Centralized Processing and Distributed I/O for Robot Control

Peter Kazanzides, Paul Thienphrapa
Department of Computer Science, Johns Hopkins University, Baltimore, MD

Abstract—*Historically, the architecture of robot controllers has been dictated by technology constraints. When computers and networks were slow, it was necessary to logically distribute the computation but to physically centralize both the computation and I/O. This is represented by a controller comprised of large rack of embedded processors, with cables to all robot sensors and actuators. As technology improved, it became feasible to distribute both computation and I/O, as illustrated by systems that use a field bus to connect a central controller to low-level processors embedded near or within the robot. This paper advocates a new architecture, centralized processing and distributed I/O, that is enabled by current computer and network technology. This architecture provides benefits in academic environments because it simplifies development of robot control software. The feasibility of this approach is demonstrated by a custom robot controller that uses IEEE 1394 (FireWire) to provide direct communication between a central computer and non-intelligent peripheral hardware devices.*

1. INTRODUCTION

Robot systems are concurrent by nature: multiple joints must be controlled simultaneously, and there is often a hierarchy of control strategies. A typical robot controller (Figure 1) contains "loops" for servo control, supervisory (e.g., trajectory) control, and the application. In many cases, the servo loop is distributed among multiple joint-level microprocessors.

In the early days of robotics, controllers consisted of a central computer with multiple joint-level control boards on a parallel bus, such as ISA, Q-Bus, Multibus, or VME; this was necessary for performance reasons alone. Although the central computer usually contained multiple processors (e.g., joint-level control boards), this architecture can be characterized as *centralized processing and I/O*.

With the emergence of high-speed serial networks, such as CAN, Ethernet, USB, and IEEE 1394, it became possible to physically distribute the joint controller boards and associated power amplifiers. By placing these components inside the robot arm, or at its base, significant reductions in cabling could be achieved. In particular, the thick cables containing multiple wires for motor power and sensor feedback could be replaced by thin network and power cables. These types of systems can be characterized as *distributed processing and I/O.*

We advocate a new approach for robot control: *centralized processing and distributed I/O*. This can be achieved by replacing the microprocessors with programmable logic, such as field-programmable gate arrays (FPGAs), that provide direct, low-latency, interfaces between the high-speed serial network and the I/O hardware. This preserves the advantages of reduced cabling, while allowing all software to be implemented on a high-performance computer that contains a familiar software development environment. We believe that this is especially important for education and research environments because it frees developers from having to learn the idiosyncrasies of the various embedded microprocessors. We contend that this approach is feasible due to recent advances in processor performance, especially the move to multi-core architectures, coupled with the extraordinarily high data rates of modern serial networks. Another key factor is the availability of low-cost real-time operating systems, such as those based on Linux.

This paper will describe a robot controller that is based on the principle of centralized processing and distributed I/O. The next section reviews the design choices that led to the selection of IEEE 1394 as the high-speed network, and discusses other alternatives. This is followed by a summary of the hardware prototype, which is a custom controller for small DC motors used in a microsurgical snake robot. Finally, we present some preliminary results obtained with the hardware.

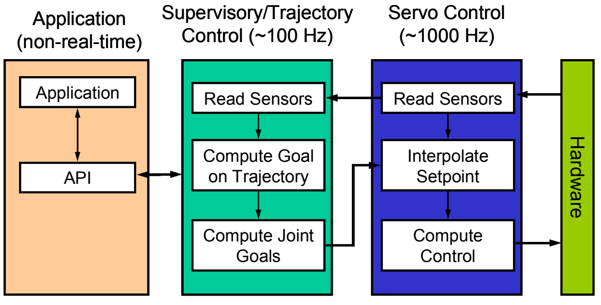


Figure 1 - Typical robot control architecture

2. CHOICE OF A HIGH SPEED NETWORK

The desire to perform real-time robot control, at frequencies of 1-10 kHz, over a serial network leads to several key requirements. First, we note that data packets are relatively small. For example, closed-loop control of a robot joint can be accomplished with as little as one feedback position (e.g., from a pot or encoder) and one control signal (e.g., voltage or current to apply to the motor). A more generous setup may contain a few feedback signals (e.g., position, velocity, motor current, amplifier status) and a few control signals (e.g., voltage and current limit). Even if 32-bit values are used for many of these, a typical data packet (read or write) would be on the order of 100 bytes. Thus, a six-joint robot would require about 1200 bytes (6*200); at a control frequency of 10 kHz, this would require a bus bandwidth of 12 Mbytes/sec, or approximately 100 Mbits/sec. This is not difficult to achieve with modern high-speed serial networks, such as IEEE 1394 (up to 400 or 800 Mbits/sec for 1394a or 1394b, respectively), USB 2.0 (up to 480 Mbits/sec) or Ethernet (10, 100, or 1000 Mbits/sec).

A more critical performance metric is the latency of the data transfers because it introduces a time delay in the control computations, which compromises performance and can lead to instability. Latency is primarily determined by overhead in the protocol and the software drivers. Based on our review of specifications and published reports, we concluded that IEEE 1394 should provide the lowest latency, especially when used with a real-time operating system. The protocol supports real-time communication with guaranteed 8 kHz (125 μs) bus cycles in isochronous mode, with faster access rates possible in asynchronous mode. It is an effective solution for real-time control, as shown in [1, 2], and by its use in fly-by-wire systems [3].

Another important requirement is the ability to daisy-chain nodes. For example, if a multi-axis robot contains embedded I/O boards, daisy-chaining allows just a single network cable to be connected to the robot – this cable connects to the first I/O board, which then connects (daisy-chains) to the second board, and so on. This allows a significant cabling reduction compared to connecting a separate network cable to each board (i.e., a star topology). Physically, all of the considered serial networks (except the outdated 10 Mbit Ethernet with coaxial cable) utilize point-to-point links but provide daisy-chaining solutions. IEEE 1394 provides an especially attractive and inexpensive solution by providing repeaters at the physical layer. In contrast, USB requires a hub (with associated cost and complexity) and Ethernet typically uses high-speed switches.

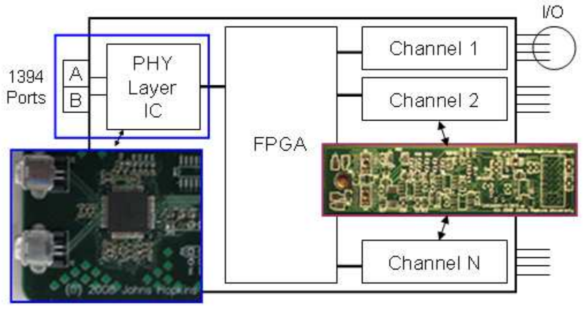


Figure 2 - Block diagram of a node

Although we selected IEEE 1394, we note that USB and Ethernet have the advantage of market dominance. Also, although we concluded that standard Ethernet was not ideal for real-time control, we did not consider the many "real-time" Ethernet variations that have been created. For example, Powerlink employs a bus manager that schedules 200-μs cycles of isochronous and asynchronous phases [4], a close resemblance to 1394. In EtherCAT, nodes forward and append data packets as they are received with the aid of dedicated hardware and software, resulting in the ability to communicate with 100 axes in 100 μs [5]. EtherCAT is a relative newcomer; [6] is an example showing its potential. SERCOS approached a communication bottleneck in [7] with increasing axes and cycle rates, but its recent combination with Ethernet (SERCOS-III) has endowed it with the ability to update 70 axes every 250 μs. Many of these Ethernet variations also support daisy-chaining without the use of switches.

IEEE 1394 has a potential drawback in the lack of high-flexibility cables for installation within the moving structure of a robot arm. In our application, this is not a serious limitation because medical robots, compared to industrial robots, move slowly and infrequently. Currently, the Ethernet variations described above provide better cabling options.

At the time of our initial evaluation (two years ago), we did not consider PCI Express because it was limited to backplanes and circuit boards. With the recent introduction of a cable-based standard, PCI Express appears to be an attractive alternative because it would not require protocol conversion between the motherboard and peripheral devices.

3. HARDWARE PROTOTYPE

We constructed a hardware prototype to test the feasibility of IEEE 1394a for centralized processing and distributed I/O. We chose 1394a, rather than the faster 1394b, because it provided ample bandwidth and the lower signal frequencies simplified the hardware design. The basic design of a node consists of an IEEE 1394a physical layer (PHY) chip, an FPGA, and multiple power amplifier boards (channels), as shown in Figure 2. Each power amplifier board provides the hardware interface to a small DC motor. It is a custom design that provides speed or torque control, with precise measurement of the motor current. An earlier version of the design is presented in [8], which described an ISA bus board that contained four power amplifier channels. The new amplifier board contains several improvements,

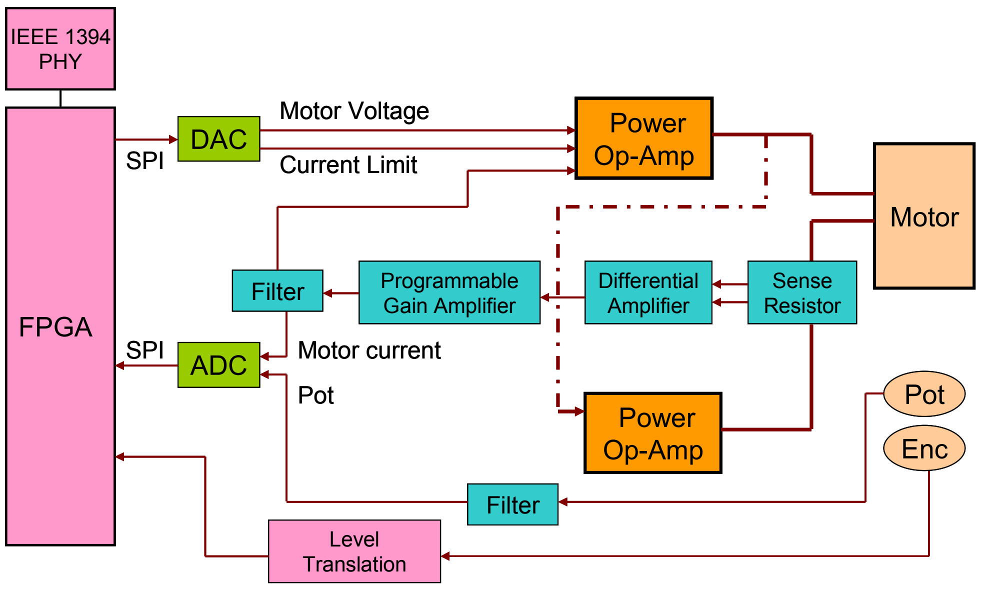


Figure 3 - Block diagram of I/O and power amplifier board with IEEE-1394 interface

such as a bridge amplifier design (two power op amps instead of one) and a programmable gain amplifier for the motor current feedback, which is also used for hardware speed control [8]. As depicted in Figure 3, the interface between the FPGA and the power amplifier board includes a Serial Peripheral Interface (SPI) to the digital-to-analog converter (DAC), analog-to-digital converter (ADC), and digital pot that implements the programmable gain amplifier. The power amplifier also provides incremental encoder feedback to the FPGA, with level translation to ensure that the FPGA input voltage limit (3.3V) is satisfied. The FPGA is responsible for the quadrature decode logic that converts

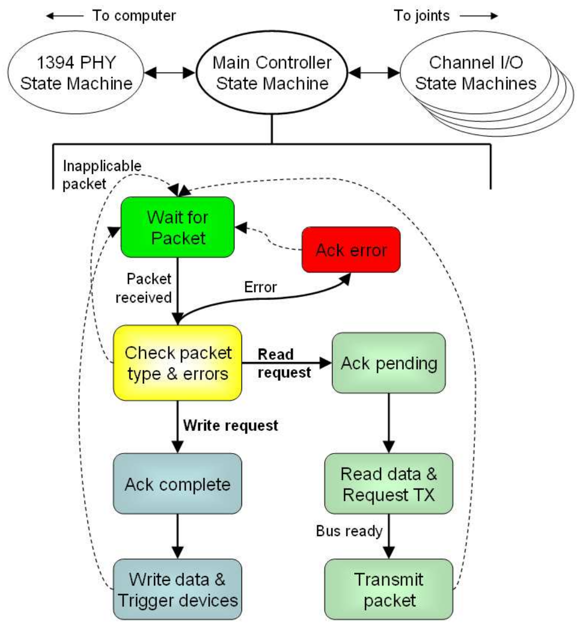


Figure 4 - FPGA structure and operation

the incremental encoder feedback into position and velocity measurements. Digital I/O lines (not shown) are used to enable/disable the power amplifiers and to provide status/fault feedback.

We used an FPGA development board (Altera UP3) for the first prototype to reduce our development risk. Realistically, we expected that our system would not work the first time we connected a PC to the prototype node. The development board eliminated the possibility that we did not properly design or fabricate the FPGA portion of the prototype. Thus, we could focus our debugging effort on the FPGA code and on the custom daughterboard that contained the IEEE 1394a PHY chip (Texas Instruments TSB41AB2).

Most of the functionality of each node is implemented as firmware on the FPGA, which serves as a low-latency interface between the power amplifier channels and the bus. The FPGA receives packets from the 1394 bus, responds to them, and communicates with the I/O devices. The computer can access the channels through control and data registers. The high-level FPGA operation is depicted in Figure 4 and the following sections briefly describe the methods for writing and reading node data. These sections focus on the peripheral devices that contain SPI interfaces because those are the most complex. The last section describes the PC software.

Writing data to a node

The PC performs a write transaction over the IEEE 1394 bus to send the motor voltage or current limit to the DAC, or to send the wiper setting to the digital pot. The FPGA receives the write request, checks the CRC, generates an acknowledgment packet (required by the IEEE 1394 protocol), and then decodes the packet to obtain the destination address and data. The data is written to a register which is then shifted out bit-by-bit to the appropriate device (DAC or digital pot) via the SPI interface.

Reading data from a node

The PC performs a read transaction to obtain the motor current or potentiometer feedback from the ADC, or to obtain the digital pot wiper setting. Because the ADC has a relatively long conversion time (about 0.7 µs per channel), it is not advisable to start a new conversion and wait for the result whenever a read request is received. Instead, the FPGA continuously requests conversions and stores the most recent results in registers. When the FPGA receives a read request, it checks the CRC, generates an acknowledgment, and then creates a reply packet from the most recent conversion result. In general, it can be challenging to properly synchronize access to the register (i.e., so the FPGA does not attempt to retrieve the latest conversion result while a new value is being stored). In our implementation this is simplified because the FPGA clock is obtained from the 49.152 MHz clock generated by the 1394a PHY chip. Thus, read requests are synchronous with respect to FPGA processing and SPI transfers.

Because the SPI signals are shared between all devices (DAC, ADC and digital pot), it is necessary to handle reading of the digital pot a little differently. Specifically, the PC must first send a write request to instruct the FPGA to pause ADC conversions, read the digital pot into an FPGA register, and then resume conversions. Next, the PC sends a read request to obtain the stored value. This extra complexity is not significant because reading of the digital pot should be a relatively rare occurrence.

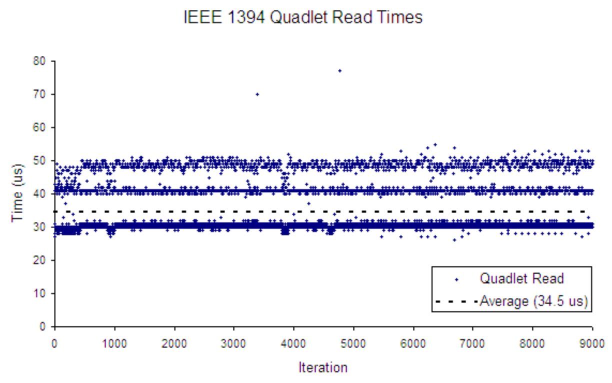
PC Software

The PC provides the environment for centralized computation. For real-time systems, we typically use the Real Time Application Interface (RTAI) for Linux. For the initial experiments reported below, we used a conventional Linux operating system. Our test software used the libraw1394 library to communicate with nodes. All read and write transactions were performed as quadlets (32 bits) because the block transfers had not yet been implemented in the FPGA firmware. In the future, when using a real-time operating system it may be beneficial to consider alternative software libraries and drivers, such as RT-FireWire [9], that are designed for real-time performance.

4. Performance Measurements

We measured the performance of quadlet (32-bit) read and write requests over the IEEE 1394a bus to our controller node. As described above, the reads are implemented on the FPGA such that there is no protocol delay (i.e. no busy wait) on the FPGA between receiving a read request and generating a response. Similarly, there is no delay between the receipt of a write request and the start of the write.

Figure 5 shows the timing results of 9,000 iterations of quadlet reads and writes. The tests were run at 400 Mbps with one node connected to a 2 GHz Pentium 4 PC by a 6-foot cable. The read times appear in three bands (30, 42, 49 µs) and the write times in two bands (27, 42 µs), probably due to variability in the discrete transaction sequences (e.g. request-ack-response for reads and request-ack for writes). The average read and write times are 34.5 and 30.2 µs respectively—each respective low band is most common. We noted a few large outliers, perhaps due to our use of a conventional operating system (Linux), rather than a real-time operating system, for these tests. We will investigate the regular distribution patterns as well as the outliers should they arise under a real-time operating system.



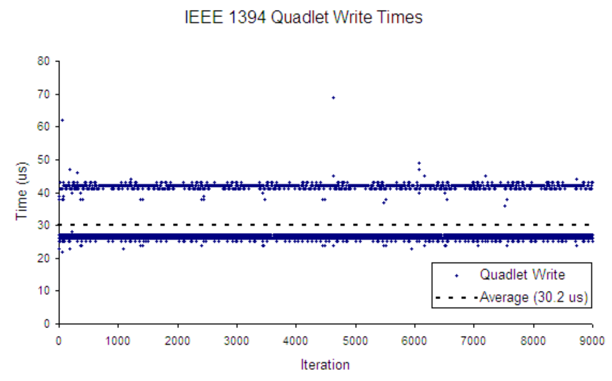


Figure 5 - IEEE-1394a quadlet read and write times at 400 Mbits/sec

Overall the times are well above theoretical maxima—e.g. a quadlet read, 296 total bits, should take less than a microsecond to complete at 400 Mbps. We believe that software overhead is the primary cause (as in [10]), and hope to reduce this in the future. Nevertheless, because software overhead should be relatively independent of data size, these results indicate that the most efficient approach is to use one block read to obtain all feedback data (from all joints serviced by the node) and one block write to send all control signals. In a straightforward implementation (read-control-write), a combined read/write time of about 65 µs leaves a generous 935 µs for control computations at 1 kHz, but only 60 µs at 8 kHz and 35 µs at 10 kHz. IEEE 1394 also supports isochronous transfers, which occur at a frequency of 8 kHz and may therefore be more efficient for control at this frequency.

5. Conclusions and Future Work

We presented the motivation for a new approach to robot control, where the computation is centralized on a PC but the I/O is distributed via a high-speed serial network. A key element is the use of programmable logic, such as an FPGA, to provide link layer services for the network by routing read and write requests to the appropriate hardware device.

The concept was demonstrated by a custom robot controller that uses IEEE 1394a (FireWire) for communication between the control PC and the distributed I/O devices. This controller is being developed to control a small snake robot for laryngeal surgery.

Preliminary performance data, with a conventional operating system (Linux), indicates that this approach is feasible for control rates up to several kHz; higher rates, such as 10 kHz, appear to be challenging with the current setup based on the measured times for quadlet read and write transactions. We are currently investigating whether significantly better performance can be obtained with a real-time operating system or by changes to the software drivers.

We evaluated the first prototype, which consisted of three types of boards: a commercial FPGA development board, a custom daughterboard with an IEEE 1394a PHY chip, and custom motor power amplifiers. We are now constructing a second prototype that combines the first two boards (FPGA and PHY) into a single custom board.

We refrain from calling centralized processing and distributed I/O a novel concept because it is likely that others have adopted this approach, especially those using one of the many enhanced “real time” Ethernet protocols. We contend that this approach is especially advantageous in areas such as research and education, where typical users are not proficient with software development using embedded microprocessors and their associated tools.

Acknowledgment

We thank Hamid Wasti of Regan Designs, Inc. (Coeur d’Alene, Idaho) and Fran Wu for their extensive help with the design and layout of the controller boards, and Mitch Williams for his help with the software setup. This project was supported by NSF EEC 9731748, NSF MRI 0722943, and Johns Hopkins University internal funds.

References

[1] Sarker, M., C. Kim, S. Baek, and B. You, “An IEEE-1394 based real-time robot control system for efficient controlling of humanoids,” *IEEE Intelligent Robots and Systems*, Beijing, China, Oct 2006.

[2] Zhang, Y., B. Orlic, P. Visser, and J. Broenink, “Hard real-time networking on FireWure,” *Real-Time Linux Workshop*, Lille, France, Nov 2005.

[3] Baltazar, G. and G. Chapelle, “Firewire in modern integrated military avionics,” *IEEE Aerospace and Electronic Systems Magazine*, vol.16, no.11, pp.12-16, Nov 2001.

[4] Ethernet Powerlink: http://ethernet-powerlink.org.

[5] EtherCAT Technology Group: http://www.ethercat.org.

[6] Robertz, S., K. Nilsson, R. Henriksson, and A. Blomdell, “Industrial robot motion control with real-time Java and EtherCAT," *IEEE Emerging Technologies & Factory Automation*, pp. 1453-1456, 2007.

[7] Lin, S., C. Ho, and Y. Tzou, “Distributed motion control using real-time network communication techniques,” *International Power Electronics and Motion Control*, vol. 2, pp. 843-847, Aug 2000.

[8] Kapoor, A., N. Simaan, and P. Kazanzides, “A system for speed and torque control of DC motors with application to small snake robots,” *IEEE Mechatronics and Robotics*, Aachen, Germany, Sep 2004.

[9] Zhang, Y., B. Orlic, P. Visser, and J. Broenink, “Hard real-time networking on FireWure,” *Real-Time Linux Workshop*, Lille, France, Nov 2005.

[10] Sarker, M., C. Kim, J. Cho, B. You, "Development of a network-based real-time robot control system over IEEE 1394: using open source software platform," *IEEE Mechatronics*, pp. 563-568, July 2006.