# A SPACEWIRE EXTENSION FOR DISTRIBUTED REAL-TIME SYSTEMS

**Session: SpaceWire Networks & Protocols**

**Long Paper**

Yusuke Murata, Takuma Kogo and Nobuyuki Yamasaki

*Department of Computer Science, Graduate School of Science and Technology,*

*Keio University*

*3-14-1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa 223-8522 Japan*

*E-mail: murata@ny.ics.keio.ac.jp, kogo@ny.ics.keio.ac.jp, yamasaki@ny.ics.keio.ac.jp*

## ABSTRACT

In this paper, we propose a real-time extension scheme for SpaceWire. We designed and implemented the proposed real-time SpaceWire function on a Dependable Responsive Multithreaded Processor I (D-RMTP I) SiP (System-in-Package) for parallel/distributed real-time control, and evaluated the basic performance of the proposed SpaceWire network.

## 1. Introduction

In real-time systems, every real-time task has a time constraint including a deadline or a cycle. The time constraint is guaranteed by real-time scheduling algorisms. Almost all real-time scheduling algorithms for single/multi-core processors are based on pre-emption and the estimation of the worst-case execution time (WCET)[2]. Distributed real-time scheduling algorithms are being investigated by extending these algorithms. Here, pre-emption is achieved by context switches in processors. In order to employ the distributed real-time scheduling algorithms, networks are also required to be pre-emptive. Pre-emption on the networks can be achieved by overtaking prioritized packets at each node. Hence, we propose a packet overtaking scheme for SpaceWire [1].

The WCET, which is one of the most important requirements for real-time scheduling algorithms, is estimated by analyzing a program in non-distributed systems. In distributed systems, the estimation of the worst-case network latency is also required. The network latency depends on the size of a packet. Since the size of packets is not fixed on a SpaceWire network, we divide a SpaceWire packet into fixed size flits. Pre-emption of packets and the estimation of the worst-case network latency are realized by overtaking prioritized packets and the fixed size flits respectively, so that distributed real-time scheduling algorithms [6][7] especially for Responsive Link [5], which is the ISO/IEC 24740 real-time communication standard, can be applied to the SpaceWire with our proposed scheme.

We designed and implemented the proposed SpaceWire on a Responsive Multithreaded Processor [3], which is a system-on-chip for parallel/distributed real-time control, and evaluated the basic performance of the proposed SpaceWire network.

## 2. Real-Time Communication

Many real-time schedulers, based on classical Earliest Deadline First (EDF) and Rate Monotonic (RM) algorithms [4], have been proposed. Most real-time operating systems based on such real-time schedulers pre-empt and execute tasks in order of priority at every tick. Figure 1 shows a sample scheduling based on the EDF. The scheduling policy of the EDF is that earlier the deadline, higher the priority.

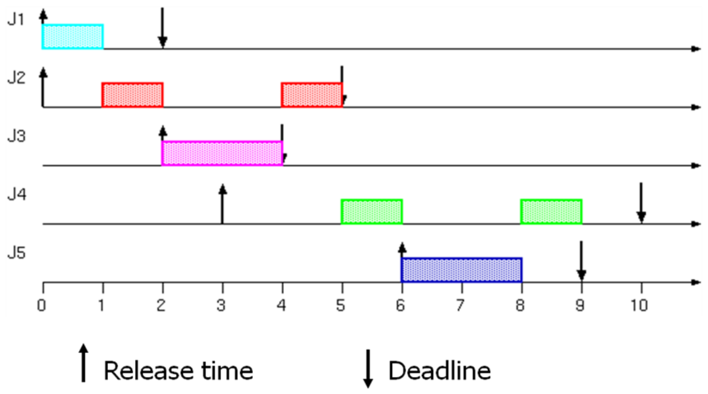
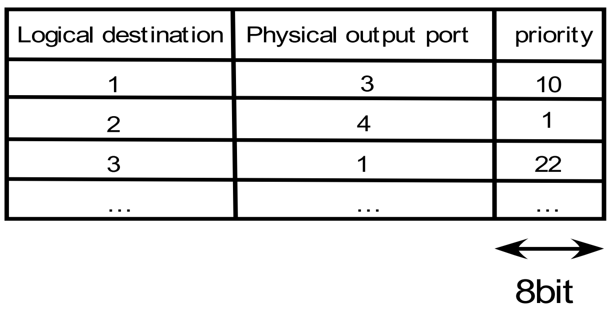


**Figure 1 An EDF sample schedule**

Pre-emptive processing (context switching) is required to realize real-time processing. Similarly pre-emptive communication that requires packet overtaking is needed to realize real-time communications. Therefore our goal is to realize a real-time communication architecture that can optimally do packet overtaking on SpaceWire, so that each control node can send and receive packets with suitable priority given by the real-time schedulers.

## 3. Packets Overtaking Scheme

We propose a packet overtaking scheme to realize pre-emption on a SpaceWire network. First we add a priority field to a routing table of SpaceWire. Figure 2 shows a routing table format for proposed real-time protocol. The routing table consists of a logical destination, a physical output, and a priority. The priority is used for packet overtaking which is realized by SpaceWire router switches with prioritized virtual channels.



| Logical destination | Physical output port | priority |
|---|---|---|
| 1 | 3 | 10 |
| 2 | 4 | 1 |
| 3 | 1 | 22 |
| … | … | … |

**Figure 2 A routing table format with priority**

Almost all real-time scheduling algorithms assume the known WCET. The network latency depends on the size of a packet and its blocked time. Since the size of packets is not fixed on a SpaceWire network, we divide a SpaceWire packet into fixed size flits that are control codes and data characters. We define a new control code in order to divide a packet for packet overtaking. Figure 3 shows a real-time control code format.

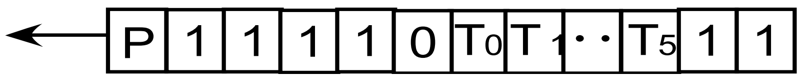


**Figure 4 A real-time control code format**

Figure 4 shows a packet overtaking scheme in a SpaceWire router switch. If a packet is overtaken in the midstream of a packet, the control code for the high priority packet is added to the head of the high priority packet. After transferring the high priority packet, the control code for the low priority packet is added to the head of the low priority packet, and the low priority communication restarts. The SpaceWire router switch keeps the destination address of the pre-empted packet to transfer the packet. Even if a low priority packet is divided by a high priority packet, the low priority packet that consists of discontinuous flits can be sent to the correct destination node.

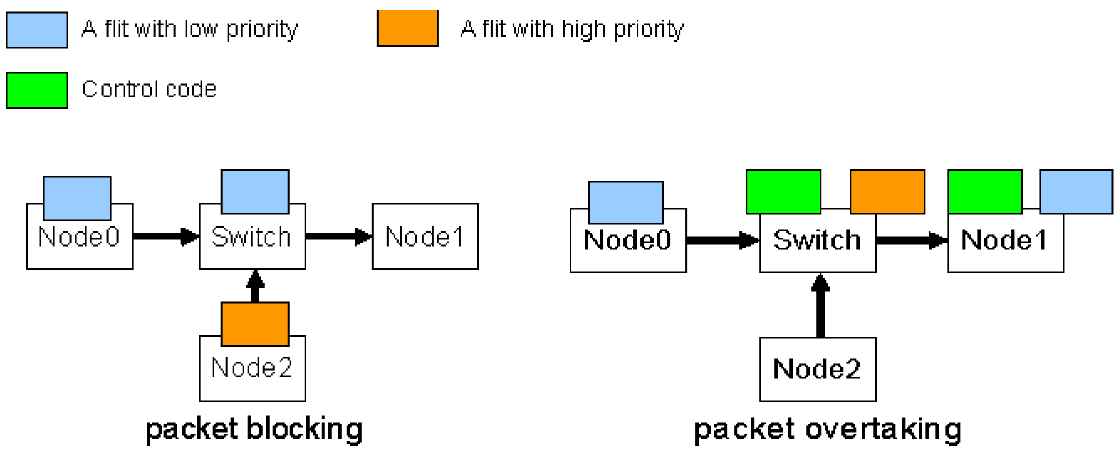


**Figure 4 A packet overtaking scheme**

Figure 5 shows prioritized router architecture for a real-time SpaceWire network. Since a low priority packet can be divided into a few groups of flits due to the flit-level pre-emption based on priority in the real-time SpaceWire network, a mechanism that merges the divided packet that consists of the groups of flits into the original packet is required. In case of flit-level pre-emption based on priority, when a router restarts sending a low priority packet that was pre-empted by a higher priority packet to the corresponding virtual channel of the next router, the virtual channel should be specified correctly. In order to solve this problem while keeping compatibility of SpaceWire networks, our real-time SpaceWire scheme uses a control code of the SpaceWire protocol for flit overtaking. The proposed real-time control code indicates the correspondence relation between the virtual channel of the current router and the virtual channel of the next router. When a pre-emption occurs at a router, the router switch generates a real-time control code for the high priority packet, and the router switch sends the real-time control code to the next router. Then the router switch sends the flits of the high priority packet. At the same time, the router controller generates a real-time control code for the low priority packet that also indicates the virtual channel of the low priority packet of the next router. The real-time control code for the high priority packet is sent to the next router and switches the virtual channel of the next router correctly. Specifically the high priority packet is buffered to another virtual channel for the high priority packet at the next router. When the router switch finishes sending the high priority packet, the router restarts to send the low priority packet. At this time, the real-time control code for the low priority packet that indicates a restart of the low priority packet is sent to the next router to switch the virtual channel correctly. Communication of the low priority packet restarts, so that the low priority packet is sent and buffered to the reserved virtual channel for the low priority packet at the next router.

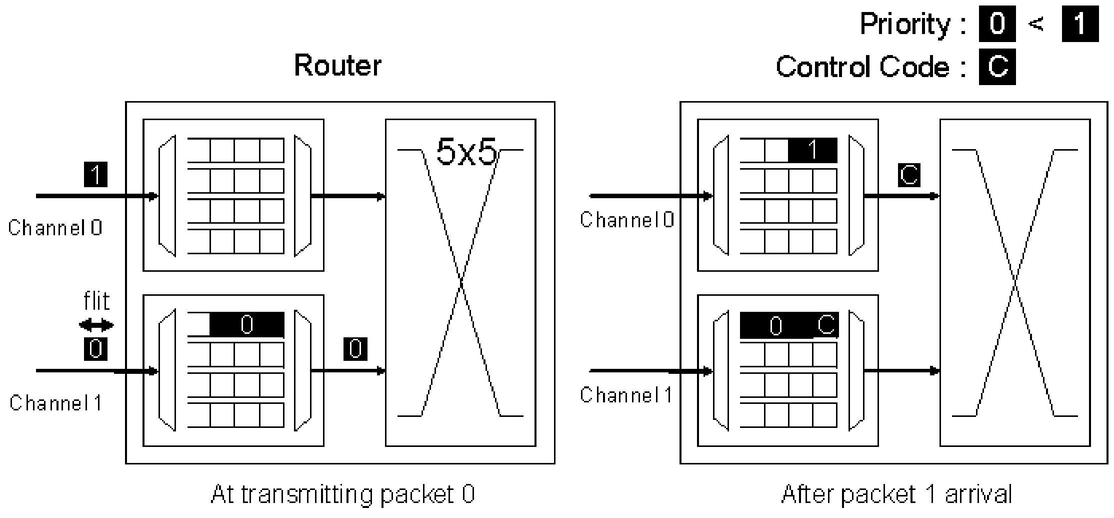


**Figure 5 A prioritized router**

## 4. Implementation

We have implemented the proposed SpaceWire router switch in the Spartan3e FPGA on the D-RMTP I (Dependable Responsive Multithreaded Processor I) SiP (System-in-Package) as shown in Figure 6. The D-RMTP I SiP, which size is 3x3cm, integrates the D-RMTP I, four DDR SDRAMs, two flash memory chips, Ethernet phy, an FPGA (Spartan3e), etc. The D-RMTP I, which size is 10 x 10mm, is a SoC (System-on-Chip) for distributed real-time control, which integrates a real-time processing core (RMT PU: 8-way prioritized SMT with 2D vector units), SRAM, DDR SDRAM IF, PCI-X, SPI, IEEE1394, Ethernet, PWMs, encoders, UART, etc into an ASIC chip as shown in Figure7. All D-RMTP I functions except the SpaceWire router switch are integrated into the D-RMTP I chip. The FPGA (Spartan3e XC3S500E) is exclusively used for the SpaceWire router switch, so that the protocol of the real-time SpaceWire can be changed easily.

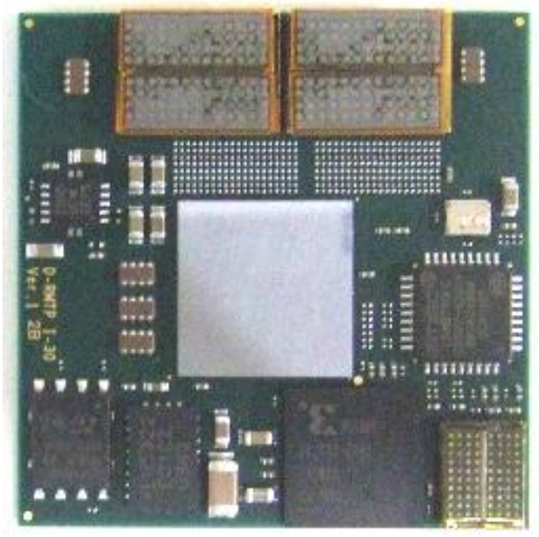


Figure 6 Photo of D-RMTP I SiP

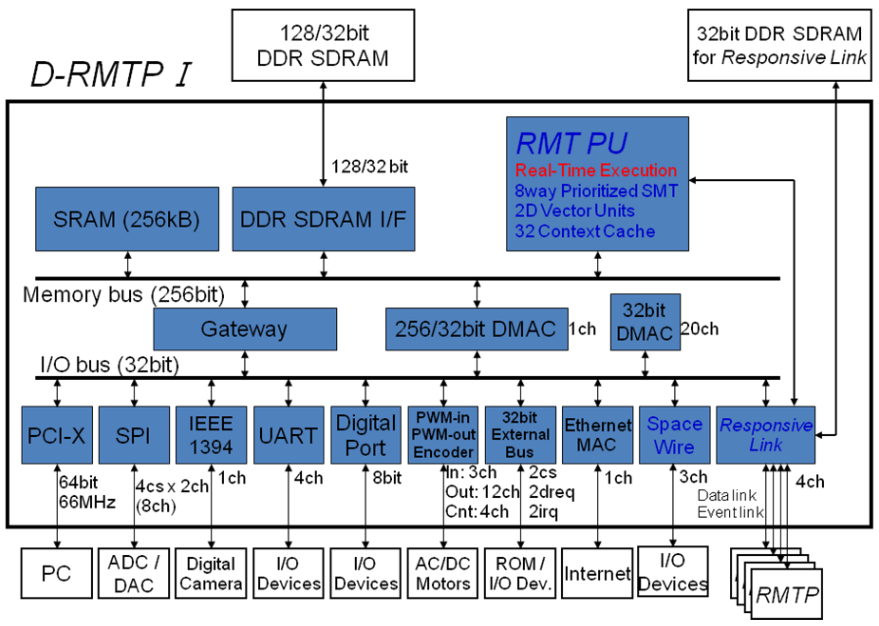


Figure 7 Block diagram of D-RMTP I SiP

## 5. Evaluation

We implemented the SpaceWire router switch written by Verilog HDL on the FPGA. We evaluated the SpaceWire network by RTL simulation using NC-Verilog. Figure 8 shows a network topology for the evaluation. Each node generates 64-byte packets, which destination addresses are changed at random, under random uniform traffic. The average latency of packets and the maximum latency of packets were measured, while the network utilization was changed.

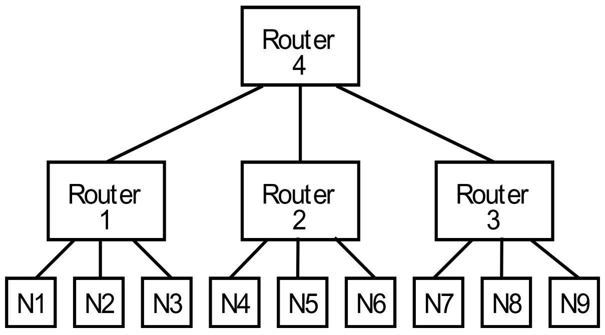


**Figure 8 Network topology**

Figure 9 shows a basic preliminary performance of the proposed SpaceWire network, which shows a network latency without priority. While the network utilization becomes higher, the average latency of a packet does not increase so much, but the maximum latency of a packet increases.

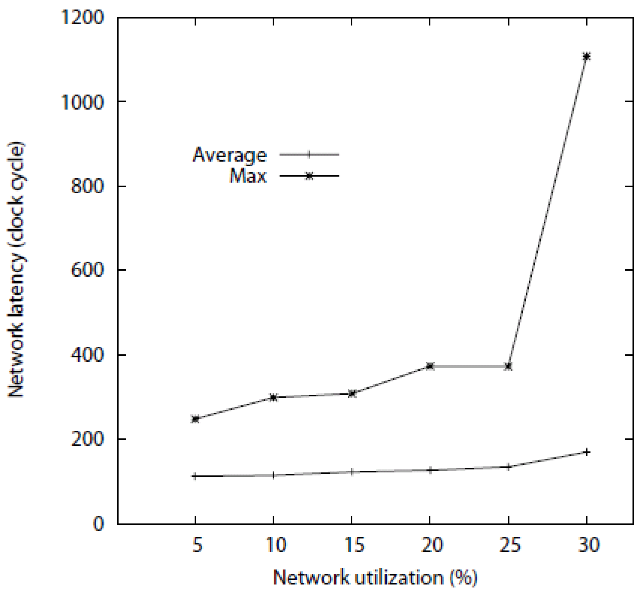


**Figure 9 Network latency (base line)**

6. Conclusion and future work

We proposed a real-time extension scheme for SpaceWire. We designed and implemented a packet overtaking function for real-time SpaceWiare networks. We designed and implemented the proposed SpaceWire router switch on the D-RMTP I SiP for parallel/distributed real-time control. We evaluated the basic performance of the proposed SpaceWire network. Now we are going to measure the real-time performance of the proposed real-time SpaceWire network, including the maximum latency and the average latency of each priority packet by RTL simulation. We will also measure them by using several D-RMTP I SiPs connected by the proposed SpaceWire network.

## 7. Acknowledgement

This research was supported by JST, CREST. This work was also supported in part by Grant in Aid for the Global Center of Excellence Program for "Center for Education and Research of Symbiotic, Safe and Secure System Design" from the Ministry of Education, Culture, Sport, and Technology in Japan.

## 8. References

1. ECSS, “ECSS-E-ST-50-12C, SpaceWire-links, nodes, routers and networks”.

2. Christian Fraboul, Thomas Ferrandiz, Fabrice Frances, “A method of computation for worstcase delay analysis on SpaceWire networks”, IEEE Symposium on Industrial Embedded Systems, Switzerland, July 8-10, 2009.

3. Nobuyuki Yamasaki, “Responsive Multithreaded Processor for Distributed Real-Time Systems”, Journal of Robotics and Mechatronics, 2005.

4. Liu, C.and Layland, J, “Scheduling algorithms for multiprogramming in a hard real-time environment”, Journal of the ACM, Vol.20, pp.46-61, 1973

5. Nobuyuki Yamasaki, “Responsive Link for Distributed Real-Time Processing” The 10th International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, pp. 20-29, January, 2007.

6. Shinpei Kato, Yuji Fujita and Nobuyuki Yamasaki, “Periodic and Aperiodic Communication Techniques for Responsive Link”, 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp135-142, August 24-26, 2009.

7. Yuji Fujita, Shinpei Kato and Nobuyuki Yamasaki, “Real-Time Communication and Admission Control over Responsive Link”, The IASTED International Conference on Parallel and Distributed Computing and Networks, pp. 131-138, Innsbruck, Austria, February 12-14, 2008.