cReComp : Automated Design Tool for ROS-Compliant FPGA Component

Kazushi Yamashina†, Hitomi Kimura†, Takeshi Ohkawa‡, Kanemitsu Ootsu‡ and Takashi Yokota‡
Graduate School of Engineering, Utsunomiya University
Yoto 7–1–2, Utsunomiya, Tochigi, Japan, 321–8585
†{kazushi, hitomin}@virgo.is.utsunomiya-u.ac.jp
‡{ohkawa, kim, yokota}@is.utsunomiya-u.ac.jp

***Abstract*—Autonomous mobile robots require high-performance computation to meet variety of requirements of functions, such as sensing, intelligent image processing and controlling actuators. We focus on FPGA as a hardware platform for autonomous mobile robot system. However, a FPGA-based system is not effective in development cost, since it requires HDL-based design whose productivity is relatively low. In order to solve this problem, we have already proposed a design principle of ROS-compliant FPGA component, which is effective in easy integration of a FPGA device into any robot system. Although it allows ROS-based software to access easily to hardware circuitry in FPGA, high development cost of HDL-based circuitry still remains as a large problem. So, in this paper, we propose cReComp which is an automated design tool to improve productivity of ROS-compliant FPGA component. cReComp generates codes of interface software and hardware automatically. We evaluate cReComp from two major aspects: improvements in design productivity, and operation speed of generated FPGA components by 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. Furthermore, our results reveal that generated FPGA component operates 1.85 times faster than the original software-based component.**

***Index Terms*—FPGA, Programmable SoC, Component oriented development, ROS, ROS-compliant FPGA component, Robot**

I. Introduction

To distribute computation combined with various sensors is one of the key issues to realize autonomous and intelligent mobile robots [1] [2]. These intelligent robots are used for disaster relief, nursing care for handicapped and elderly persons and so on. Although, they are required to perform intelligent image processing and calculations using enormous sensor data, the robots are under a limitation of power supply because of battery operation. And, it is difficult for mobile robots to utilize sufficient computation power from conventional microprocessors. Therefore, it is necessary to realize a high performance processing at low power in the robots. To satisfy the requirements of autonomous mobile robots, we focus on FPGA (Field Programmable Gate Array) which would be able to realize high performance processing at low power [3] [4].

FPGA contributes to speeding up processing in fields such as network packet processing and image processing since FPGA can realize specialized circuits of parallel processing with bitwise optimization [5] [6]. However, developing the whole system on FPGA is not a practical way, since the development based RTL (Registrer Transfer Lebel) with HDL (Hardware Description Language) is very difficult and complicated.

To reduce the development cost, HW/SW co-design (Hardware/Software cooperative design) is a promising approach [7] [8] [9]. Reducing the amount of hardware by limiting hardware implementation in necessary part can improve design productivity. On the other hand, robotics depends not only on HW/SW but also on various expertise like electricity, mechanics, and so on, so robot systems are more difficult to develop [10].

We have proposed ROS-compliant FPGA component [11] which is the component-based design principle for easy integration of FPGA into any robot system. ROS-compliant FPGA component offers an effective and transparent interface between hardware circuits in FPGA and software on CPU. Typically, component-oriented development is a well-known method to improve productivity of robot system [10]. ROS (Robot Operating System) is one of component-oriented developments and is becoming the mainstream as a software platform for development of robotic application software. ROS provides communication library and build system for robotic application software. ROS-compliant FPGA component can realize easy integration of FPGA into any robot system by componentization of a HW/SW co-system therefore improve processing performance at low power [11].

In this paper, we propose cReComp which is an automated design tool to improve productivity of ROS-compliant FPGA component. An overview of ROS-compliant FPGA component and one of the major problems of its design flow are described in Section II. In Section III, we propose an automated design tool named **cReComp** (creator for Reconfigurable Component) for ROS-compliant FPGA component. Section IV shows productivity of designing a ROS-compliant FPGA component using cReComp, and the performance of ROS-compliant FPGA component is evaluated. Then, we survey previous research and describe effectiveness of our proposal in Section V. Finally, this paper concludes in Section VI.

II. ROS-COMPLIANT FPGA COMPONENT

A. ROS Overview

ROS [12] is an open source project by OSRF (Open Source Robotics Foundation) [13]. It is not an operating system but a software platform for component-oriented development of robotic application software on an OS. Communication library and a build system for robotic application software are provided in ROS. The operating system on which ROS can run is mainly Ubuntu Linux.

Each software component in ROS corresponds to a software package in ROS system [12]. It is packaged as a function such as image processing filter, camera input, motor control and so on. There are many open sources and reusable software packages available on the ROS official web [17]. So, ROS is becoming the mainstream in robot development [14].

In ROS, major communication models among the software components are publish / subscribe messaging and service invocation (Figure 1). Each software component, named *node*, communicates by sending and/or receiving messages (data) through communication channel, named *topic*. A *node* has two roles; *publisher* and *subscriber*. A *publisher* can publish messages to any *topic*, and the messages are queued in the *topic*. A *subscriber* can subscribe to any *topic*. When a *topic* is updated, *subscribers* receive messages from the *topic* and run pre-defined callback tasks.

In ROS, a *node* need not know its peer node to communicate with. Since the nodes are connected loosely, a component can be added to or removed from the system easily. These loose binding among components improves the productivity in designing and in debugging the robot software.

B. ROS-compliant FPGA component

ROS-compliant FPGA component is a ROS component which includes FPGA. FPGA circuits are less reusable than software. One of the reasons is that interface of FPGA circuits is defined as input/output logic signals at bit level or at an address mapped register level. Therefore, it is necessary to implement the interface of FPGA circuits not at logic signal level or address mapped register level but with publish / subscribe messaging at application level.

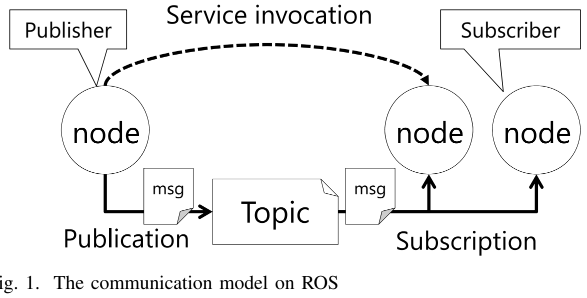


Fig. 1. The communication model on ROS

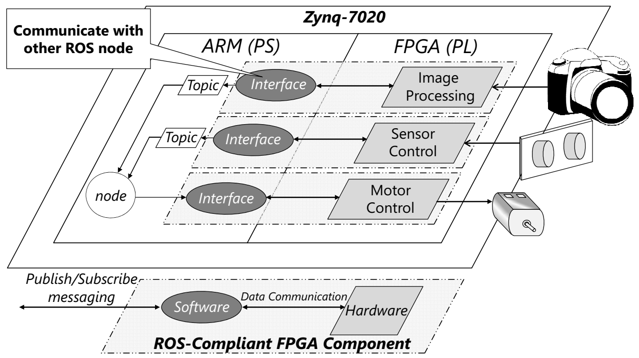


Fig. 2. An example of system configuration with ROS-compliant FPGA components

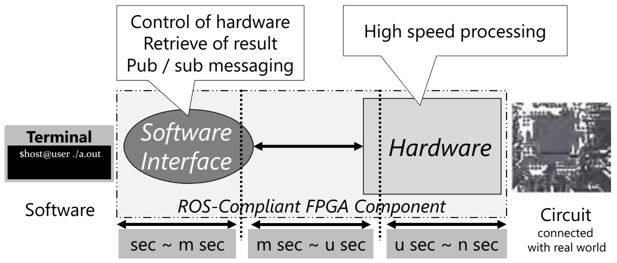


Fig. 3. The task partitioning of ROS-compliant FPGA component

In order to realize application level interface for FPGA, we have proposed ROS-compliant FPGA component [11], which is a ROS component including FPGA that complies with ROS-style publish / subscribe messaging completely. It can be replaced with a pure software component and has an interface which can communicate with software from the hardware (FPGA). The interface has two parts: an interface software on CPU and an interface circuit on a FPGA. Figure 2 shows an example of a ROS system configuration with ROS-compliant FPGA components. In this study, we use Xilinx Zynq [15] as a platform for ROS-compliant FPGA components. Zynq is a Programmable SoC equipped with ARM processor (PS: Processing System) and FPGA logic (PL: Programmable Logic). Each ROS-compliant FPGA component has an interface for communication between PS and PL. PS is connected with PL thorough AXI bus of ARM. The interface software running on PS is in charge of publish/subscribe messaging. By wrapping FPGA logic with software, the component can communicate with other ROS nodes, so it makes the integration of an FPFA into a ROS system easily.

To design a high performance ROS-compliant FPGA component, task assignment between hardware and software is important. Componentization for publish / subscribe messaging at application level and the communication between hardware and software cause delay. In our previous study [11], to transfer a full HD image, it takes 6 milliseconds between FPGA and ARM processor on the Zynq-7020, while it takes approximately 0.8 seconds for publish / subscribe messaging. From the study, we learned that the task assignment is one of the most important issues. That is, FPGA can deal with

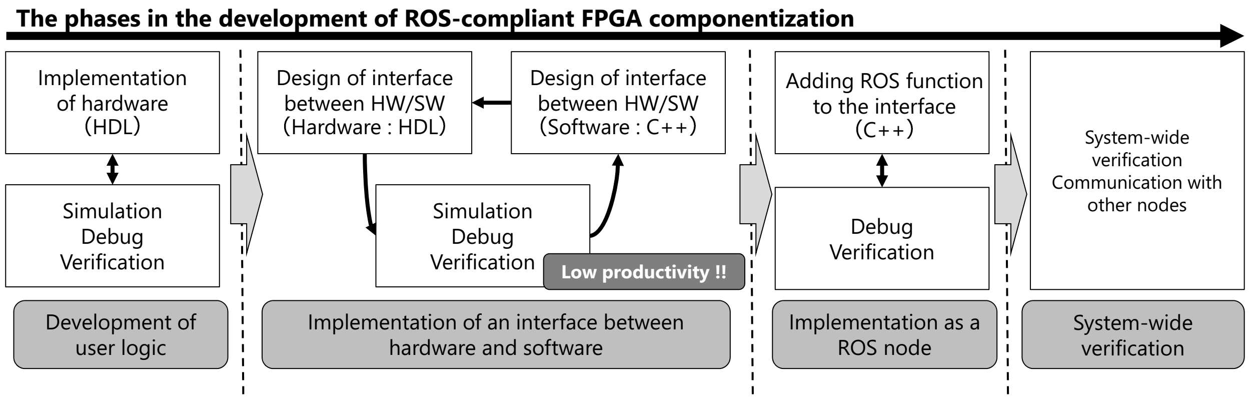

The phases in the development of ROS-compliant FPGA componentization
Implementation of hardware (HDL)
Simulation Debug Verification
Design of interface between HW/SW (Hardware : HDL)
Design of interface between HW/SW (Software : C++)
Simulation Debug Verification
Low productivity !!
Adding ROS function to the interface (C++)
Debug Verification
System-wide verification Communication with other nodes
Development of user logic
Implementation of an interface between hardware and software
Implementation as a ROS node
System-wide verification

Fig. 4. The conventional development of ROS-compliant FPGA component in HW/SW co-design

tasks which take from nanoseconds to microseconds, however, software cannot deal with such high-speed tasks. As shown in Figure 3, a role of software is to control the hardware and to retrieve the processing results of the hardware tasks. According to these design principle, ROS-compliant FPGA component can achieve both high performance and reusability.

To demonstrate and share the effectiveness of the ROS-compliant FPGA components in the ROS-community, we have already published two ROS-compliant FPGA components [16] at ROS Wiki [17].

C. Problem in the conventional development of ROS-compliant FPGA component

A problem is its development cost of implementing ROS-compliant FPGA component. Figure 4 shows the problem of conventional development flow of ROS-compliant FPGA component in HW/SW co-design.

In the componentization of processing using a FPGA, firstly, target processing of componentization is determined. Then it is implemented in hardware. In this paper, the target processing of hardware is named *user logic*. It is necessary to implement interface circuit for the communication between the processor and the FPGA (user logic). Furthermore, to implement interface software, a developer needs to design how to access FPGA through the interface circuit. The implementation of ROS-compliant FPGA component is finally completed by appending functions of ROS to interface software.

The development of the interface costs huge due to debugging and implementation on a programmable SoC. The componentization for the easy integration of user logic (FPGA) into a robotic system requires high development costs. Therefore, the development productivity is poor.

III. Proposal of automatic design tool for componentization : cReComp

In this section, we propose a tool named **cReComp (creator for Reconfigurable Component)** to solve the problem explained in Section II-C. cReComp is an automated design tool which contributes to improve productivity of ROS-compliant FPGA component. With cReComp, a developer can implement ROS-compliant FPGA component, only by describing user logic (HDL files) and configuring how to access FPGA logic from ARM processor. The input files to cReComp which should be described by developer are an user logic file and a configuration file. Then cReComp componentizes the user logic on an FPGA automatically and it generates the interface circuit and software for communication between ARM processor and the user logic. To realize the automated generation in cReComp, the requirements of cReComp are as follows.

- Interface generation model corresponding to the input files
- Design of the communication logic between HW and SW
- Target model of user logic
- Describing the configuration of communicational transaction

So, following section describes more details about these requirements.

A. Interface generation model of cReComp

One of the most important matters of cReComp is the interface generation model. Figure 5 shows that of cReComp. The automatic generation of ROS-compliant FPGA component needs the configuration for how to access FPGA logic from ARM processor. In this study, we define **scrp (Specification for cReComp)** style description as an input file for cReComp. Scrp is a configuration format which decides how to access FPGA (the target user logic) from ARM processor. The developer just describes the user logic and scrp file, without implementing any of interface for communication between hardware and software. By inputting these files, cReComp generates both *interface circuit* which is an HDL file and *interface software* which is a C++ file.

B. Communication between FPGA and ARM processor

In implementation of ROS-compliant FPGA component on programmable SoC (ARM processor and an FPGA), interface circuit for the communication is needed to access from software to FPGA logic. In order to implement communication

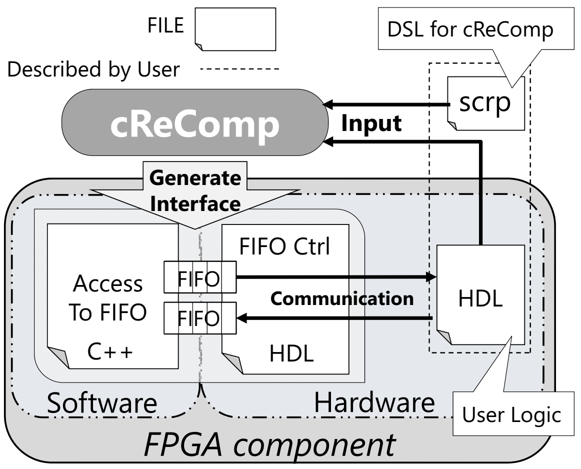


Fig. 5. The interface generation model of cReComp

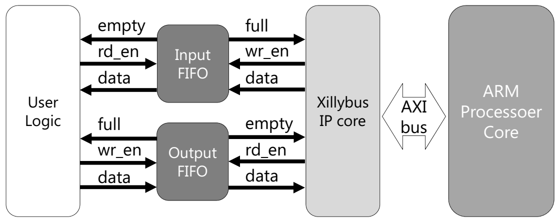


Fig. 6. The communication between ARM processor and an FPGA in Xillinux

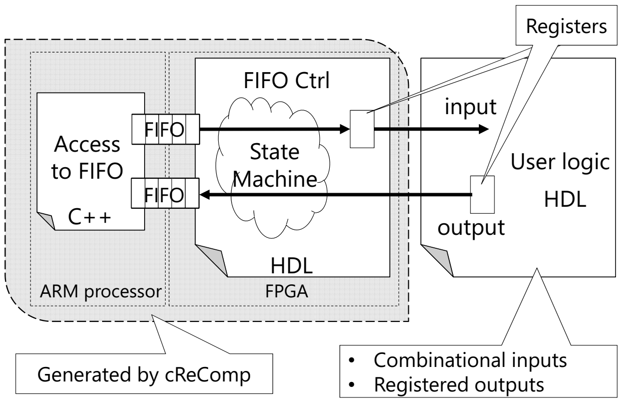


Fig. 7. The model of the target user logic

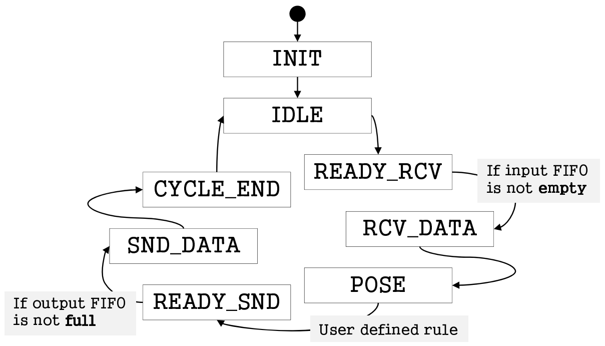


Fig. 8. The state machine for control FIFO buffers

logic between ARM processor and FPGA logic, we use Xillinux, released by Xillybus Ltd [18]. Xillinux has been used in several research projects [19] [20], and is a platform specialized for programmable SoC. In Xillinux, Linux OS (Ubuntu) runs on ARM processor. Software on ARM processor can read/write data from/to FPGA logic as a device file that corresponds to FIFO buffers (Figure 6). So, developers can connect user logic to the FIFO buffers of Xillinux. Two FIFO buffers 32- and 8-bits are prepared; the former is for data transfer, the latter is for debug. In ROS-compliant FPGA component, these two FIFO buffers are used for input/output of FPGA. Therefore, ARM processor and FPGA logic can read/write from/to FIFO buffers at any time.

C. Target user logic and generation of state machine

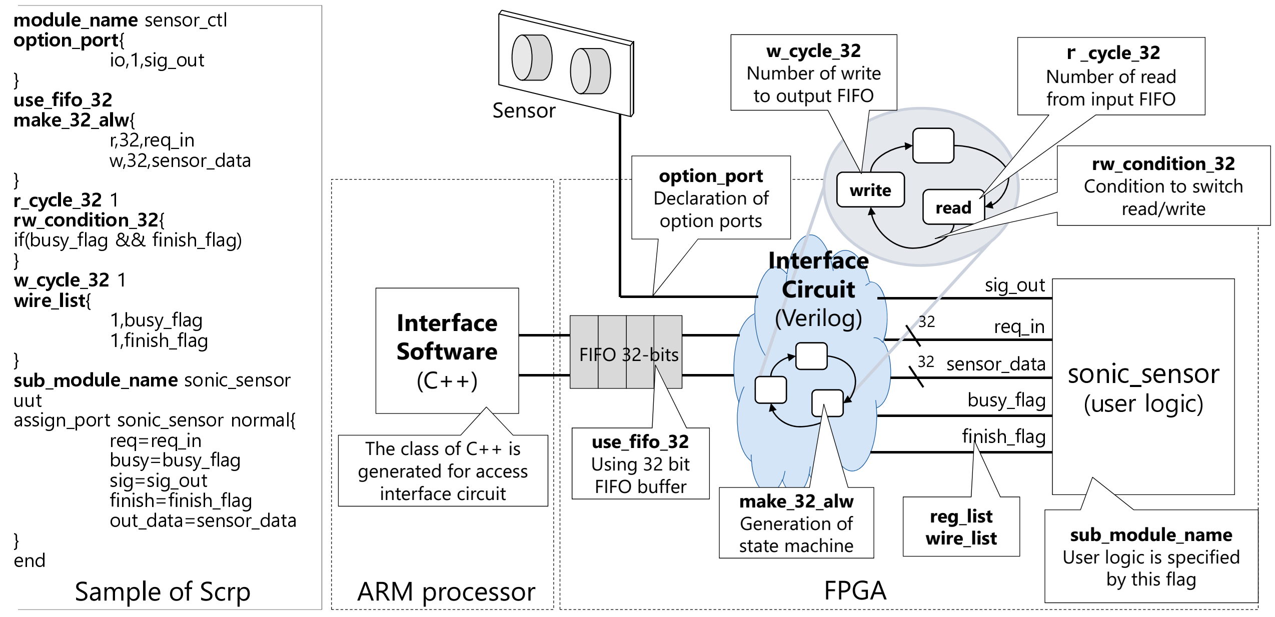
Figure 7 shows target user logic model for input of cReComp. Target user logic has combinational input ports and registered output ports. So, cReComp generates state machine in the *interface circuit* for communication between user logic and ARM processor. As shown in figure 8, the *FIFO Ctrl* module is in the state of `READY_RCV` after reset. When ARM processor inputs data to input-FIFO buffer, the register in FIFO Ctrl module receives the data from the input-FIFO buffer and the data is passed to the input ports of user logic in the state of `RCV_DATA`. After receiving some pre-defined amount of data, the state moves to `POSE`. The condition of state transition from `POSE` to `READY_SND` can be defined in the scrp file. When the user logic outputs data to ARM processor, the output port of user logic is directly connected to output-FIFO buffer and the data is output to ARM processor in the state of `SND_DATA`.

D. Definition of transaction by Scrp

cReComp needs a scrp file for defining transactions of communication between hardware and software. Figure 9 shows an example of HW/SW system whose accessing method of FPGA from software is defined by scrp file. Each element to be configured is named **flag**, and the component is generated automatically by defining each flag. Table I shows flags for 32-bits FIFO buffer only and there are same flags for 8-bits FIFO buffers, marked "*". In addition, a template of scrp is generated by using an option command ("`−s`") of cReComp. The information of input/output ports in the target user logic is automatically reflected in the scrp template by specifying a name of user logic at command line. Therefore it is not necessary for any developer to describe HDL for the communication between software and hardware.

IV. Development of component with cReComp

Figure 10 shows the proposed development flow of ROS-compliant FPGA component with cReComp. In the conventional development flow (Figure 4), it is necessary to implement the interface to communicate between hardware and software. In the development with cReComp, the development process is reduced greatly since cReComp can generate a component automatically by describing scrp file



```
module_name sensor_ctl
option_port{
        io,1,sig_out
}
use_fifo_32
make_32_alw{
        r,32,req_in
        w,32,sensor_data
}
r_cycle_32 1
rw_condition_32{
if(busy_flag && finish_flag)
}
w_cycle_32 1
wire_list{
        1,busy_flag
        1,finish_flag
}
sub_module_name sonic_sensor
uut
assign_port sonic_sensor normal{
        req=req_in
        busy=busy_flag
        sig=sig_out
        finish=finish_flag
        out_data=sensor_data
}
end
```

Sample of Scrp
Sensor
w_cycle_32
Number of write to output FIFO
r _cycle_32
Number of read from input FIFO
option_port
Declaration of option ports
write
read
rw_condition_32
Condition to switch read/write
Interface Circuit (Verilog)
Interface Software (C++)
FIFO 32-bits
sig_out
32
req_in
32
sensor_data
busy_flag
finish_flag
sonic_sensor (user logic)
The class of C++ is generated for access interface circuit
use_fifo_32
Using 32 bit FIFO buffer
make_32_alw
Generation of state machine
reg_list
wire_list
sub_module_name
User logic is specified by this flag
ARM processor
FPGA

Fig. 9. An example of HW/SW system whose accessing method of FPGA from software is defined by scrp file

TABLE I
DEFINITION OF EACH FLAG IN SCRP

Flag	Definition
module_name	Name of *interface circuit*
option_port	Declaration of option ports
use_fifo_32*	Use 32 bit FIFO buffer
make_32_alw*	Generate state machine
r_cycle_32*	Number of read from input FIFO
rw_condition_32*	Condition to switch read/write
w_cycle_32*	Number of write to output FIFO
reg_list	Declaration of option register
wire_list	Declaration of option wire

*) There are same flags for 8-bits FIFO buffers.

and defining signal assignments for user logic. The component generataion is executed only by running a command `./cReComp FileName.scrp`.

Following subsections show two evaluations; *Evaluation 1* the design productivity of cReComp and *Evaluation 2* the performance of ROS-compliant FPGA component developed with cReComp.

A. Evaluation 1 : design productivity

In order to evaluate effect on the design productivity by using cReComp, experiments of 1-day development were done. 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 [22]. Figure 11 shows the system configuration of sensor control component. In this system, interface software (C++) receives sensor value which is obtained by `sonic_sensor.v` and publishes the value to another ROS node. The experiment flow details were shown bellow.

1) Componentization of user logic with cReComp
   - a) Installation of tool environment
   - b) Description of scrp file
   - c) Componentization
2) Unit-level verification of the component
   - d) Synthesis of hardware
   - e) Compile of software
   - f) Verification on FPGA
3) Integration of the component into ROS system and verification at system level
   - g) Initial tasks of ROS development
   - h) Coding software as a ROS node
   - i) Verification on ROS system

In addition, the subjects' experience in HW/SW co-design is shown bellow.

- Development with FPGA : from nothing to 3 years
- Development with C++ : from 1 to 6 years
- Use of Linux : from 1 to 3 years

All the subjects finished whole process of experiment in three hours. Figure 12 shows the difficulty rated in five stages (5: Very easy, 4: Easy, 3: Normal, 2: Difficult and 1: Very difficult) and the average of elapsed time. In 1) componentization of user logic with cReComp, a) and c) were rated *5 very easy* mostly and they were finished in 2 minutes. On the other hand, the rating of b) was *3.7 slightly easy* (maximum : 5, minimum : 2) and b) took 17 minutes. Therefore b) took the longest time in the three processes a) to c). The componentization is *very easy* since the components are generated automatically only by executing a command. We

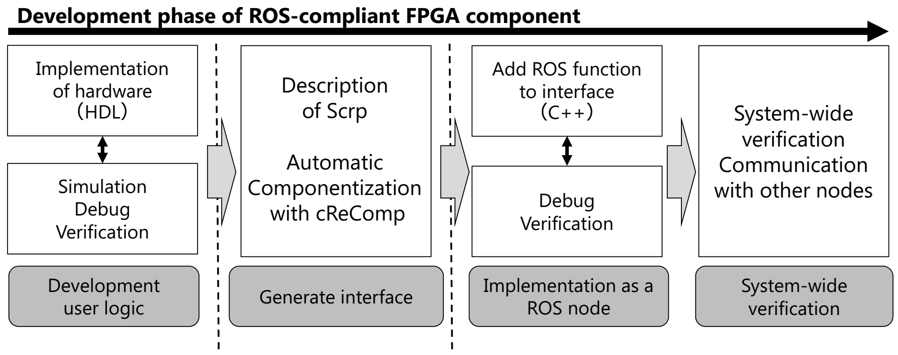

Development phase of ROS-compliant FPGA component
Implementation of hardware (HDL)
Simulation Debug Verification
Description of Scrp
Automatic Componentization with cReComp
Add ROS function to interface (C++)
Debug Verification
System-wide verification Communication with other nodes
Development user logic
Generate interface
Implementation as a ROS node
System-wide verification

Fig. 10. Our proposed development flow with cReComp

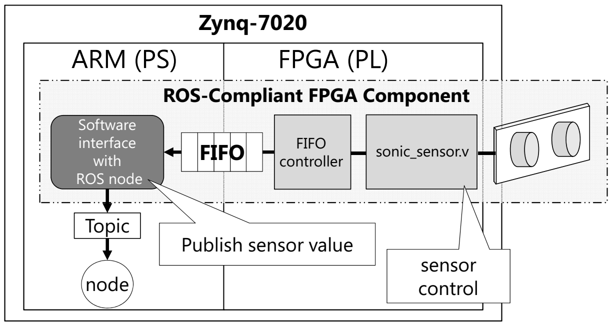


Fig. 11. The system configuration of the ultrasonic sensor control component for the development experiment by the subject (novice developer)

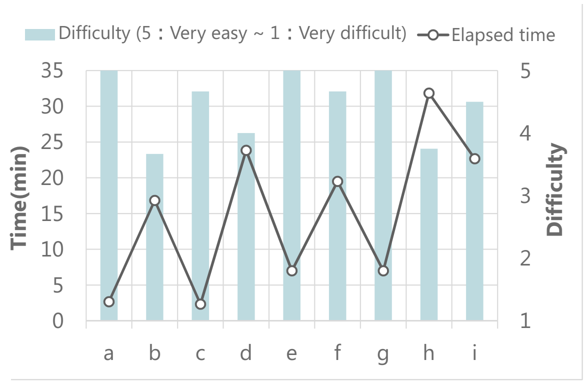


Fig. 12. The difficulty rating by the subject and the elapsed time for each step of the development of ROS-compliant FPGA component

expected that the description of scrp file is very easy for the subjects to understand, however, the result was as low as *3.7 slightly easy*. We guess the main reason is that understanding meaning of flag in scrp has taken some times. Even so, the subjects could complete the componentization within an hour without any experience of the development with FPGA.

In the 2) unit level verification of the single ROS-compliant FPGA component, d) was rated *4 Easy* and it took 23 minutes. In this process, besides the synthesis of hardware, the subjects connected the user logic and FIFO control module generated by cReComp to API provided by Xillybus. Firstly, instance of user logic was described to the top module of Xillinux. In addition, pins were assigned for the sensor input/output. The process of pin assignment is important in the development of FPGA, however, the pin allocation is different among FPGA devices. This lowers the development efficiency. f) took 19 minutes including debug of the ROS-compliant FPGA component. We found out from our observation the subjects tended to make mistakes in description of scrp file, pin assignment and ports definition of user module HDL.

In the 3) Integration of the component into ROS system and verification at system level, the process g) contained creation of workspace and package template. h) was rated *3.8 slightly easy* and it took more than 30 minutes. The process h) contained adding some functions for initialization of ROS-node, publication and subscription to software generated by cReComp. cReComp also generates a C++ class with some member functions which access to FIFO buffer. In the development of ROS, the main programming language is C++ or Python. So, generated software can be used immediately. For example, to obtain data from user logic, their description at C++ is `msg.sensor_data = obj.get_sensor_ctl_32().`

As a result, the subjects can complete their componentization within an hour without any experience of the development with FPGA.

B. Evaluation 2 : Performance

The performance of ROS-compliant FPGA component which is developed with cReComp is evaluated [23]. Target user logic estimate attitude angle by sensor fusion. Sensor fusion [24] is a method which obtains highly reliable information from variety of combination of plural sensors.

In this evaluation, attitude angle is estimated by data from a gyro sensor and an acceleration sensor. Complementary filter is applied to the data to obtain stable values of attitude angle. Figure 13 shows the configuration of component of attitude angle estimation.

The sensors used in the development are two modules of MPU9250 InvenSense, Inc. as a gyro sensor and an acceleration sensor. The target user logics are

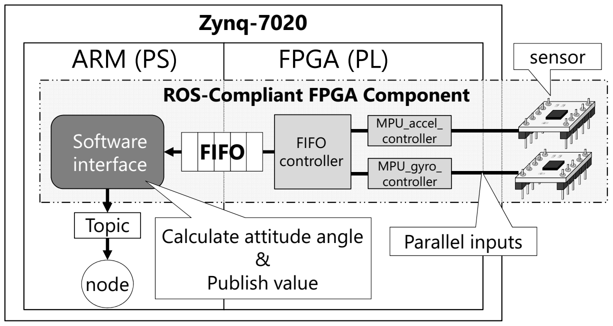


Fig. 13. Configuration of component of the attitude angle estimation by sensor fusion

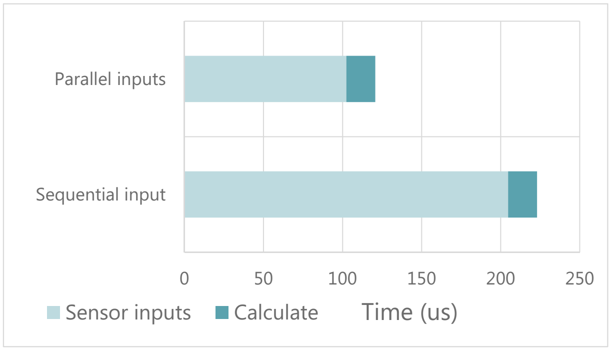


Fig. 14. Elapsed time of the attitude angle estimation

`MPU_gyro_controller` which obtains values from the gyro sensor and `MPU_accel_controller` which obtains values from the acceleration sensor. They are componentized with cReComp. It takes long to obtain sequentially data from a sensor module by software. To solve this problem, we parallelized the inputs from the sensors by implementing special hardware on FPGA. So, this component can achieve very low latency of sensor input and improve total time for sensor fusion than a component implemented by software. And from other ROS-nodes, the component can obtain data from sensors at any time.

Figure 14 shows elapsed time of the attitude angle estimation. The *parallel inputs* indicates the processing time of the ROS-compliant FPGA component with parallel sensor inputs implemented on FPGA. On the other hand, the *sequential input* indicates the processing time of a ROS-node with software. The total time of attitude angle estimation in the ROS-compliant FPGA component with parallel inputs from two sensors is 1.85 times faster (120us) than that of software with sequential input from single sensor (223us).

As a result, ROS-compliant FPGA component accomplished speeding up of sensor fusion, while maintaining the accuracy of the attitude angle estimation with software.

V. Related Work

In traditional component-oriented development, in general, a robot is quipped with hardware resources to compute at large power consumption [25]. Even with hardware resources, a current autonomous mobile robot can not operate more than one to three hours continuously [26], [27]. To operate it longer, high performance processing is required with equipping an embedded processor rather than equipping high performance one. So, for power saving of a robot, it is effective to use FPGA in order to realize high performance processing at low power.

As described previously, to employ FPGA in a robot, it is effective to develop HW/SW co-system on a programmable SoC. In addition, the development cost of the interface for the communication among hardware and software should be reduced. Yanbing Li et al. have developed *Nimble* as a support for HW/SW co-design [28]. Nimble is a framework which generates two outputs by inputting C code. First, Nimble analyzes C code of input and declares loops in the code. Second, Nimble generates HW/SW co-system composed of hardware which has loops in the original C code, and a binary (software) which runs on a CPU.

Throughput has been regarded as the most important performance criteria in computing system generally. However, in cyber-physical system like robots, there are not only logical processing but also physical processing which are connected to the real world closely. And, physical processing requires to be dealt in parallel and low-delay since sequential processing of software takes a lot of time [29]. In FPGA, when very low-delay processing is required, it is effective to design with RTL using HDL. A previous research realizes that non linear controller needs real time mobility operation which is hardly realized by general purpose microprocessors[30], [31]. Therefore, a supporting tool is required to develop a cyber-physical system which is integrated with physical processing of hardware and sequential processing of software. In previous research [32], Derler et al. proposed a development environment, *METRO II*. In METRO II supports heterogeneous devices like multi-core processor, digital signal processor and FPGAs.

cReComp supports programmable SoC and development of cyber-physical system which can deal with all processing on a chip. By supporting HW/SW (FPGA logic and embedded processor) co-system, cReComp can realize high performance and low-delay. In other words, cReComp generates the interface of HW/SW co-system which can be connected to the real world automatically. Therefore, it is possible that physical processing connected to the real world closely can be controlled from a software while maintaining low-delay.

VI. Conclusion

In this paper we proposed an automated design tool, cReComp, in order to generate ROS-compliant FPGA component. Firstly, we analyzed requirements for cReComp in Section III. We designed a model of interface generation in cReComp, the developer just describes the user logic and scrp file. By inputting these files, cReComp generates both interface circuit which is an HDL file and interface software which is a C++ file. As a communication channel between FPGA and ARM processor, a FIFO buffer of Xillybus/Xillinux is employed on Xilinx Zynq platform. cReComp generates logic and state machine to connect user logic with the FIFO buffers. We also defined a format of the scrp file which represents rule of communication between hardware and software.

cReComp contributes to improve the productivity of ROS-compliant FPGA component. The evaluation 1 shows that any developer in the experiment can complete the componentization within an hour, without experience of the development with FPGA. In the evaluation 2, the total time of attitude angle estimation in the ROS-compliant FPGA component with parallel input of two sensors is 1.85 times faster (120us) than that of software with sequential input of single sensor (223us). As a result, ROS-compliant FPGA component accomplished speeding up of sensor fusion, while maintaining the accuracy of the system with software.

Currently, cReComp only supports a user logic model written in pure HDL. By supporting various types of user logic (like codes generated by high level synthesis and so on), it is expected that cReComp can improve design productivity of FPGA in ROS system drastically.

Acknowledgment

This research and development work was supported by the MIC/SCOPE#152103014.

References

[1] Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D., "Introduction to autonomous mobile robots," MIT press, 2011.

[2] Brooks, Rodney A. ",A robust layered control system for a mobile robot" IEEE Journal of Robotics and Automation, Vol.2, pp.14-23, 1986.

[3] Kentaro Sano,Wang Luzhou, Yoshiaki Hatsuda, Takanori Iizuka and Satoru 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, DOI:10.1145/1862648.1862651 (35 pages), 2010.

[4] Li, F., Lin, Y., He, L., and Cong, J., "Low-power FPGA using pre-defined dual-Vdd/dual-Vt fabrics," In Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays (pp. 42-50). ACM, 2004.

[5] Asano, S., Maruyama, T., and Yamaguchi, Y., "Performance comparison of FPGA, GPU and CPU in image processing", In Proceedings of International Conference on Field Programmable Logic and Applications, FPL2009, pp. 126-131, 2009.

[6] Song, H., and Lockwood, J. W., "Efficient packet classification for network intrusion detection using FPGA", In Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays, pp. 238-245, 2005.

[7] Schirner, G., Gerstlauer, A., and Dmer, R., "System-level development of embedded software," In Proceedings of the 2010 Asia and South Pacific Design Automation Conference (pp. 903-909). IEEE Press., 2010.

[8] Li, Y., Callahan, T., Darnell, E., Harr, R., Kurkure, U., and Stockwood, J., "Hardware-software co-design of embedded reconfigurable architectures," In Proceedings of the 37th Annual Design Automation Conference (pp. 507-512). ACM, 2000.

[9] Kirovski, D., Lee, C., Potkonjak, M., and Mangione-Smith, W., "Application-driven synthesis of core-based systems," In Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design (pp. 104-107). IEEE Computer Society, 1997.

[10] Robotics Society of Japan (ed.), "Robot technology", Ohmsha, Ltd, 2011.

[11] Kazushi Yamashina, Takeshi Ohkawa, Kanemitsu Ootsu and Takashi Yokota : "Proposal of ROS-compliant FPGA Component for Low-Power Robotic Systems - case study on image processing application -", Proceedings of 2nd International Workshop on FPGAs for Software Programmers, FSP2015, pp. 62-67, 2015.

[12] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... and Ng, A. Y., "ROS: an open-source Robot Operating System," In ICRA workshop on open source software, Vol. 3, No. 3.2, p. 5, 2009.

[13] Open Source Robotics Foundation : http://www.osrfoundation.org/.

[14] Cousins, S., "Exponential growth of ros [ros topics]," IEEE Robotics and Automation Magazine, 1(18), pp.19-20, 2011.

[15] Zynq-7000 All Programmable SoC, Xilinx : http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[16] OpenReroc : https://github.com/Kumikomi/OpenReroc

[17] ROS Wiki : http://wiki.ros.org

[18] Xillybus : http://xillybus.com/

[19] Lin, Z., and Chow, P., "Zcluster: A zynq-based hadoop cluster," In proceedings of International Conference on Field-Programmable Technology (FPT) 2013, IEEE, pp.450-453, 2013.

[20] Sumarudin, A., Adiono, T., and Putra, W. P., "Flexible and reconfigurable system on chip for wireless sensor network," In proceedings of International Conference on Information Technology Systems and Innovation (ICITSI), IEEE, pp.230-234, 2014.

[21] cReComp : https://github.com/kazuyamashi/cReComp.git

[22] Instruction manual of the cReComp experiment: http://kazuyamashi.github.io/exp/

[23] Hitomi Kimura, Kazushi Yamashina, Takeshi Ohkawa, Kanemitsu Ootsu and Takashi 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.

[24] Xiao, L., Boyd, S., and Lall, S., "A scheme for robust distributed sensor fusion based on average consensus", The Proceedings of Fourth International Symposium on Information Processing in Sensor Networks, IPSN2005, pp. 63-70, 2005.

[25] Mamat R., Jawawi, A., Dayang, N., and Deris, S., "A component-oriented programming for embedded mobile robot software", International Journal of Advanced Robotic Systems, pp. 371-380, 2007.

[26] Michita Imai, Masaki Takahashi, Toshiki Moriguchi, Takuya Okada, Yuuichirou Minato, Tsuyoshi Nakano, Shoji Tanaka, Hideo Shitamoto and Toshio Hori, "A Transportation System using a Robot for Hospital", Journal of The Robotics Society of Japan, vol. 27, no. 10, pp. 1101-1104, 2009.

[27] Takuya Suzuki, Yusaku Ymazaki, Hakaru Tamukoh, and Masatoshi Sekine, "A Mobile Robot System using Intelligent Circuit in Silicon" Technical Report of IEICE, vol. 111, no. 399 (2012).

[28] Li, Y., Callahan, T., Darnell, E., Harr, R., Kurkure, U., and Stockwood, J., "Hardware-software co-design of embedded reconfigurable architectures", In Proceedings of the 37th Annual Design Automation Conference, pp. 507-512, 2000.

[29] Schirner, G., Erdogmus, D., Chowdhury, K., and Padir, T., "The future of human-in-the-loop cyber-physical systems", Computer, vol. 1, pp. 36-45, 2013.

[30] Piltan, F., Rahmani, M., Esmaeili, M., Tayebi, M. A., Cheraghi, M. P. H., Rashidian, M. R., and Khajeh, A., "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.

[31] Farzin, Piltan., N, Sulaiman., M, H, Marhaban., Adel, Nowzary., and Mostafa, Tohidian., "Design of FPGA-based Sliding Mode Controller for Robot Manipulator", International Journal of Robotics and Automation, pp. 173-194, 2011.

[32] Derler, Patricia, Edward A. Lee, and Alberto Sangiovanni Vincentelli., "Modeling cyber-physical systems", Proceedings of the IEEE, vol. 100, no. 1, pp. 13-28, 2012.