Design and Implementation of Responsive Processor for Parallel/Distributed Control and Its Development Environments

Nobuyuki Yamasaki

* Kao University
3-14-1, Higashi, Kohoku-ku, Yokohama, Kanagawa 223-8522, Japan
** National Institute Advanced Industrial Science and Technology
1-1-1, Umezono, Tsukuba, Ibaraki 305-8568, Japan

[Received October 10, 2000, accepted November 7, 2000]

In this paper, Responsive Processor for parallel/distributed real-time processing, which can control various electronic control systems, is designed and implemented. Responsive Processor is integrates many functions in an ASIC chip, such as a RISC processing core (SPARC), Responsive Links that realize real-time communication, many peripheral functions including SDRAM/Fs, DMAC, PCI, USB, SIO, PIP, timers, pulse counters, PWM generators, A/D converters, D/A converters, etc. Its control boards and development environments are also designed and implemented.

Keywords: Responsive processor, Parallel/distributed control, Real-time, System-on-a-chip

1. Introduction

Recently, parallel/distributed processing is widely used in the field of information and communication. However, since control has been thought to be unsuitable for parallel/distributed processing, parallel/distributed processing for control(called distributed control) is not generally used.

In order to realize distributed control, real-time communication among processors and controllers is absolutely indispensable. However, communication specifications for such purposes are still not sufficient and not even standardized at the moment. The lack of common platforms, which can easily be used for designing a system using distributed control, is a factor that makes distributed control difficult. At the present, efforts to realize something close to distributed control by using multiple processors have been yielding unique systems with ad hoc designed controllers.

To realize distributed control easily and efficiently, we make common platforms with basic functions for distributed control to combine these, as in block building, to configure a system. Functions for such platforms include an MPU, real-time communication, control peripherals (A/D converters, DA converters, PWM generators, etc.), and computer peripherals (counters/timers, DMAC, SIO, etc.). To generalize distributed control, real-time communication is particularly important, and the standardization of real-time communication is desired. The platform is required to be a system-on-a-chip that integrates required functions into one chip, so that the chip can be embedded easily in various systems.

We discuss the design and implementation of a device called Responsive Processor, which realizes parallel/distributed processing, rather than the conventional centralized processing in the field of control. We also design and implement reference control boards using the Responsive Processor and software development environments. The Responsive Processor is designed so that it can be used as a common platform for electronic units of various systems.

We have been standardizing the interface and protocol of the Responsive Link that is a real-time communication specification for the Responsive Processor at ISO/IEC JTC1 SC25. When this international standardization is completed, real-time communication will be possible among different systems and an even larger distributed control system can be realized. The Responsive Processor is expected to bring a paradigm shift to the field of control.

2. Overview

First, we give an overview of the Responsive Processor. A Responsive Processor is a system-on-a-chip in which functions for parallel/distributed control are integrated on a single chip by using system LSI technology. Specifically, the ASIC integrates a RISC processing core (SPARC), four Responsive Links for realizing real-time communication, various computer peripherals (SDRAM/Fs, PCI, USB, DMA controller, timers, SIO, PIO, etc.), various control peripherals (A/D converters, D/A converters, pulse counters, PWM generators, etc.), etc into a single chip. (Fig.1). One significant difference between a Responsive Processor and a conventional one-chip microcomputer is that the Responsive Processor has four real-time communication links (called Responsive Links. Here, "responsive" means4.5) 1) real-time2) and reactive.) It is possible to configure a distributed control system easily by con-

3. Parallel and Distributed Control

Parallel processing and distributed processing are generally widely used in the field of computers that deals with scientific and technical computations requiring exceedingly high computing power and databases that must process huge numbers of transactions. Such techniques are not generally used in the field of control, because most control algorithms are sequential, so parallel control has been thought to be difficult.

For example, a sorting algorithm that rearranges numbers over the computer is essentially sequential. However, sorting is achieved at high speeds with the advent of parallel computers, since different algorithms for parallel sorting have been devised by many researchers. Similarly, a shift is considered to take place in the field from concentrated control, in which a central processing unit (CPU) controls all I/O devices using a sequential algorithm, to parallel/distributed control where a group of node controllers, each controlling I/O devices locally, controls the whole system in the near future.

Here, we assume that the most important factor for realizing parallel/distributed control is real-time communication among controllers (nodes), as discussed in the next section.

3.1. Parallel Control

Parallel control is realized by multiple processors by making a parallel arrangement of sequential control algorithms. It is necessary to devise a newly parallel control algorithm, as in parallel sorting as a previously given example. The idea behind making a parallel arrangement similar to parallel sorting, but the most significant difference between parallel sorting and parallel control is that because the concept of time is required for control, control process must be processed in real-time while a time limitation is fulfilled.

In other words, an operation can be executed stably as long as consistency is secured in parallel sorting, even when extra time is used by a different processor in referencing data. In contrast, stable control cannot be achieved in parallel control, if referencing cannot be completed within a certain time because control processes cannot be processed in real-time.

3.2. Distributed Control

Distributed control, unlike parallel control, does not mean that each control operation is divided. A single controller controls more than one I/O device including actuators, sensors, etc., independently. A system is composed of several controllers, and these controllers control the system as a whole while communicating with each other in real-time. A main controller may be present for controlling the whole system (control of each controller). Conversely, all controllers may be on equal terms, without the presence of a main controller. Fig. 2 gives an example of humanoids.

In regard to Responsive Processors and Responsive Links, the first research objective was the construction of an electronic control unit for humanoids to reduce internal LAN wiring and to simplify electronic units, etc. In Fig. 2, all parts embedded in joints are Responsive Processors, while the signal wires indicated by broken lines are Responsive Links that realize real-time communication.

In distributed control, functionally distributed modules are controlled by an independent controller, and the system as a whole is controlled via communication of all controllers involved. Controllers must communicate with

each other in real-time. For instance, if an actuator acquires sensor data from the sensor module and controls an actuator, stable control cannot be achieved unless the communication channel operates in real time.

3.3. Real-Time Communication in Parallel/Distributed Control

Although parallel control requires fine-grained real-time communication than distributed control, parallel/distributed control is a necessary condition for parallel/distributed control, but is not a sufficient condition. Sufficient is achieved only when a stably parallel/distributed control algorithm has been devised; however, it is not our objective to devise a parallel/distributed control algorithm for individual cases. Our objective is to make available tools for realizing parallel/distributed control, which were previously difficult to create. For example, when the proposed Responsive Processor is used, it becomes possible to design and implement a theoretical model of autonomous distributed control and real-time parallel computation model without any modification of these models.

Intercontroller communication for parallel/distributed control is considered to require different functions as follows:

- Hard real-time
- Error correction
- Automatic transmission by hardware

What is meant by hard real-time communication is real-time communication that guarantees time delays. Real-time communication may be classified roughly into 2 modes:

- Soft real-time communication
- Hard real-time communication

Almost all conventional real-time communication is soft real-time. Soft real-time communication allows for a certain number of missed deadlines, which means a bandwidth guarantee for communication in many cases. So, this mode of communication is intended to transmit multimedia data including images, sounds, etc. within the guaranteed bandwidth. In contrast, to realize parallel/distributed control, hard real-time communication is required which guarantees a latency period required for communication data to reach its intended receiver. Because the communication network guarantees real-time, the division of a parallel/distributed control algorithm and other operations are well supported.

In real-time transfer of multimedia data, hardly any effect is created even when data is somewhat corrupted, but in control, when communication data is corrupted, control may become impossible, or corrupted data may lead to an accident. In many current real-time communication systems, error correction is not supported, so it is difficult to realize a control system by using them.

Even when high-speed real-time communication could be achieved, significance is lost if almost all processor power for this realization were used without modification. If automatic real-time communication could be conducted by hardware without using most processor power for real-time communication, it would seem that controllers could use sufficient processor power in control operations.

4. Responsive Processor

Functions for the Responsive Processor are classified into four function and the design and implementation of each is discussed:

1. Real-time communication: Responsive Link
2. Computational operations
3. Peripherals for control
4. Peripherals for computers

4.1. Real-Time Communication: Responsive Link

4.1.1. Real-Time Communication Architecture

Up to now, specifications for high-speed interfaces, such as 10Bit Ethernet, ATM, Fibre Channel, IEEE-1394, and USB, have been proposed and designed. Some support real-time communication mode (isochronous transfer, etc.) for multimedia data, but they suffer from problems such as no error correction in real-time communication mode, the guarantee of only soft real-time, etc., when they are to be used for parallel/distributed control.

In contrast, real-time communication architecture for the Responsive Processor is designed to be used for embedded control. In general, communication data are divided into minimum communication units (called packets) for transmission. Because multitasking is required to communicate with other systems and those systems have to process in parallel by time sharing and transmit packets to other systems. Here, the size of ordinary data including image data, voice data, etc. is large; in contrast, the size of an event is exceedingly small. There are a large number of data packets that must be communicated and a very slight amount of event packets which are very important for real-time control on the
same link at the same time. In a real-time system aimed at control, it is most important to shorten and fix latency for event transmission and to strictly keep time limitations. Nevertheless, according to conventional methods in which communication of data and events is conducted by time sharing through a single shared communication line, it is not possible to bind the time of event transmission accurately; it is difficult to realize a hard real-time system.

In a serial bus in which a single communication channel is used by several modules, the bandwidth varies dynamically depending on how many modules are engaged in communication at the same time; so it is difficult to keep time limitations and effective speed is not easily attained.

Because, as a trade-off in real-time communication, soft real-time communication is used mainly for multimedia data communication, while hard real-time communication is mainly used for control, the following demands must be met:

- Soft real-time: guarantee of bandwidth requires to make throughput as high as possible
- Hard real-time: guarantees of latency requires to make latency as short as possible

As indicated in Table 1, when the packet size is made larger, throughput becomes higher, but the latency ends up becoming longer. Conversely, when the packet size is made smaller, the latency becomes shorter, but the overhead becomes larger, and the throughput ends up becoming lower.

In the Responsive Link, a line for data communication called a data link is separated from a line for event communication called an event link as shown in Fig. 3. A link is point-to-point and full-duplex. The data link realizes soft real-time communication. Its packet size is fixed length and large. It guarantees the bandwidth. The event link realizes hard real-time communication. Its packet size is fixed length and small. It guarantees the latency of events.

The data link is used for communicating ordinary data and data for soft real-time communication, while the event link is used for communicating events. The size and flow rate of events are exceedingly small, the data path is separated from the event path, and a link is point-to-point; event packets can be transmitted in hard real-time via the event link. (This has nothing to do with how congested the data links may be.)

The network switch section of the Responsive Link is furnished with the over-taking function. Packets are of fixed length (data packets: 64 bytes, event packets: 16 bytes), and the packets are each given a priority. A data packet is composed of a header section (network address with priority), a payload section, and a trailer section (control and status) as shown in Fig. 4. As long as a collision does not occur, data is transmitted to the next node by receiving overhead of the header section, but if a collision takes place at a certain node, a packet with a higher priority is able to overtake a packet with a lower priority.

The priority of a packet can be replaced at each node.

<table>
<thead>
<tr>
<th>Point Packet Size</th>
<th>Large</th>
<th>Smaller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>Higher</td>
<td>Lower</td>
</tr>
<tr>
<td>Latency</td>
<td>Larger</td>
<td>Shorter</td>
</tr>
</tbody>
</table>

Fig. 3. Responsive link

The overtaking of a packet is conducted based on the priority attached to that packet, and it is possible to add a new priority to a packet when it is transmitted from the current node to the next one. The new priority is used for overtaking at the next node.

The priority attachment and priority replacement on a packet are conducted by software (parallel/distributed real-time operating systems). Because of these functions, real-time communication is realized by distributed control rather than conventional centralized control.

The routing tables of the event link and the data link can be set up so that the event link and the data link can be routed independently.

The Responsive Link can correct communication errors by hardware at every hop. Each byte consists of 12-bit frame including, and when an error is 1 bit per 1 frame 8-bit data + 4-bit redundant code. One bit error can be corrected by hardware without retransmission as shown in Fig. 4.

The communication speed can be changed by software under different circumstances (configuration, application, etc.). Because the Responsive Processor is designed for embedded systems, its power consumption is a serious problem. In general, the faster the communication speed, the larger the power consumption, and the slower, the smaller.

If the Responsive Processor is used as it is embedded and battery-driven, its communication speed is not so important, but it is very important to save power consumption. In another application, high performance may be required as much as possible, but power consumption is not so important. If it is fixed near a very noisy environment (for example, near a large AC motor that is not shielded), a situation may occur where, when speed becomes faster, error correction cannot be done under the
Table 2. Speed vs. power on responsive link

<table>
<thead>
<tr>
<th>Speed of Modulation (MHz)</th>
<th>10</th>
<th>8</th>
<th>6</th>
<th>4</th>
<th>2.5</th>
<th>1.25</th>
<th>0.625</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed of Data (Mbaud)</td>
<td>67</td>
<td>33</td>
<td>17</td>
<td>8</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latency of Event (usec)</td>
<td>3.1</td>
<td>0.7</td>
<td>0.2</td>
<td>0.1</td>
<td>0.04</td>
<td>0.01</td>
<td>0.004</td>
</tr>
<tr>
<td>Power (W)</td>
<td>0.6</td>
<td>0.1</td>
<td>0.05</td>
<td>0.05</td>
<td>0.01</td>
<td>0.004</td>
<td></td>
</tr>
</tbody>
</table>

The inference of noise levels is that the maximum modulation speed of the Responsive Link is 100Mbaud. Table 2 shows the relationship between communication speed and power consumption. The communication speed can be dynamically changed during operations by software. When the modulation speed is 100Mbaud, the latency of an event packet with 8-byte payload is expressed as follows:

Latency of Event = 2usec + 1usec x n hops.

Transmitting and receiving overheads require 2usec, and 1usec is required per hop. Given that the 4-byte latency of Gbit Ethernet and Myrinet represent high-speed networks in use, this is nearly 10usec, which is at the low end of the event link realization, very short latency. For the Responsive Processor, five Responsive Links (5 x 5 network switches) were designed and implemented, with one connected to the processor and 4 extended outside the chip as shown in Fig. 1. Each link is full-duplex and 5 sets are available for each set of the data and event link, so the switching speed of the backbone is about 2Gbps.

The electric interface was P-CML (Pseudo Current Mode Logic), and input and output interfaces are basically designed differentially. The P-CML can interface LVTT and (Low Voltage TTL) and ECL (Emitter Coupled Logic) by simply external circuits. In connections to the short-distance connections, LVTT is used for direct connections. For long-distance connection, ECL or optical fibers are used.

4.1.2. Automatic Transmitting and Receiving by Hardware

The Responsive Processor would lose its significance if its processing power were wasted solely to realizing real-time communication. The Responsive Processor was designed combining it with a DMA controller, so that packets can be transmitted and received automatically.

We designed and implemented the automatically transmitting and receiving mechanism by hardware. In transmission, transmitting data are allocated in the main memory (SDRAM), and its address and data length are set up in control registers of the 3MAC and the Responsive Link. Then hardware can transmit data without using its processing power and with the bus utilization as low as possible. Specifically, procedures are as follows:

1. Transmitting data made available in main memory
2. Setting up in the Responsive Link control registers
3. Setting up in the DMA controller and triggering start-up

4. The DMA controller acquires processor bus, and transmitting data is transferred by the length of its transmitting buffer from main memory to the Responsive Link
5. After completion of the burst transfer of the length of its transmitting buffer, the processor bus is released (during this time, processor and other bus master devices can process in parallel)
6. After transmission of the length of the transmitting buffer is over, the Responsive Link makes a DMA request to the 3MAC, and then the transmitting data is transferred (above steps 4 and 5 are repeated)
7. After all transmitting data is transmitted, an interrupt is generated into the processor, and transmitting completion is transferred (the interrupt may or may not be generated depending on setup)

When transmitting data is made available in the main memory as stated above, it is then possible to transfer data automatically and to lower the utilization of the internal bus. Since it is necessary to make latency as short as possible in the event link, so a channel other than the above processor bus channel is designed. Specifically, the hard-bus, which is the internal bus of the SPARC, is connected directly to the inside of the event link. In other words, the event link can be seen as if it were an internal register from the SPARC. In connection: via the above processor bus, an external bus master device may, in some cases, get an access right to the bus. In such a case, no access to the Responsive Link is allowed until the external bus master device releases the bus and the processor acquires access right to the bus, with the result that the time can no longer be bound. In contrast, when the hard-bus is connected directly to the ResponsiveLink, transmitting and receiving are possible, as long as the cache is being hit, even when the external master bus is
holding the processor bus. In addition, no overhead exists in the exclusive line; data transfer is very fast.

We also designed and implemented the automatically receiving mechanism by hardware, without using the processing power and, with the bus utilization rate kept as low as possible.

4.2. Computational Functions

In order to realize heavy on-line processing, such as following, image processing, etc., it is necessary that the core processor of the Responsive Processor should have a sufficiently high processing speed. At present, the majority of high-speed processors are RISC, but if one is used embedded, problems are encountered as described below.

* Pipeline stalls at interrupt
* Saving registers at interrupt

To ease these problems, we have been using the regis-
ters windows of the SPARC architecture. (11) For the Responsive Processor, we shall similarly use Fujitsu’s SPARClite (12,13) (clock speed: 190MHz; data cache: 8KB; and instruction cache: 8KB) and realize both the high-speed capability and responsiveness. For embedded use, power consumption becomes an important issue. A power management unit, capable of changing the clock speed dynamically by software as indicated in Table 3, is designed and implemented in the chip, with the result that speed and power consumption are controlled successfully.

4.3. Control Peripherals

In general, the functions and specifications for controlling motors, sensors, etc. are different for each researcher and developer. In designing the chip, consideration was given to the desirability of using the chip by as many researchers and developers as possible, so first, the following questionnaire was sent via e-mail to those studying control and robotics:

1. Developing environment for real-time control
   (a) Hardware
   (b) Operating system
   (c) Developing environment
   (d) Developing language
   2. Real-time capability
   3. I/O devices

Based on this survey, peripherals for control as indicated in Table 4 were designed and implemented as I/O control devices.

4.4. Computer Peripherals

As in control peripherals, the survey was used to determine computer peripherals in Table 5.

As a peripheral function for computers, interfacing with personal computers was regarded as important, and PCI and USB, which were complex in circuitry, were integrated into the chip.

4.5. Physical Features of the Responsive Processor

The layout of the Responsive Processor is given in Fig.5. The large square area at left bottom indicates the SPARC processor (about 300kigate), while the small rectangular area at right bottom is the analog section includ-

<table>
<thead>
<tr>
<th>Table 3. Clock vs. power on processing core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock(MHz)</td>
</tr>
<tr>
<td>Speed(MIPS)</td>
</tr>
<tr>
<td>Power(W)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4. Peripherals for control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripherals</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>10Mb A/D Converter</td>
</tr>
<tr>
<td>8bit D/A Converter</td>
</tr>
<tr>
<td>50MHz PWM Generator</td>
</tr>
<tr>
<td>34bit Pulse Counter</td>
</tr>
<tr>
<td>Digital I/O Port</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5. Peripherals for computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripherals</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>SDRAM 512MB</td>
</tr>
<tr>
<td>Interrupt Controller</td>
</tr>
<tr>
<td>DMA Controller</td>
</tr>
<tr>
<td>PCI 64bit</td>
</tr>
<tr>
<td>USB 64bit</td>
</tr>
<tr>
<td>Asynchronous SID</td>
</tr>
<tr>
<td>Synchronous SID</td>
</tr>
<tr>
<td>10bit Timer/Counter</td>
</tr>
</tbody>
</table>

ing AO converters, DA converters, etc. The group of rectangles at top right is the routing table for the Responsive Link, and the group of rectangles representing 2 blocks atop left designates buffers for packet overtaking by the Responsive Link. The bigger rectangle is for the data link, while the smaller one is for the event link. In addition, the large region enclosing the routing table and the overtaking buffers is the Responsive Link unit having about 400kigate.

Design rules for the Responsive Processor are as follows:

* Process rule: 0.35um CMOS 4-layered metal
* No. of total gates: 2,378k gate
* Die size: 14.5mm x 14.0mm = 210mm²
* Package: 416-pins IGQA (40mm x 40mm)
* Voltage: 3.3V
* Maximum power consumption: 2W

Figure 6 shows the Responsive Processor.

5. Control Boards

To evaluate the Responsive Processor for many users, different Responsive Processor control boards were designed and implemented as described below.

* PCI Card
* PC Card Bus Card (PCMCIA type)
* Embedded Card (Credit card size)

The PCI control board (Fig.7) outputs all pins of the

Responsive Processor: Use for checking and evaluating purposes, AD converters, DA converters, PWM generators, pulse counters, etc., are exposed as header pins on the board; this design makes it possible to evaluate a desired function easily. The USB, 232C, etc., are designed to be compatible so that cables for personal computers would be usable without modification.

Because researchers in control found that the number of channels and accuracy were not satisfactory in on-chip AD converters and DA converters, we developed a new PCI board for analog control with high-performance ADC chips and DAC chips as shown in Fig. 8. Most commercially available AD/DA boards use multiplexers for realizing many channels because of cutting cost. These boards control AD/DA by time-sharing and multiplexing; they are not durable enough for high-end uses requiring high performance for research.

A PCI board for analog control with multiple high-performance channels, durable enough for such high-end uses, was designed and implemented. This board is equipped with high-performance AD converters and DA converters in addition to the on-chip AD converters and DA converters as shown in Table 6. This board does not use multiplexers, and operates AD converters and DA converters with 6 channels each in parallel by using FPGA. A variety of control modes was designed and implemented to respond to different demands. In AD converter control, for example, the following modes were designed and implemented:

- Mode for data read and write operations from the processor.
- Mode in which a command is issued to read the current value from the processor and immediately after that the latest data is read.
- Mode in which every time the latest data is read by AD converters, that latest data is latched and when the data of the AD converters is read from the processor, the latest data can always be read.
- Mode for generating an interrupt to the processor so as to read data from the AD converters.

Since we expect that the Responsive Processor will be used by many researchers and developers, user-friendly development environment is necessary. The results of the previous surveys made it clear that development environments with robustness and control were favored by more Windows than UNIX users. To have the Responsive Processor used by a large number of users, a cross-development environment was constructed over Windows and UNIX systems (Linux, FreeBSD, Solaris), focusing on Windows below.

6.1. Interface for Computers
To cross-develop the board by using an external computer (PC), it is necessary that the Responsive Processor communicates with the external computer. The Responsive Processor has the following functions as interfaces of PC systems:

1. PCI
2. USB
3. Serial (RS232C)

We designed the initial cross-development environment through RS-232C that can be used by all control...
boards\(^{(2,9)}\). The cross-development via RS-232C has generality, but is slow. Because many users wanted to develop software by inserting a control card into their PCs, a cross-development environment via PCI was constructed. We are designing and implementing a cross-development environment via USB.

6.2. Design and Implementation of Cross-development Environment

The processing core of the Responsive Processor is SPARC and GCC (GNU C compiler) already supports SPARC, so cross-development environments were based on GNU tools.

To construct a cross-development environment storable in ROM over a Windows system, we transplanted a cross compiler, a cross assembler, etc., of GNU onto a Windows system.

Since a simple cross-development environment cannot be debugged easily, an environment integrating a source-level debugger was constructed.

WinGDB obtained by transplanting GDB as the GNU debugger over Windows is used.

The host PC executes the main routine of GDB, while the target board executes a program under the control of the host PC. The target board has a boot ROM chip that works with WinGDB and has a function to execute programs while communicating with the host PC. The flow of actual program development (debugging) is as follows:

1. In the cross-environment over the host PC, the source files in C language are compiled to the execution format of SPARC with g-option.
2. The target board is reset, and WinGDB is launched on the host.
3. The executable files with debugging information are loaded onto the target.
4. Symbol files produced by GCC are loaded onto WinGDB.
5. The files on the target board are executed remotely by using WinGDB of the host PC and debugging at the source level.

WinGDB has all functions of an ordinary GDB. When you debug C over UNIX using GDB, you can debug the program by setting up break points and referencing variables while executing. WinGDB has the GDB expanded and a variety of debugging functions, such as, for example, a window showing all registers, a window displaying variables. (Fig.9)

After debugging is completed by WinGDB, it is possible to develop an embedded system easily by storing data on a ROM chip.

Using WinGDB makes it possible to design and implement operating systems on the Responsive Processor using source-level debugging operating systems. In many cases, debugging can be conducted only using low-level debugging methods. When WinGDB is used, an operating system can even be designed and implemented while source-level debugging is being conducted. In this cross-development environment, a real-time cooperating system \(\mu\)-PULSER\(^{(2,10)}\) is transplanted over the Responsive Processor.\(^{(9)}\)

At present, the following operating systems are supported as host operating systems:

- Windows (95, 98, NT, 2000)
- Linux
- FreeBSD
- Solaris
- IRIX

7. Conclusions

The design and implementation of the Responsive Processor as a platform for parallel/distributed control was discussed. This Responsive Processor is a system-on-a-chip integrating on parallel/distributed control functionalities.

A system using the Responsive Processor allows for flexible configurations by simply connecting controllers with communication links and can conceivably be applied to different forms of robots, including humanoid type ro-

<table>
<thead>
<tr>
<th>Peripherals</th>
<th>Num. of Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>12Bit A/D Converter</td>
<td>6</td>
</tr>
<tr>
<td>16Bit D/A Converters</td>
<td>6</td>
</tr>
</tbody>
</table>

![Fig. 8. PCI control board with ADCs and DACs](image)

![Fig. 9. Screen of WinGDB](image)
bots, cart type robots, etc. Such a system can easily be embedded because it is small and connection is simple, and is therefore used as a controller for office or home automation. It can conceivably be applied as a control system inside cars, which are becoming increasingly complex, or for amusement.

The Responsive Link that is a real-time communication standard for Responsive Processors has been standardized internationally in interfacing and protocol by ISO/IEC JTC1 SC25. By this standardization, the realization of a common platform for the construction of mechatronics systems and embedded systems is intended, and connections among different kinds of systems are latently possible.

Since it may be difficult for general developers and users to use a simple chip, various control boards were designed and implemented. Moreover, cross-development environments for the Responsive Processor were designed and implemented. This cross-development environment is not simply a cross compiler but rather a GDB environment (WinGDB) over Windows that can realize source-code level debugging.

The use of WinGDB has made it possible to develop a wide range of tasks from embedding applications in the ROM on the Responsive Processor to the development of operating systems. By using this cross-development environment, µ-PULSER as an real-time OS was designed and implemented on the Responsive Processor.

An integrated environment that can easily realize parallel/distributed control using these hardware and software was configured. In the future, many researchers and users will be asked to make evaluations. It is our sincere wish that the Responsive Processor be used for various parallel/distributed control such as robot control, etc.

Acknowledgments

Responsive Processor R&D was supported by funds from the MITI’s International Standard Creation Research & Development System and Electrochemical Laboratory COE. The Responsive Link is the result of a study at PRESTO of Japan Science and Technology Corporation. In the Humanoid Platform and Task Force of the International Standard Creation Research & Development System, valuable advice was given by Prof. H. Isobe and Prof. T. Inaba, both of the university of Tokyo. Dr. K. Tanie, Department Head of the Mechanical Engineering Laboratory, and Dr. H. Hirakawa, chief researcher of the Electrochemical Laboratory COE. In addition, support was provided by Dr. K. Okmai, Department Head at Electrochemical Laboratory COE. We thank them all for helping make this study possible.

References:


Name: Nobuyuki Yamamuro
Affiliation: Assistant Professor, Keio University
Address: 3-14-1, Hiyoshi, Kawasaki-ku, Yokohama, Kanagawa, 223-8522, Japan
Brief Biographical History:
1980 - 1985 Researcher, Electrochemical Laboratory
1997 - 1999 Researcher, PRESTO, Japan Science and Technology Corporation (JST)
1998 - 2000 3 Researcher, Keio University
1998 - 2001 Researcher, Electrochemical Laboratory
2001-5 - Assistant Professor, Keio University

Main Works:
- Membership in Learned Societies:
  - IEEE
  - The Robotics Society of Japan
  - Information Processing Society of Japan

137