitude estimation. Two 9-axis sensor modules (MPU9250 InvenSense, Inc.) were used as a gyro sensor and an acceleration sensor. The target user logic comprised *MPU_gyro_controller*, which obtained gyro sensor values, and *MPU_accel_controller*, which obtained acceleration sensor values.

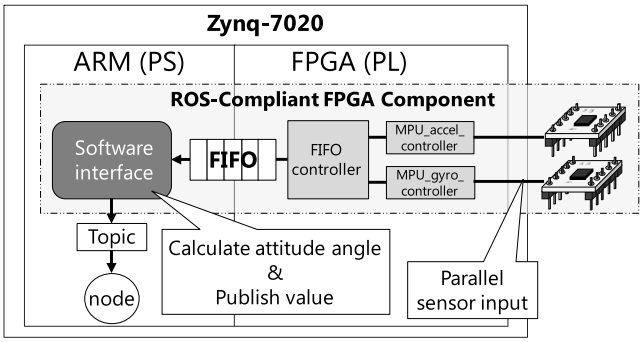


Fig. 16 ROS-compliant FPGA component for sensor fusion using two sensor inputs (Xilinx Zynq-7020, InvenSense MPU9250 x 2)

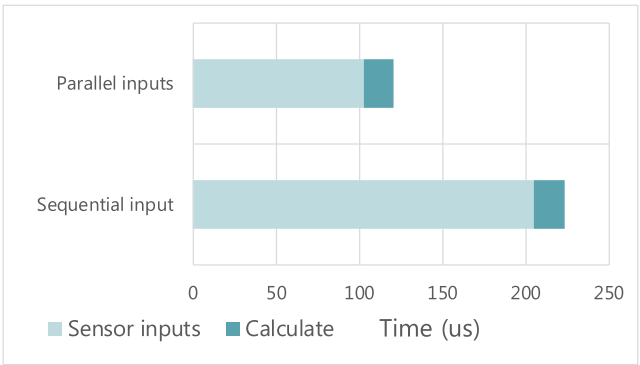


Fig. 17 Measured latency for sensor fusion processing using two sensor inputs

Table 7 Environment of the sensor fusion FPGA component

Processor	Xilinx Zynq-7020 (ARM:667MHz・FPGA:100MHz)
Motion sensor	InvenSense Inc. MPU9250 (I/F: SPI, clock freq. 1 MHz)
Sensor fusion algorithm	Complementary filter using accelerometer and gyro

The measured processing times for the component are shown in Fig. 17. Since receiving input data from sensor modules is slow, the sensor inputs were parallelized using a dedicated FPGA hardware implementation, achieving significantly lower sensor input latency (102 μs) than the pure-software implementation (204 μs).

This shows that an FPGA component for sensor fusion can process input from multiple sensors efficiently and with very low latency via parallel processing.

6. Discussion

In this section, we would like to discuss the design productivity and the limitations of the proposed ROS-Compliant FPGA component.

There are two aspects of design productivity concern-

ing the proposed ROS-compliant FPGA component. First is the design productivity of ROS system integration by using a ROS-compliant FPGA component. Development of ROS system with FPGA, which is based on the concept of ROS-compliant FPGA component, is not different from the development of normal ROS systems, which are purely implemented in software and do not use FPGAs. As defined in the Sect. 3.1, an "ROS-compliant FPGA component" is *an ROS component made from an FPGA whose functionality is equivalent to that implemented in software*. For example, if there are two ROS components: (1) "process_sw" is a pure software ROS node, (2) "process_fpga" is a ROS-compliant FPGA component which has same functionality as "process_sw" with accelerated processing time. In ROS system, it is very easy to run a ROS node by command-line or a launch file [28]. Therefore, "process_sw" and "process_fpga" can switch easily. However, it is noted that the functional equivalence must be verified in the ROS system.

The other aspect is the design productivity of ROS-compliant FPGA component itself. The development process of ROS-compliant FPGA component is complicated and requires both knowledge of FPGA and software. Therefore, the development of the ROS-compliant FPGA component may become the bottleneck process of the entire project in robot development. To solve the problem, we are also working on an automated tool for designing the proposed ROS-compliant FPGA components, called cReComp [14]. By using cReComp, experimental results show that only less than one hour is enough for novice designers to implement a ROS-compliant FPGA component into programmable SoC. The experiments of 1-day development were done in order to evaluate the design productivity. Six novice users (subjects) developed a ROS-compliant FPGA component with cReComp. As a component ultrasonic distance sensor (Parallax Inc PING) was given. The subjects were given user logic (a HDL file described control ultrasonic distance sensor) at the beginning of the experiment and they followed experimental instruction manual described by the author.

Considering from the two aspects, the proposed ROS-compliant FPGA component is effective to improve the design productivity of robot system with FPGA. Especially, if experts of FPGA design provides ROS-compliant FPGA component with very high-performance, robot developer can use the FPGA component very easily.

The limitation of ROS-compliant FPGA component is portability among FPGA devices. As long as the FPGA configuration (bitstream) generated by the component developer is applicable for another FPGA device of the user, the component can be used without any trouble. For example, however, in the case of using different external pins, the bitstream must be re-compiled using FPGA tools. As discussed in the Sects. 4.1 and 4.2, if the user circuit written in HDL follows the connection manner of HW/SW interface, component may be reused with slight modification and recompile. These kinds of issues about portability remain as a future work.

7. Related Work

A significant number of papers have reported the application of FPGAs to robots, so some examples will be discussed in this section.

In traditional component-oriented development, robots are generally equipped with computational hardware resources that consume large amounts of power [6]. With these hardware resources, current autonomous mobile robots cannot operate for more than one to three hours continuously [17], [18]. Operating longer will require high processing performance using embedded processors, rather than high-performance processors. Therefore, to save power in robots, FPGAs are an effective way to achieve high-performance processing with low power consumption.

As described previously, an effective way to employ FPGAs in robots is to develop a combined HW/SW system using a programmable SoC. This will also reduce the development cost of the HW/SW communication interface. Yanbing Li et al. have developed Nimble to support HW/SW co-design [19]. The Nimble framework generates two outputs from C code input. First, it analyzes the input and identifies loops in the code. Then, it generates a HW/SW co-system, composed of hardware that implements the loops in the original C code and a software binary that runs on a CPU.

Throughput has generally been regarded as the most important performance criterion for computing systems. However, cyber-physical systems such as robots require not only logical processing but also physical processing that is closely connected to the real world. In addition, physical processing must be parallel and low latency, since sequential processing is slow [20]. In FPGAs, when very low delay processing is required, it is effective to design at the register-transfer level using HDL.

As they allow digital logic to interact with real world interfaces, FPGAs are often used for robot manipulators that need advanced control logic. This is because most non-linear controllers need real-time mobility, which is difficult to achieve with general-purpose microprocessors [21], [22]. Autonomous fuzzy behavior control and sensor-based behaviors for self-driving cars have been implemented using FPGAs [23], both of which are needed if such mobile robots are to have human-like driving skills.

Another aspect of the application of FPGAs to robotic systems is the environment used for designing FPGA-based systems. The development of FPGA-based systems is more difficult than that of software since they must be implemented using an HDL, which is still difficult for conventional software engineers to handle. Finding ways to reduce the development cost of FPGA-based systems is therefore very important. It has previously been suggested that the development of FPGA-based systems using traditional programming languages, such as C, C++, or MATLAB, would improve developer productivity [24].

The development of sensor fusion devices using

FPGAs (Zynq-7020 programmable SoC) has also been reported [25]. In that report, fusion of two visual (image) sensors and two inertial sensors (gyro and accelerometer) was implemented and evaluated for the purpose of preprocessing sensor data for accurate visual simultaneous localization and mapping (SLAM). Since ROS is often used in SLAM applications, this would be a good target for applying our component.

There are two communication mechanisms in ROS: publish/subscribe messaging for data communication and service invocation for controlling nodes in the system. This paper has mainly focused on data communication, but control is also important for building ROS systems. Our group has previously developed a remote call mechanism for FPGAs: ORB Engine [26] is a distributed object platform that requires a broker compliant with the Common Object Request Broker Architecture (CORBA) standard. As an example, an inverted-pendulum system was built using a high-level synthesis tool that generates HDL code from pure Java code. The system's control logic was greatly accelerated by an FPGA designed using Java, without the need to write the HDL code [26].

8. Conclusion

We have proposed the concept of ROS-compliant FPGA component for easy integration of FPGAs into robot systems. In order to achieve seamless connection between the FPGA and the robot software platform (ROS), we have discussed a mapping between FPGA signal semantics and ROS messages. Using a programmable SoC, FPGA hardware processing and software on a CPU can be combined into a portable ROS package. The design of the proposed FPGA components has been described in detail, particularly the communication channels between the FPGA and the ARM processor, in terms of a state machine using FIFO communication.

Two experiments were conducted to demonstrate the execution time and power efficiency performance of the proposed component. First, we designed an ROS-compliant FPGA component for use as an image processing filter. The execution time of the proposed component was 1.99 s, 1.7 times faster than the pure-software implementation running on the ARM processor (3.33 s). The execution time mostly consisted of the time taken for publish/subscribe messaging by the ROS system. Then, the power efficiency (throughput/power) was estimated, showing that the proposed component was 2.51 times more efficient than an ordinary PC processor, despite the communication overhead.

The second experiment involved a component for sensor fusion. This example demonstrated that sensor input processing in robot software systems is slow and can be parallelized by utilizing the programmability of an FPGA. An ROS-compliant FPGA component with independent circuits for each of two sensor inputs achieved much lower latency (102 μs) for the sensor input than a system implemented purely in software (204 μs).

Two aspects of design productivity concerning the proposed ROS-compliant FPGA component were discussed. Users of ROS-compliant FPGA component can benefit from it. And the design and implementation of the FPGA component itself can be partly automated. However, the portability is limited and still remains as future work.

The large communication time overhead required for publish/subscribe messaging in the ROS system remains a significant problem for ROS-compliant FPGA component. If this overhead can be reduced, it will lead to substantial improvements in the power efficiency of FPGA components used to accelerate processing in robots.

Acknowledgments

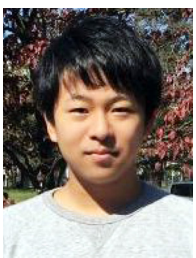
This research and development work was supported by MIC/SCOPE #152103014. The authors would like to thank Toko Ninomiya and Enago (www.enago.jp) for the English language review.

References

[1] R. Siegwart, I.R. Nourbakhsh, and D. Scaramuzza, "Introduction to autonomous mobile robots," MIT press, 2011.
[2] R.A. Brooks, "A robust layered control system for a mobile robot," IEEE J. Robot. Autom., vol.2, no.1, pp.14–23, 1986.
[3] K. Sano, W. Luzhou, Y. Hatsuda, T. Iizuka, and S. Yamamoto, "FPGA-Array with Bandwidth-Reduction Mechanism for Scalable and Power-Efficient Numerical Simulations based on Finite Difference Methods," ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol.3, no.4, Article No.21, 35 pages, 2010.
[4] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," Proc. International Conference on Field Programmable Logic and Applications, FPL2009, pp.126–131, 2009.
[5] H. Song and J.W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," Proc. 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays, pp.238–245, 2005.
[6] Robotics Society of Japan (ed.), "Robot technology," Ohmsha, Ltd, 2011 (in Japanese).
[7] N. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," ICRA workshop on open source software, vol.3, no.3.2, p.5, 2009.
[8] K. Yamashina, T. Ohkawa, K. Ootsu, and T. Yokota, "Proposal of ROS-compliant FPGA Component for Low-Power Robotic Systems - case study on image processing application," Proc. 2nd International Workshop on FPGAs for Software Programmers, FSP2015, pp.62–67, 2015.
[9] R. Mamat, A. Jawawi, N. Dayang, and S. Deris, "A component-oriented programming for embedded mobile robot software," International Journal of Advanced Robotic Systems, pp.371–380, 2007.
[10] S. Cousins, "Exponential growth of ros [ros topics]," IEEE Robot. Autom. Mag., 1(18), pp.19–20, 2011.
[11] M.M. Wirthlin and B.L. Hutchings, "Implementing highperformance, low-power FPGA-based optical flow accelerators in C," 2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors (ASAP), IEEE, pp.363–369, 2013.
[12] E.S. Chung, et al., "Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs?," Proc. 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, pp.225–236, 2010.

[13] H. Kimura, K. Yamashina, T. Ohkawa, K. Ootsu, and T. Yokota, "Speed-up of underwater robot control with FPGA component," conference paper collection of The 78th National Convention of IPSJ, Speech No.2H-03, 2016 (in Japanese).
[14] K. Yamashina, H. Kimura, T. Ohkawa, K. Ootsu, and T. Yokota, "cReComp: Automated Design Tool for ROS-Compliant FPGA Component," Proc. IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-16), 2016.
[15] L. Xiao, S. Boyd, and S. Lall, "A scheme for robust distributed sensor fusion based on average consensus," The Proc. Fourth International Symposium on Information Processing in Sensor Networks, IPSN2005, pp.63–70, 2005.
[16] R. Mahony, T. Hamel, and J.-M. Pflimlin, "Nonlinear complementary filters on the special orthogonal group," IEEE Trans. Autom. Control, vol.53, no.5, pp.1203–1218, 2008.
[17] M. Imai, M. Takahashi, T. Moriguchi, T. Okada, Y. Minato, T. Nakano, S. Tanaka, H. Shitamoto, and T. Hori, "A Transportation System using a Robot for Hospital," Journal of The Robotics Society of Japan, vol.27, no.10, pp.1101–1104, 2009.
[18] T. Suzuki, Y. Yamazaki, H. Tamukoh, and M. Sekine, "A Mobile Robot System using Intelligent Circuit in Silicon," IEICE Tech. Rep., vol.111, no.397, VLD2011-105, pp.83–88, Jan. 2012 (in Japanese).
[19] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood, "Hardware-software co-design of embedded reconfigurable architectures," Proc. 37th Annual Design Automation Conference, pp.507–512, 2000.
[20] G. Schirner, D. Erdogmus, K. Chowdhury, and T. Padir, "The future of human-in-the-loop cyber-physical systems," Computer, vol.1, pp.36–45, 2013.
[21] F. Piltan, M. Rahmani, M. Esmaeili, M.A. Tayebi, M.P.H. Cheraghi, M.R. Rashidian, and A. Khajeh, "Research on FPGA-Based Controller for Nonlinear System," International Journal of U-& E-Service, Science and Technology, vol.8, no.3, pp.11–28, 2015.
[22] Farzin, Piltan, N. Sulaiman, M.H. Marhaban, Adel, Nowzary, and Mo-stafa, Tohidian, "Design of FPGA-based Sliding Mode Controller for Robot Manipulator," International Journal of Robotics and Automation, pp.173–194, 2011.
[23] S. Sánchez-Solano, A.J. Cabrera, I. Baturone, F.J. Moreno-Velo, and M. Brox, "FPGA implementation of embedded fuzzy controllers for robotic applications," IEEE Trans. Ind. Electron., vol.54, no.4, pp.1937–1945, 2007.
[24] T.-H.S. Li, S.-J. Chang, and Y.-X. Chen, "Implementation of Human-Like Driving Skills by Autonomous Fuzzy Behavior Control on an FPGA-Based Car-Like Mobile Robot," IEEE Trans. Ind. Electron., vol.50, no.5, pp.867–880, 2003.
[25] J. Nikolic, et al., "A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM," 2014 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp.431–437, 2014.
[26] T. Ohkawa, D. Uetake, T. Yokota, K. Ootsu, and T. Baba, "Reconfigurable and Hardwired ORB Engine on FPGA by Java-to-HDL Synthesizer for Realtime Application," Proc. 4th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART 2013), pp.45–50, 2013.
[27] Open Source Robotics Foundation: http://www.osrfoundation.org/
[28] ROS Wiki: http://wiki.ros.org
[29] ROS message: http://wiki.ros.org/msg
[30] Zynq-7000 All Programmable SoC, Xilinx: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html
[31] ARM Ltd.: http://www.arm.com/
[32] Intel SoCs, Intel: https://www.altera.com/products/soc/overview.html
[33] Xillybus: http://xillybus.com/
[34] Yujin Garage: http://garage.yujinrobot.com/
[35] iRobot: https://www.irobot-jp.com/product/
[36] nihon binary: http://www.nihonbinary.co.jp/
[37] https://developer.mbed.org/users/onehorse/code/MPU9250AHRS/
[38] ARMmbed: https://www.mbed.com/

Takeshi Ohkawa received the B.E., M.E., and Ph.D. degrees in electronics from Tohoku University in 1998, 2000, and 2003, respectively. He has been engaged in research on dynamically reconfigurable FPGA device and system at Tohoku University since 2003. He joined the National Institute for Advanced Industrial Science and Technology (AIST) in 2004 and started research on distributed embedded systems. He has been working in TOPS Systems Corp on heterogeneous multicore processor design since 2009. He joined Utsunomiya University in 2011 as an assistant professor. His current research interests are the design technology of an FPGA to realize low-power robots and vision systems. He is a member of IEEE and ACM. He is also a member of IEICE, IPSJ, and RSJ of Japan.



Kazushi Yamashina received the B.E. and M.E. degrees in information systems science from Utsunomiya University in 2015 and 2017, respectively. He joined Hitachi Ltd. in 2017 and started research and development on robotics for industry. He is a member of SCI of Japan.

Hitomi Kimura received the B.E. degree in information engineering from Utsunomiya University in 2016. She joined East Japan Institute of Technology Co. Ltd in 2016.

Kamemitsu Ootsu received his B.S. and M.S. degrees from the University of Tokyo in 1993 and 1995, respectively, and later he obtained his Ph.D. in information science and technology from the University of Tokyo in Japan. From 1997 to 2009, he was a research associate, and then in 2009, he became an associate professor at Utsunomiya University. His research interests are in high-performance computer architecture, multi-core multi-thread processor architecture, binary translation, and run-time optimization. He is a member of IPSJ.

Takashi Yokota received his B.E., M.E., and Ph.D. degrees from Keio University in 1983, 1985, and 1997, respectively. He joined Mitsubishi Electric Corp. in 1985 and was engaged in several research projects in special-purpose, massively parallel and industrial computers. He was engaged in research and development of a massively parallel computer RWC–1 at Real World Computing Partnership as a senior researcher from 1993 to 1997. From 2001 to 2009, he was an associate professor at Utsunomiya University. Since 2009, he has been a professor at Utsunomiya University. His research interests include computer architecture, parallel processing, network architecture, and design automation. He is a member of IPSJ, IEICE, ACM, and the IEEE Computer Society.