A Centralized Processing, Distributed I/O Motor Controller Based on IEEE 1394 for the Snake Robot

Paul Thienphrapa

Department of Computer Science
Johns Hopkins University
3400 North Charles Street
Baltimore, MD 21218

May 14, 2010

This project fulfills a requirement for the Ph.D. degree.

Advised By: Peter Kazanzides

Advisor Signature and Date: _____

Abstract

Research in surgical robots often calls for multi-axis motor controllers and other I/O hardware for interfacing various devices with computers. To facilitate convenient prototyping of robots with large numbers of axes and I/O lines, it would be beneficial to have controllers that scale well in this regard. We would like to incorporate additional components to a system without necessitating an unwieldy increase in I/O devices, nor introduce too many disjoint software interfaces and environments. High speed serial buses such as IEEE 1394 (FireWire) make it possible to consolidate multiple data streams into a single cable, and contemporary computers have the computational resources to process such dense data streams. These factors motivate a centralized processing, distributed I/O control architecture, which is particularly advantageous for education and research. This paper documents the design, implementation, and testing of a motor controller and its associated API using this design approach. A real-time controller for the Johns Hopkins University Snake Robot, a novel, miniature, and dexterous surgical manipulator with many degrees of freedom, is developed that combines previous experience and the aforementioned motivations with readily available but powerful new technologies.

Contents

List of Figures

List of Tables

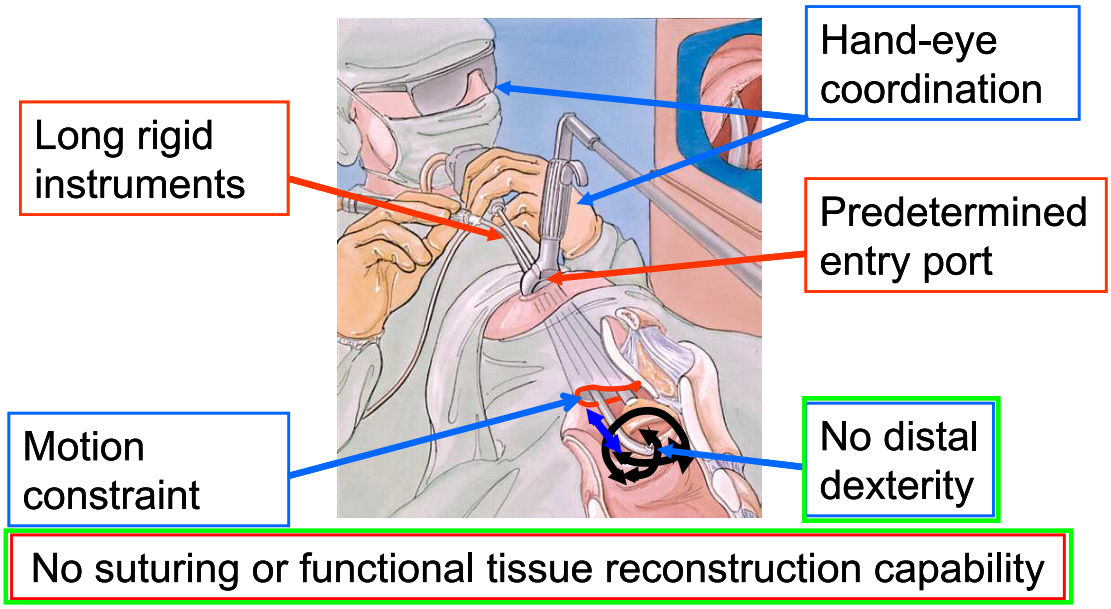

Figure 1: A minimally invasive surgery setup

1 Introduction

1.1 Background

Minimally invasive surgery (MIS) is often beneficial for patients due to reduction of trauma, leading to fewer complications and shorter hospital stays. However, MIS poses a number of challenges for surgeons, including constrained workspaces, limited field of view, and lack of dexterity at the distal end. These challenges remain despite the availability of manual MIS-specific instruments, which are rigid, difficult to manipulate through narrow insertion tubes, and they lack adequate suturing and tissue reconstruction capability. The situation is illustrated in Figure 1. In such situations, the efficacy of a surgical robot is strongly tied to its dexterity.

Research on the Snake Robot [1] seeks to improve MIS of the throat and upper airways by providing surgeons with highly dexterous robotically-controlled tools. This dexterity is achieved by incorporating more degrees of freedom (dof). More sophisticated surgical tasks can be accomplished by increasing dof, but the corresponding hardware increase imposes a practical limit on the exploration of these ideas. Similarly, research on different types of multi-axis surgical robots is often mired in the hardware construction effort. In response to these difficulties, this paper presents the development of a system that is well suited for real-time control of robots with many axes of control.

Figure 2: Snake Robot prototype

1.2 Snake Robot

A unique design targeted for MIS of the upper airways, the Snake Robot addresses these issues by introducing a small, dexterous end effector that can be teleoperated. To avoid obscuring the work area, the end effector is attached to its actuators via a hollow, narrow, meter-long shaft containing its wires, and appears at the distal end of a laryngoscope; this shaft is also teleoperable, as described below. Figure 2 depicts the configuration.

The Snake Robot is an eight-degree of freedom (dof) (11-actuator) manipulator. Multiple Snake Robots may be used for surgical tasks including bimanual suturing, suction, and camera placement. The dexterous end effector consists of two snake-like units (SLUs) connected in series; each SLU is constructed using four superelastic NiTi tubes. The anatomy of an SLU is shown in Figure 3. The center tube, i.e. the primary backbone, is surrounded by the three other tubes, the secondary backbones, at equally-spaced radial and angular distances. These four backbones are fixed to the end disc, but only the primary (central) backbone is fixed to the base disc; all backbones are free to glide through holes in the equally-spaced intermediate spacer discs.

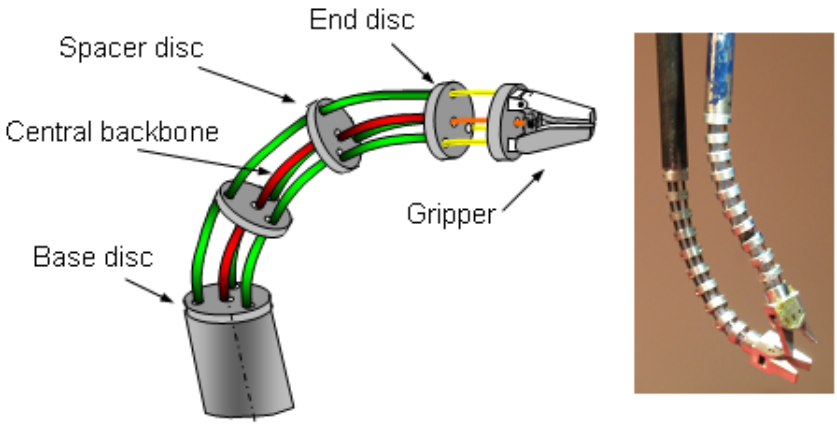

Figure 3: Anatomy of a snake-like unit (left) and photo (right)

Two degrees of freedom result from pushing and pulling the secondary (outer) backbones using three actuators located proximally. The push-pull actuation modes help prevent the backbones from buckling while satisfying structural statics [2]. The second 2-dof SLU is appended to the end of the first SLU. The backbones of this second SLU pass through the hollow backbones of the first SLU. Attached to the end of the second SLU is a gripper that is actuated via a wire passed through the hollow central backbone. This mechanical composition allows for a high payload capacity with a small size [3]. The existing SLU prototypes are 4.2 mm in diameter.

The aforementioned actuators (seven actuators for four dof and a gripper) are encased atop the shaft in a compact cylindrical actuation unit (shown in Figure 4). As mentioned, the shaft-actuation unit assembly is also teleoperable, with four dof via four actuators, leading to eight dof total for the robot. The ZΘ stage allows for translation along and rotation about the shaft respectively. Finally, a passive universal joint mounted on a five bar mechanism gives the shaft XY mobility. This stage also stabilizes the shaft against lateral perturbations.

Each Snake Robot is actuated by 11 dc motors, accounted for by two three-axis SLUs, a gripper, a two-axis ZΘ stage, and a two-axis five bar mechanism. In our setup, two seven-axis da Vinci masters are used to command two Snake slaves for bimanual control. The pair of da Vinci masters we are using is an engineering version that did not originally include a controller.

1.3 Scalable Motor Controller for the Snake Robot

All told, the low-level controller for the aforementioned prototype must be scalable to handle at least 36 axes (i.e. two 11-axis Snakes Robots and two seven-axis da Vinci masters). A significant reduction in dimensionality and hardware complexity was achieved via the push-pull actuation of flexible wires combined with derived kinematics [1], as opposed to

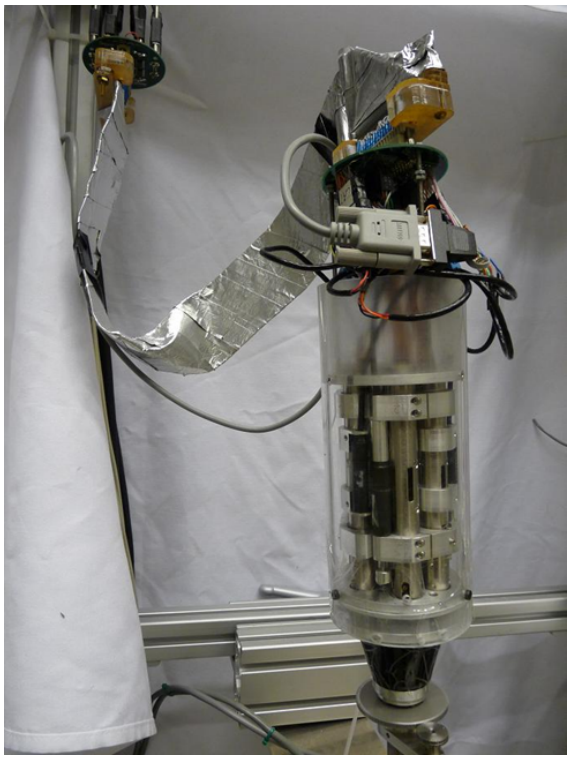

Figure 4: Snake Robot actuation unit

actuation of several precision joints. Nevertheless, the number of control axes for the Snake Robot remains considerable, as does its associated cabling. One can envision the increases in both as systems are devised for more sophisticated surgical tasks. By distributing I/O over an IEEE 1394 (FireWire) bus to nodes with low-latency field-programmable gate arrays (FPGAs), and by centralizing processing, the control system hardware proposed here aims to mitigate the potential problems in robot mobility and reliability that arise with increasing structural complexity.

The efforts presented here and in [18, 19, 21] originate in part from the need to improve the old multi-axis controller (Figure 5) [2] of the Snake Robot and replicate the controller for other research projects. The old controller utilizes a centralized I/O arrangement, whereby command and feedback signals are transmitted in raw analog form over long cables running between the robot and the computer. The I/O devices reside on custom circuit boards, which in turn are directly attached to the computer via its ISA bus. Though the design is conceptually straightforward, the cumbersome wiring associated with it introduces complications such as noise, cable drag, reduced reliability, and greater construction effort. The debug space is vast as there are many candidates for connectivity problems, so this approach limits the ability to develop increasingly dexterous surgical robots. Figure 6 is a photo of the Snake Robot that captures this scenario.

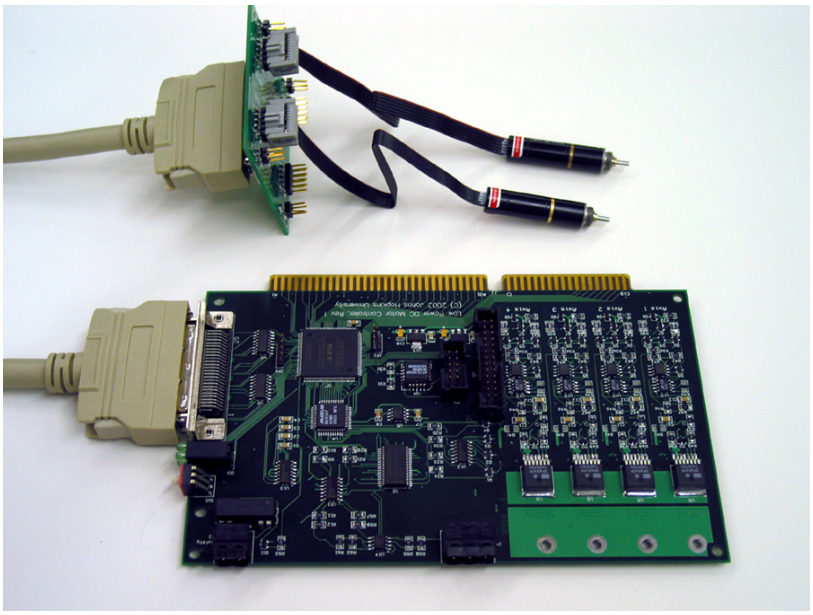

Figure 5: Original ISA-based Snake Robot motor controller

1.4 Motivations

The long term benefits of developing a control system using IEEE 1394 are multifold. The high speed serial bus encourages the distributed I/O and centralized processing architecture. One advantage of this approach is that the I/O processing logic is simple and requires little maintenance. Signal integrity is improved because digitization occurs near the actuators, reducing the potential for noise corruption.

Cable complexity is greatly reduced because distributed I/O hardware is accessed through a serial link. This has several benefits in educational and research environments. Less effort is required in creating cables and breakout boards when new robots are developed. Since I/O hardware is replicated, standard wiring conventions are enforced. Robustness is improved overall because there is less potential for wiring problems and less cable drag affecting mobility.

Parallel buses limit the number of I/O channels that can be connected to one computer. Using the old ISA-based controller as an example, an industrial-grade computer can reliably drive four ISA cards, and the number of channels per card is constrained to four by physical size. As a result, a teleoperated dexterous robot system may need multiple computers to run.

In contrast, IEEE 1394 allows for centralized processing of a large number of multiplexed channels on a single centralized computer, so low-latency local data exchange can be used instead of network communication. The integration allows for a familiar software development environment and a standard API; this alleviates researchers and programmers from

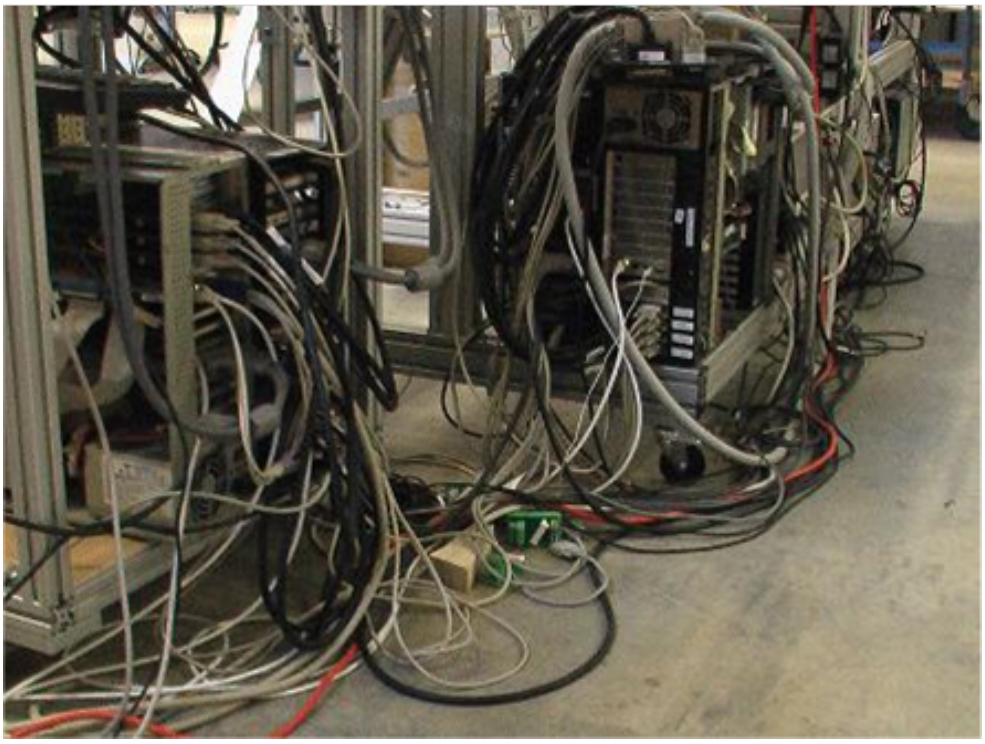

Figure 6: Original Snake Robot cabling with centralized I/O

learning the idiosyncrasies of individual embedded microcontrollers, so they can instead focus on higher-level tasks. Furthermore, this architecture can more readily harness the power of high performance computers.

This work is also intended to facilitate further research by providing a generic interface and scalable mechanism for fine-grain real-time control. Such a custom solution was necessary for servoing the low power dc motors of the Snake Robot [2] because an adequate commercial solution was not available. The solution allows for flexible customization of parameters, investigation of complex control laws, and high-density distributed I/O. It is designed to ease the development of dexterous surgical robots from both hardware and software perspectives.

1.5 Proposed Solution

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 a 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.

A traditional solution to robot control comprises multiple joint-level control boards housed in a central computer through a high-performance parallel bus, such as ISA, Q-Bus, Multibus, or VME. Under this approach, however, the requisite cabling and control processing for surgical robots can become unwieldy as dexterity is increased, due to the increasing degrees of freedom. This can be a prohibitive factor for medical robot research, and furthermore does not extend well to increases in the number of joints.

Motivated by the dexterous Snake Robot, we have developed a real-time low-level control system that takes advantage of a high-speed serial bus, IEEE 1394, and capitalizes on the processing power of contemporary computers. Using a low-latency field-programmable gate array as the link between the computer and a possibly large quantity of I/O devices, all processing tasks are performed on a single computer while I/O tasks are distributed. Centralized processing simplifies software development, and a standard API is developed to augment this feature. Meanwhile, distributed I/O helps cleanly confine most wires to local joint sites thereby substantially reducing complex cabling, a high-risk source of failures. This architecture enables the exploration of complex surgical systems and technologies by allowing for more agility, reliability, and scalability.

We have thus far equated the dexterity of a robot with its degrees of freedom. Though this is not always the case, the level of scalability being sought ultimately requires a proportional increase in degrees of freedom. From the perspective of the low-level controller, there is little distinction between the concepts degrees of freedom, joints, actuators, axes, and channels because they can all be decomposed into I/O signals. Thus in this paper we sometimes use these terms interchangeably.

2 Related Work

Though based on IEEE 1394, the works of [6] and [12] focus on real-time control bandwidth, and not on the physical benefits of distributed I/O and centralized processing. The differences manifest in their use of IEEE 1394 as a link to an onboard computer, contrasting with our use of compact custom electronics. Our work is most similar to that of [16], where custom FPGA-based I/O boards communicate with the computer over IEEE 1394. The bandwidth was sufficient for at least six (possibly 12) dof to be updated at 1 kHz, with unit delay latency. On the other hand, this study emphasizes the physical benefits of the architecture, performance, and scalability. Ref [17] notes that using IEEE 1394 for high bandwidth PET scan data acquisition is viable due to the availability of powerful commodity computers. We agree in principle, though our respective applications are fundamentally different.

2.1 Ethernet-Based Alternatives to IEEE 1394

Fair bus access is incorporated into IEEE 1394 hardware; bus arbitration in Ethernet is nondeterministic, but kilohertz-range motor control is achievable on isolated networks with software modifications [14, 15]. Several Ethernet variations have been developed that make the medium very promising. Powerlink [22] employs a bus manager that schedules 200-μs cycles of isochronous and asynchronous phases. SERCOS approached a communication bottleneck in [9] 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.

A relative newcomer, EtherCAT [23] is an attractive protocol in which the nodes forward and append packets on-the-fly using dedicated hardware and software, resulting in the ability to communicate with 100 axes in 100 μs; [8] is an example showing its potential.

2.2 Other Alternatives to IEEE 1394

Many of the themes highlighted in this paper, including distributed I/O, centralized computing, scalability, and form factor, echo those of [10], which documents the MIRO surgical robot developed by the German Aerospace Center (DLR). Scalability in the MIRO robot is aided by the use of SpaceWire, a 1 GB/s full duplex serial link with latency less than 20 μs. Whereas SpaceWire has been developed by major international space agencies for space-borne systems, we prefer IEEE 1394 as it is a more accessible protocol for research, and its performance is more than adequate for demonstrating our claims. We are particularly more interested in the software-induced latency and overcoming this latency to enhance scalability.

PCI Express is a fairly new serial interface designed to replace computer expansion buses; a cable-based standard was not fully established at the time of the designs presented in this paper. PCI Express supports real-time applications such as the industrial control example in [11]. High data rates are readily available with USB, but its reliance on the host processor for bus level tasks compromises its scalability in real-time control. Conversely, IEEE 1394 self-manages the bus at the physical layer. The Controller Area Network (CAN) [24] bus is well-suited for real-time control and has been widely used, but its bandwidth is limited to 1 Mbps. Though not a serial bus, CompactPCI is an industrial backplane interface capable of 132 MB/s throughputs, used notably in space systems by NASA in transitioning from VMEbus [20].

3 Selection of IEEE 1394

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 MBps, or approximately 100 Mbps. This is not difficult to achieve with modern high-speed serial networks, such as IEEE 1394 (up to 400 or 800 Mbps for 1394a or 1394b, respectively), USB 2.0 (up to 480 Mbps) or Ethernet (10, 100, or 1000 Mbps).

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, though asynchronous mode is used instead as it allows for even faster access rates. It is an effective solution for real-time control, as shown in [6, 12], and by its use in fly-by-wire systems [7]. In the present work, the minimum requirement is to support all of the Snake Robot I/O lines with a system bandwidth of one kilohertz.

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.

A potential drawback of IEEE 1394 is 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 move slowly and infrequently compared to industrial robots. Currently, the Ethernet variations described above provide better cabling options. Ethernet, as well as USB, also have the advantage of market dominance. 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. Many of these Ethernet variations also support daisy-chaining without the use of switches. Given the aforementioned considerations, we ultimately selected IEEE 1394 while noting that it is not necessarily the single best choice. We furthermore chose 1394a, rather than the faster 1394b, because it provided

ample bandwidth and the lower signal frequencies simplified the hardware design.

At the time of our initial evaluation (ca. 2006), 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.

4 Centralized Processing, Distributed I/O

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 7) 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.

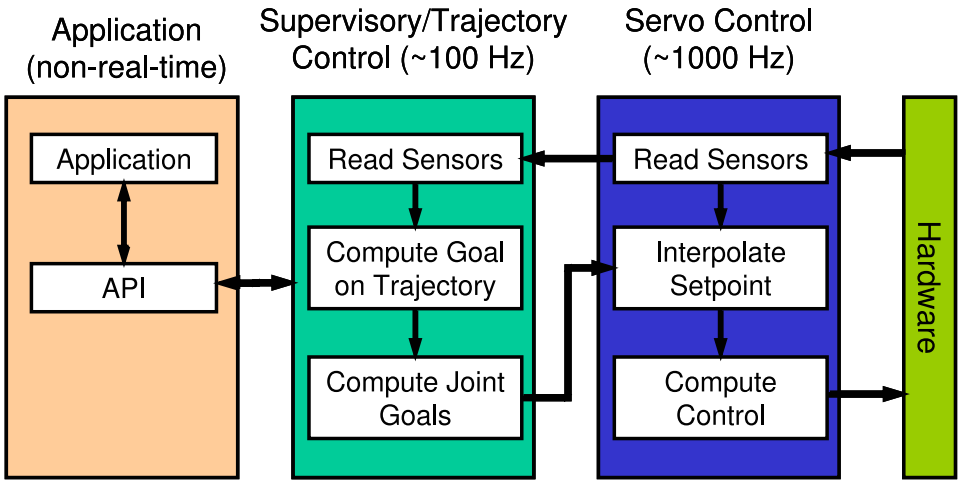

Figure 7: Robot control hierarchy

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.

Given the recent advances in processor performance, especially the move to multi-core architectures, coupled with the extraordinarily high data rates of modern serial networks, we advocate a new approach for robot control: centralized processing and distributed I/O (Figure 8). This can be achieved by replacing the microprocessors with FPGAs that provide direct, low-latency, interfaces between the high-speed serial network and the I/O hardware. As discussed earlier, this preserves the advantages of reduced cabling, while allowing all software to be implemented on a single high-performance computer that contains a familiar software development environment, freeing developers from having to learn the idiosyncrasies of the various embedded microprocessors. Another key factor in realizing real-time centralized processing is the availability of low-cost real-time operating systems, such as those based on Linux.

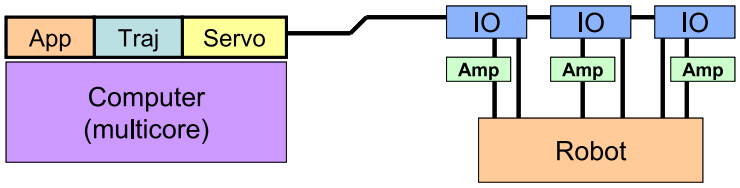

Figure 8: Centralized processing, distributed I/O architecture

A key motivation for building a novel control system is to ease the process of developing multi-axis robots in terms of both hardware and software construction. The hardware provides fine-grain real-time control over a large number of motors, with I/O conversion tasks delegated to the actuator sites. Using IEEE 1394, the hardware confines raw analog signals to those sites and multiplexes the digital data for all channels over a high speed serial connection to a single computer.

5 Hardware Design

5.1 System Overview

Figure 9 provides an overview of the control system, with I/O conversion distributed away from the computer to the actuator sites. Each node on the bus contains multiple channels (or axes of control, and described as the amplifier section below). Nodes can be added to the system by daisy-chaining or by direct connection to the computer—each node contains two IEEE 1394 ports for this reason. The bus is attached to a real-time computer that reads feedback signals from the channels, generates actuation commands, and writes them to their respective channels. The completed controller hardware for the Snake Robot is shown in the photo (Figure 10). Note that this particular hardware has been designed to physically integrate on the top of the Snake Robot actuation unit; other form factors have been developed for general use.

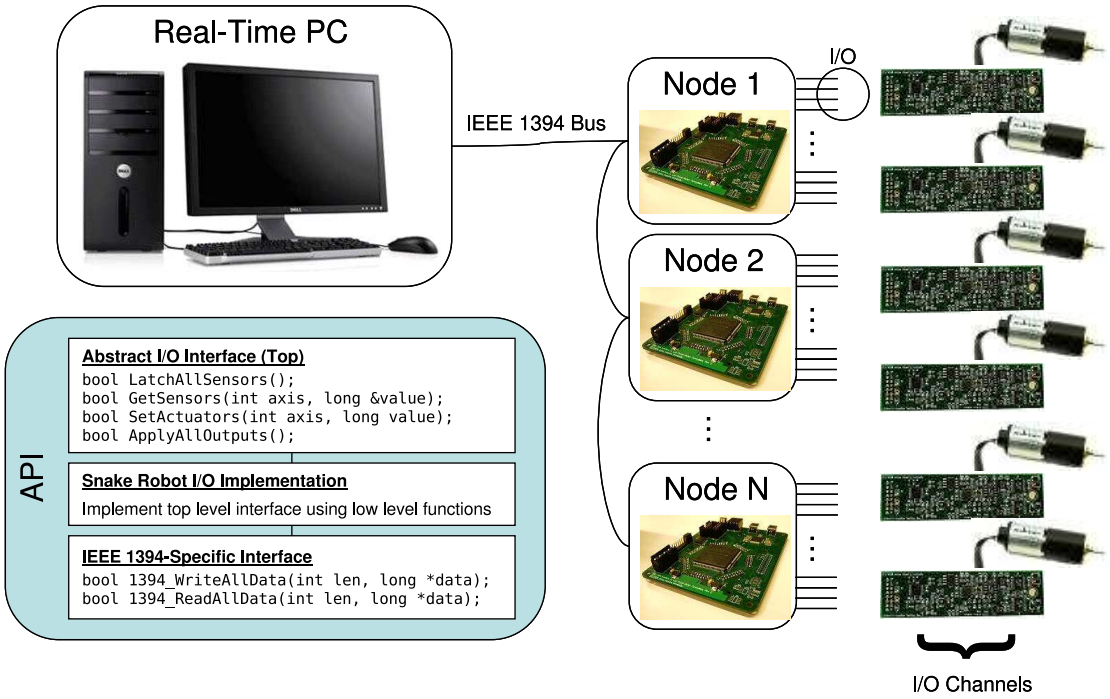

Figure 9: Conceptual overview of the control system

5.2 Amplifier Section

Depicted in Figures 11 and 12, the amplifier section contains the power amplification and I/O components required to control one dc motor. It is referred to as an I/O channel from the perspective of the digital section, which is detailed in the next subsection. Among the various I/O signals, one is used to enable or disable the motor, and one used to signal an op amp fault, e.g. due to overheating. The power amplifier is implemented as bridge amplifier [28]; two op amp devices are configured in such a way as to require only one motor power supply. The midpoint voltage of this supply, denoted VM/2, is used as the zero level of the motor. That is, to stop the motor, both the positive and negative terminals of the motor are set to VM/2. Depending on the desired magnitude and direction of rotation, the motor command sets the master op amp voltage to some value above or below VM/2. In turn, the slave op amp voltage is driven an equal magnitude but opposite direction away from VM/2, simulating the presence of a positive and negative voltage source. Extensive tests have shown some asymmetric behavior and saturation at the supply voltage rails [26], but we expect to overcome this issue in the closed PID loop.

Other fundamental I/O components include an analog-to-digital converter (ADC), a digital-to-analog converter (DAC), and a digital potentiometer. The dual-channel ADC digitizes analog feedback, namely the potentiometer voltage that measures absolute actuator position, and the motor current, which can be used to determine torque. The DAC

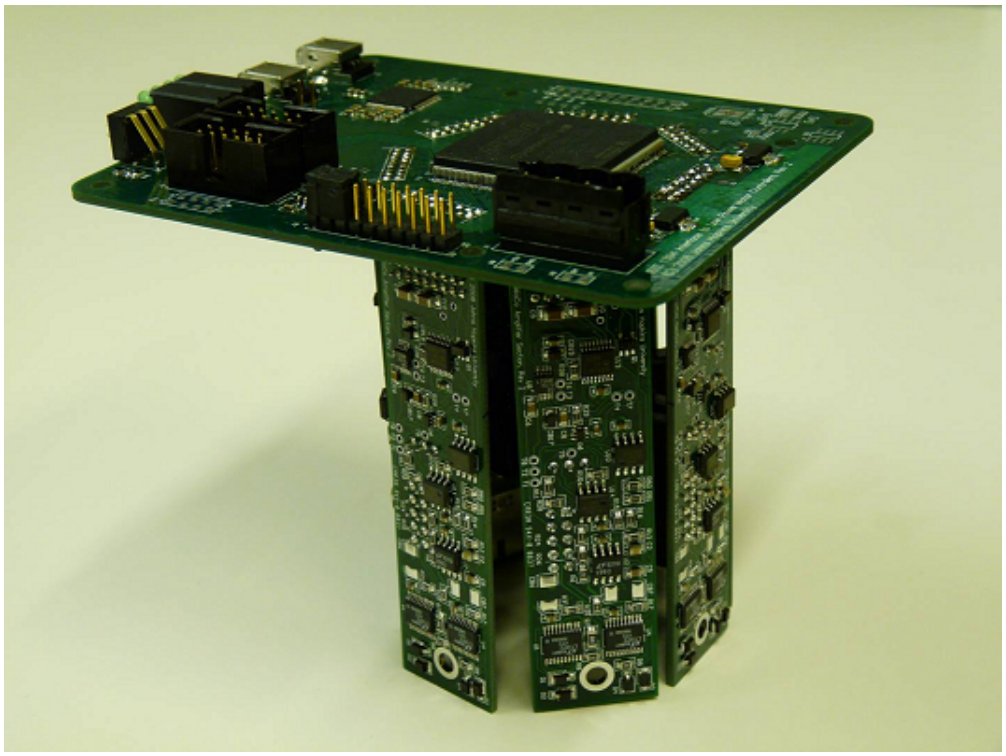

Figure 10: Photo of the completed controller hardware

provides voltage commands for the motor speed, as well as the motor current limit signal. The board outputs incremental encoder pulses, properly level translated to ensure that the FPGA input voltage limit (3.3 V) is satisfied. The FPGA decodes the quadrature encoder pulses to compute motor displacement and velocity measurements.

The digital potentiometer is used to program the gain on motor current feedback from the power amplifier. The intent is to pretune the gain for a specific motor and store that setting on its nonvolatile, 14-word, 16-bit EEPROM for operational use. Calibration procedures and measurement results are documented in [26]. In addition to a programmable wiper and EEPROM, the digital potentiometer also contains two programmable digital outputs. One of these, labeled O1, is used to select between control modes. Setting this output low selects torque control, in which case current feedback into the motor amplifier is shut off in order to maintain constant motor torque. A high setting selects speed control; motor current is fed back into the amplifier to maintain constant speed.

The ability to program the amplifier settings allows for adaptation to other robots with low-power motors. The hardware modifications required, e.g. changing the value of the motor current sense resistor, are of minimal difficulty and cost. This programmability, combined with the current and anticipated need for many such boards, serves as motivation towards a custom solution. The distributed I/O architecture encourages modular one-board-per-motor

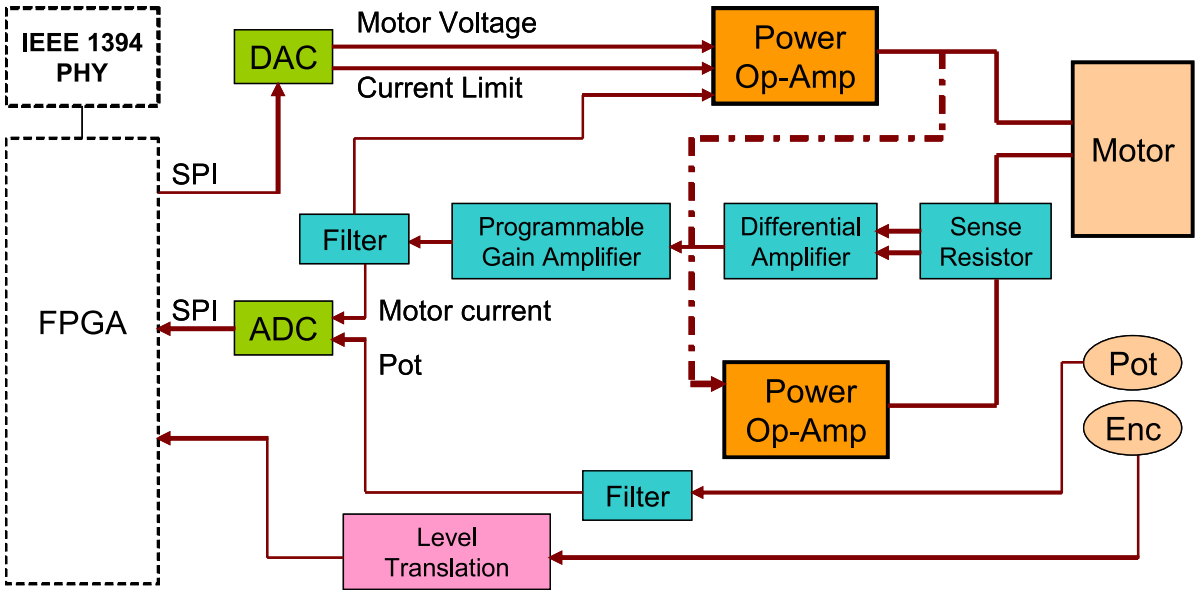

Figure 11: Block diagram of the amplifier section (FPGA and PHY chip also shown)

Figure 12: Photo of the completed amplifier section

design, making the assembly compact and the boards readily replaceable.

5.3 Digital Section

The digital section (Figure 13), which we loosely refer to as a node following IEEE 1394 parlance, contains circuitry for accessing the I/O channels (amplifier sections) and handling bus transactions. IEEE 1394 allows up to 63 nodes per bus; each Snake Robot requires two nodes, one for the actuation unit and the other for the ZΘ and five bar stages combined—a tally of four nodes for two Snake Robots. Multiple buses can be used for yet larger numbers of axes or for heterogeneous control environments. Though not necessary for the Snake Robot, improved bus topologies over a daisy-chain configuration can be achieved by including greater than two ports per node.

Most of the functionality of each node is implemented as firmware on the FPGA, which

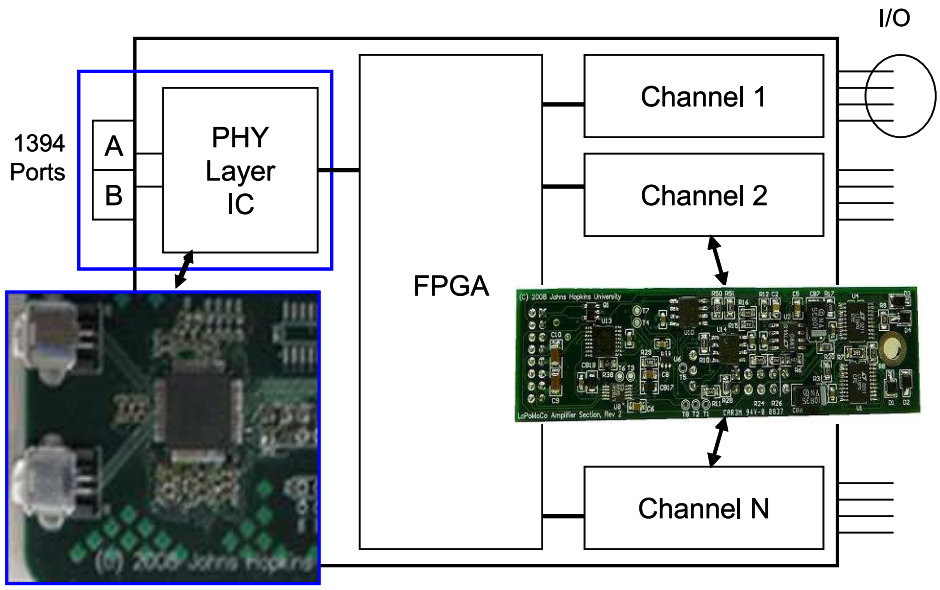

Figure 13: Block diagram of a node

serves as a low-latency interface between the channel I/O ports and the bus. The FPGA receives packets from the bus, responds to them, and communicates with the I/O devices. The computer accesses channel data through control and data registers on the FPGA. The maximum number of channels accessible on one node is governed in large part by the characteristics of the FPGA, such as the number of logic elements, number of general purpose I/O pins, and speed. This design uses the Altera Cyclone II EP2C8Q208C7N [27], a low-cost model with 8256 logic elements and 138 I/O pins. It is capable of clock speeds of up to 260 MHz, though this depends heavily on firmware complexity. Under our current firmware implementation, the Cyclone II is comfortably capable of handling the seven channels of a Snake Robot actuation unit at about double the required 49.152 MHz clock frequency and 60% resource utilization. Several I/O pins remained as spares and are routed out to one of several header pins or test points. Nevertheless, more powerful FPGAs are favorable, and subsequent designs utilize higher-end devices. The FPGA is programmed through a JTAG connector during development, and a separate connector is used for storing completed firmware into a configuration PROM. Specifications allow for these connectors to be merged; though we did not do this initially due to a conservative design approach, the feature is exploited in our later work.

The Texas Instruments TSB41AB2 is a two-port IEEE 1394a physical layer integrated circuit that converts between digital data and the analog signals used to convey them at high speed (400 Mbps) over the bus. The FPGA sends and receives data over the bus by interacting with this device, specifically via control lines and an eight-bit data bus. The device also generates a 49.152 MHz clock, derived from a 24.576 MHz crystal, which the FPGA uses as its system clock. In this way all communications between the various I/O

devices and the physical layer chip are synchronized. Due to the relatively high speeds that the device is capable of, careful attention was paid to its layout on the PCB, particularly around the impedance-controlled differential analog lines, termination resistors, clock I/Os, and bypass capacitors.

Also included in the digital section is a four-bit rotary switch used to uniquely identify a node to control software. Although nodes are uniquely identified via the IEEE 1394 protocol, the address assignment is dynamic. An array of LEDs is used to indicate the status of each of the seven amplifiers; a lit LED signifies that the corresponding amplifier is enabled, while an unlit LED means that the amplifier is disabled, possibly due to assertion of its fault line.

For noise isolation, and to facilitate emergency shutdowns, the motor and digital voltages are drawn from separate regulated supplies. Power from the bus is not used for these reasons, as well as to simplify the development effort, but future designs may draw bus power.

5.4 Implementation

We used an FPGA development board (Altera UP3, Figure 14) for the first prototype to reduce our development risk. Realistically, we expected that our system would not work the first time we connected a computer 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 (Figure 15) that contained the IEEE 1394 physical layer chip.

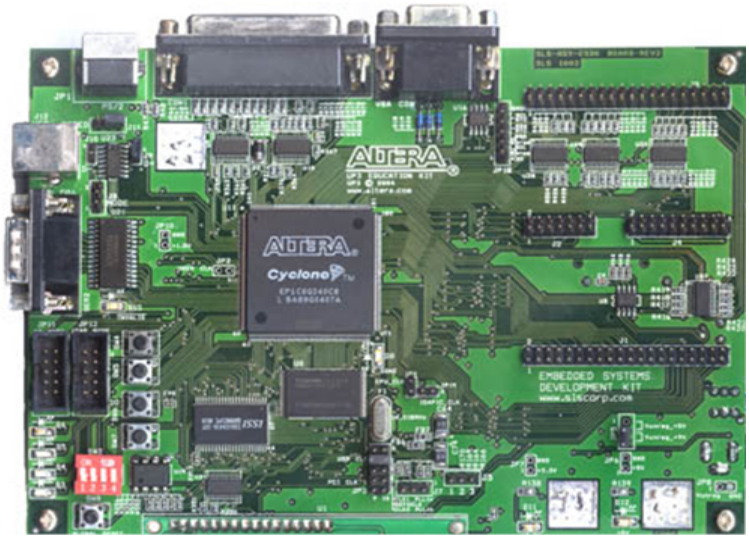

Figure 14: Altera UP3 FPGA development board

The design files for this initial phase are designated as Revision 1 ("-rev1"). This set consists of the daughterboard, the firmware for the FPGA development board, and the first

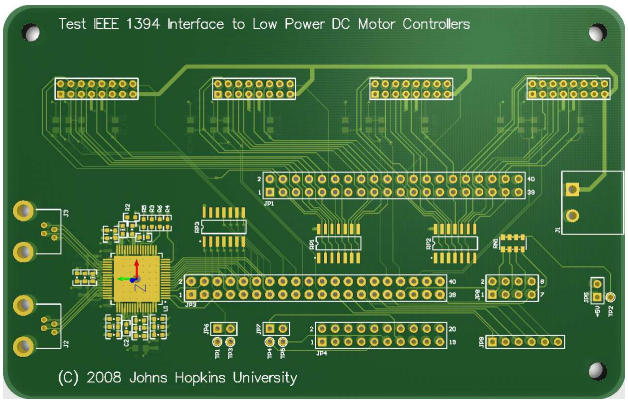

Figure 15: Custom daughterboard containing IEEE 1394 physical layer chip

revision of the amplifier section. The final revisions discussed in this paper are labeled as Revision 2 and include the complete digital section, its firmware, and a revised amplifier section. Revision 2 of the amplifier section represents extensive repairs and improvements over its predecessor.

6 Firmware Section (FPGA)

6.1 IEEE 1394 State Machine

Figure 16 shows the IEEE 1394 state machine, illustrating the top level of the FPGA operation. It describes how the FPGA handles bus transactions and triggers activity on the rest of the board based on such events. As the FPGA plays a passive role in the system, action is initiated when the computer requests a data transaction, for instance a sensor read or an actuator write. Once the FPGA on the addressed node receives the request, it responds immediately with an acknowledgement, as required by the IEEE 1394 protocol. In the case of read requests, the FPGA then fetches data from an intermediate buffer and sends them to the computer with a timestamp, indicating the number of 49.152 MHz clocks cycles that have elapsed since the previous read request. As described in further detail below, the contents of the intermediate read buffer are refreshed continuously to preserve real-time performance. For write requests, the FPGA loads the appropriate buffers and triggers the corresponding channel I/O devices, which in turn access the buffers for their data.

In accordance with the IEEE 1394 protocol, the FPGA firmware verifies incoming CRC values and appends CRC values to outgoing packets. Checksum failure is expected to be an extremely rare occurrence, so given the real-time nature of the system, the defined behavior is to silently drop these packets. For simplicity of implementation, and to minimize FPGA resource consumption, only a subset of the IEEE 1394 protocol is supported. Recognized

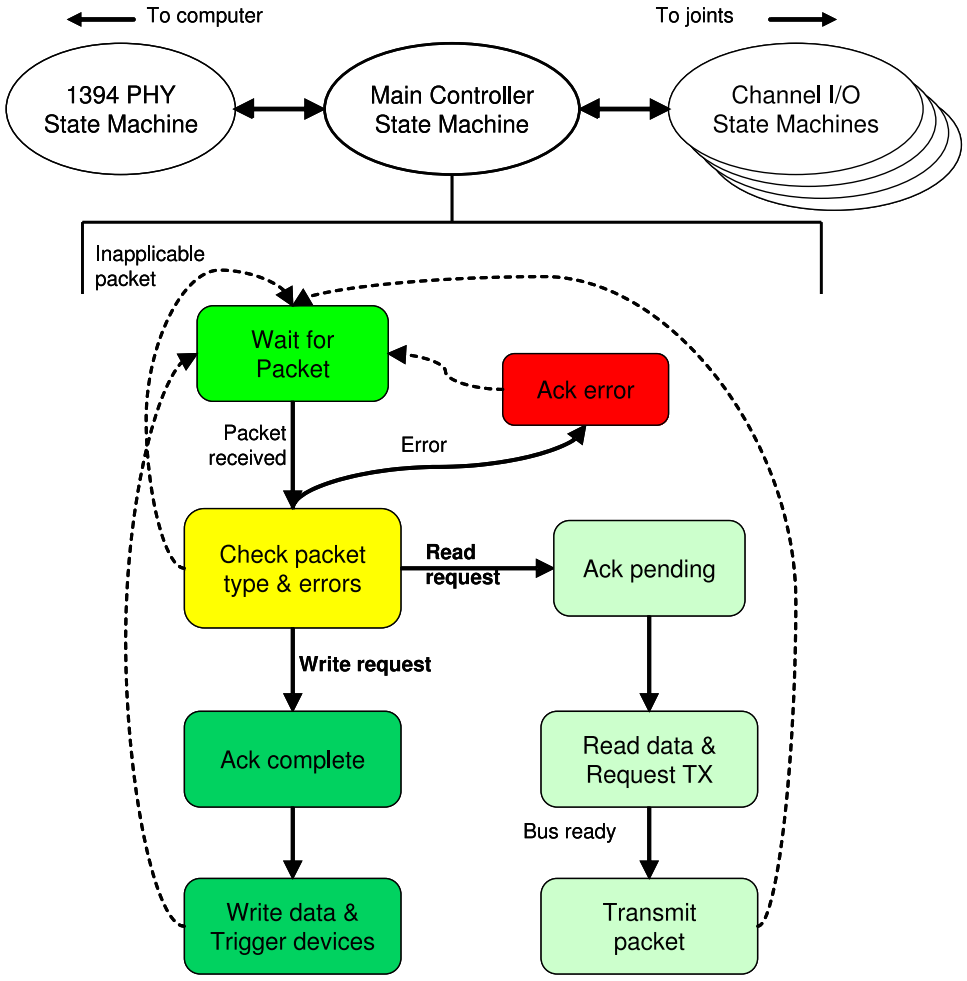

Figure 16: FPGA structure and operation

transaction types as of this writing include quadlet read, quadlet write, block read, and block write. In particular, reads are of the concatenated variety, meaning that once a node receives a read request, it responds immediately without relinquishing control of the bus. This is in contrast to split read transactions, where a node may release the bus and send the response at a later time. Writes are unified; that is, the FPGA acknowledgment to a write request also indicates that the transaction is complete. This contrasts with a protocol provision that allows for a delayed response, in case a checksum failure is detected, for example. Finally, attempts have been made in the firmware implementation to allow the FPGA to accept block transactions of any size. While such an effort may be helpful for adapting the firmware to other boards, for any specific board the behavior is ultimately interwoven with the number of I/O signals per channel and number of channels on the board. Thus only specific block sizes should be requested for proper functionality.

6.2 Channel Modules

As detailed in the description of the amplifier section, each channel contains one control axis worth of I/O, namely an ADC, DAC, digital potentiometer, encoder lines, amplifier enable line, and amplifier status line. The FPGA interacts with the ADC, DAC, and digital potentiometer via the Serial Peripheral Interface (SPI). The encoder signals originate as an analog voltage, but arrive at the FPGA in digital form after level translation in the amplifier section. Counter logic on the FPGA converts the incremental encoder pulses into motor position and velocity measurements. The enable and status lines are simple digital signals.

Communication with these I/O devices is performed in parallel with the IEEE 1394 state machine described above. The read (input) devices, namely the ADC and encoder, run continuously and transfer their data into FPGA buffers as soon as conversion or computation results are complete. The ADC is utilized in this manner because its conversion time, roughly 2 μs, is long compared to the bus transaction time, so it is not advisable to start a new conversion and wait for the result whenever a read request is received. This implies that an ADC conversion result can be up to 2 μs stale, but this is negligible relative to the Snake Robot servo frequency of 1 kHz (1 ms). The write (output) devices, namely the DAC and digital potentiometer, wait idly for write requests, transfer write data from FPGA buffers into scratch registers when a request arrives, and perform the operation. The digital potentiometer functions as both a read and a write device, though it is not read continuously.

While each SPI device is run in separate submodules, all of the I/O devices for a channel are collectively managed by a higher level state machine. This larger module is replicated for each amplifier section and runs independently of any others. It is responsible for arbitrating the shared SPI lines of each channel's respective ADC, DAC, and digital potentiometer. The ADC and DAC accesses can run simultaneously because even though they share some SPI signals, their data move in different directions (the former on the SDO line and the latter on SDI) and thus do not conflict. This is a valuable property because both devices are involved in real-time operation. Since the digital potentiometer is both a read and a write device, however, its accesses must be handled as a special case. A read request from the computer targeting the digital potentiometer will instruct the FPGA to pause ADC conversions, read the digital potentiometer data into an FPGA register, and then resume ADC conversions. The computer then sends a read request for the stored value. A similar procedure is followed for digital potentiometer write requests by pausing the DAC temporarily. This extra complexity does not affect real-time performance because digital potentiometer accesses should be a relatively rare event.

Another special case is establishing the motor current limit through the DAC. Because the same dual-channel DAC is also used for setting the motor command, both channels cannot be written simultaneously, and thus must be performed in separate transactions. While such a limitation would seem to compromise real-time performance, setting of the current limit is expected to be a seldom act, for example, as an initialization step. Under normal operation only the motor command is set.

6.3 Global I/O Module

The amplifier enable lines and fault status flags were previously described in conjunction with their associated channels. However, in order to combine these signals with global events, they are passed on to higher level firmware for management. Introducing an enable mask allows different combinations of amplifiers to be enabled (or disabled) arbitrarily, so that it can be done in software using a single write transaction. Absent of this feature, separate transactions would be required for different amplifier enable requests. Managing the enable lines globally allows us to disable them automatically based on a watchdog timeout, discussed below. It also facilitates convenient grouping of the enable lines, status lines, watchdog timeout flag, and board identifier in order to report them as a single status register. Finally, a logic combination of enable and fault signals are used to set the LED array on the board.

The global I/O module includes an optional watchdog resource, which is activated via software by setting the desired timeout period. The value is in terms of a 5.2083333 μs (49.152/256 MHz) counter. As the counter is an unsigned 16-bit integer, the longest timeout period that can be specified is approximately 0.34 seconds. A timeout occurs when the value of the free running counter matches or exceeds that of the timeout period, at which point all amplifiers are disabled. The counter is cleared whenever a write transaction is received, so timeouts are avoided so long as write transactions are issued at a rate faster than the timeout period. Watchdog functionality is disabled by setting the timeout period to zero, the default setting.

Two seldom used features include the ability to access the registers of the IEEE 1394 physical layer chip, and a read-only version number register that is meant to be set during firmware synthesis. This latter function was quickly deprecated in favor of using software to store version numbers in the digital potentiometer EEPROM.

6.4 Address Maps

All of the I/O resources described thus far are directly accessible by computer software via quadlet transfers, with the desired device specified by an appropriate value in the address field of the packet header. In all cases, the FPGA checks the destination ID field in the packet header and only acts if the value matches the local node ID, which is automatically assigned by the physical layer. Furthermore, it only acts if the specified transaction code is supported. Otherwise packets are considered inapplicable to the node and are silently ignored. When active, the FPGA reads data from or writes data to intermediate buffers as described previously.

The address field for quadlet transfers is interpreted as follows: bits 7-4 address a channel, while bits 0-4 address a device on that channel. The address map is listed in Table 1. One detail of note is that channel devices begin at channel address 1, as channel address 0 specifies the global I/O module, whose address map is also shown in the table. The address

Bits 7-4	Bits 3-0	Description
0 (Global Registers)	0	Status Register
	1	IEEE 1394 Physical Layer Chip Control Register
	2	IEEE 1394 Physical Layer Chip Data Register
	3	Watchdog Timeout Period Register
	4	Version Number Register (deprecated)
1-7 (Channel Select)	0	ADC Data Register
	1	DAC Control Register
	2	Digital Potentiometer Control Register
	3	Digital Potentiometer Data Register
	4	Quadrature Encoder Preload Register
	5	Quadrature Encoder Data Register
	6	Encoder Period Data Register
	7	Encoder Frequency Data Register (not implemented)

Table 1: Device map of quadlet transaction address field

segmentation reveals our assumptions on the number of devices per channel and the number of channels per node.

In contrast, the devices involved in block reads and writes are fixed according to its position in the data block. Also, only certain devices are accessible through block transfers, i.e. those necessary for real-time operation. Figure 17 depicts the respective packet formats expected by the FPGA, as well as how it interprets the data fields. The FPGA firmware is meant to handle write blocks of a fixed size due to the simplicity and reliability of such an approach. Thus the MSB of each data quadlet is interpreted as a valid bit, which is set by software to indicate to the FPGA whether or not the corresponding device is to be written.

7 Software API

7.1 Block Diagram

We developed a generic, intuitive API for the control hardware described above. The interface has the three-layer hierarchy shown in Figure 18. The top layer consists of abstract I/O operations that can be used for different robots. It includes commands to latch all sensors and to apply all outputs, which are often supported by the hardware (e.g., simultaneously sampled ADCs and double-buffered DACs). All read and write operations are performed for a single axis at a time, requiring only primitive C data types (e.g., int, long).

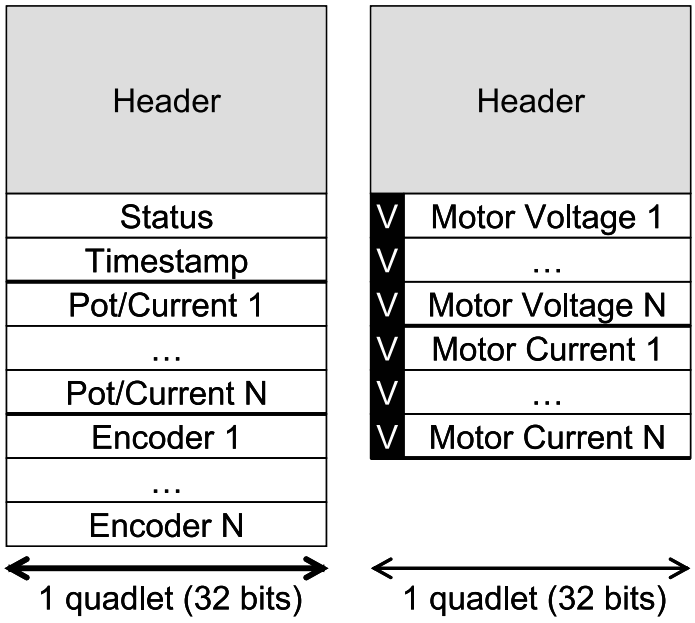

Figure 17: Block read (left) and write (right) packet formats as interpreted by the FPGA

The second layer, which implements the abstract interface for the Snake Robot, is customized to work with data blocks, since for efficiency reasons the data for the multiple axes of a node are bundled into a single bus transaction. This layer provides access to individual axes via local buffers that are filled by `LatchAllSensors` and emptied by `ApplyAllOutputs`. So that a fixed block size can be used to simplify the FPGA implementation, the second layer maintains a valid bit for each axis that indicates to the FPGA whether or not to write the corresponding axis. The bottom layer contains function calls to the IEEE 1394 API library (libraw1394), though for real-time performance considerations RT-FireWire [12] is an alternative of interest. By carefully designing a general robot control API, the developed software can be easily maintained, and the system can be used in other surgical robots.

7.2 Application Programming Interface

The methods of the API are listed below. The data formats produced or expected by the hardware are diverse, as they are specific to each device. In order to maintain uniformity in the data types of the API arguments and return values, the library manipulates the data in ways that the calling code must handle properly. Some of the less straightforward considerations are detailed as follows.

The ADC, through which analog potentiometer and motor current measurements are obtained, reads voltages in the range of 0 to 2.5 V. However, it reports them as signed 14-bit integers relative to 1.25 V. To match the API convention, the library converts these to

```
// read data from given channel (all channels combined for power status)
unsigned long GetStatus();
unsigned long GetTimestamp();
unsigned long GetMotorCurrent(    unsigned int );
unsigned long GetAnalogPosition(  unsigned int );
unsigned long GetEncoderPosition( unsigned int );
unsigned long GetEncoderVelocity( unsigned int );
unsigned long GetDigitalPotWiper( unsigned int );
unsigned long GetAnalogSwitches(  unsigned int );
unsigned long GetEepromDataRaw(   unsigned int );
float         GetEepromDataFloat( unsigned int );

// write data for given channel (all channels combined for power control)
bool SetPowerControl(                  const unsigned long );
bool SetMotorVoltage(    unsigned int, const unsigned long );
bool SetMotorCurrent(    unsigned int, const unsigned long );
bool SetEncoderPreload(  unsigned int, const unsigned long );
bool SetDigitalPotWiper( unsigned int, const unsigned long );
bool SetAnalogSwitches(  unsigned int, const unsigned long );
bool SetEepromDataRaw(   unsigned int, const unsigned long );
bool SetEepromDataFloat( unsigned int, const float         );

// actual operations on buffer
bool ApplyAllOutputs();
bool LatchAllInputs();
```

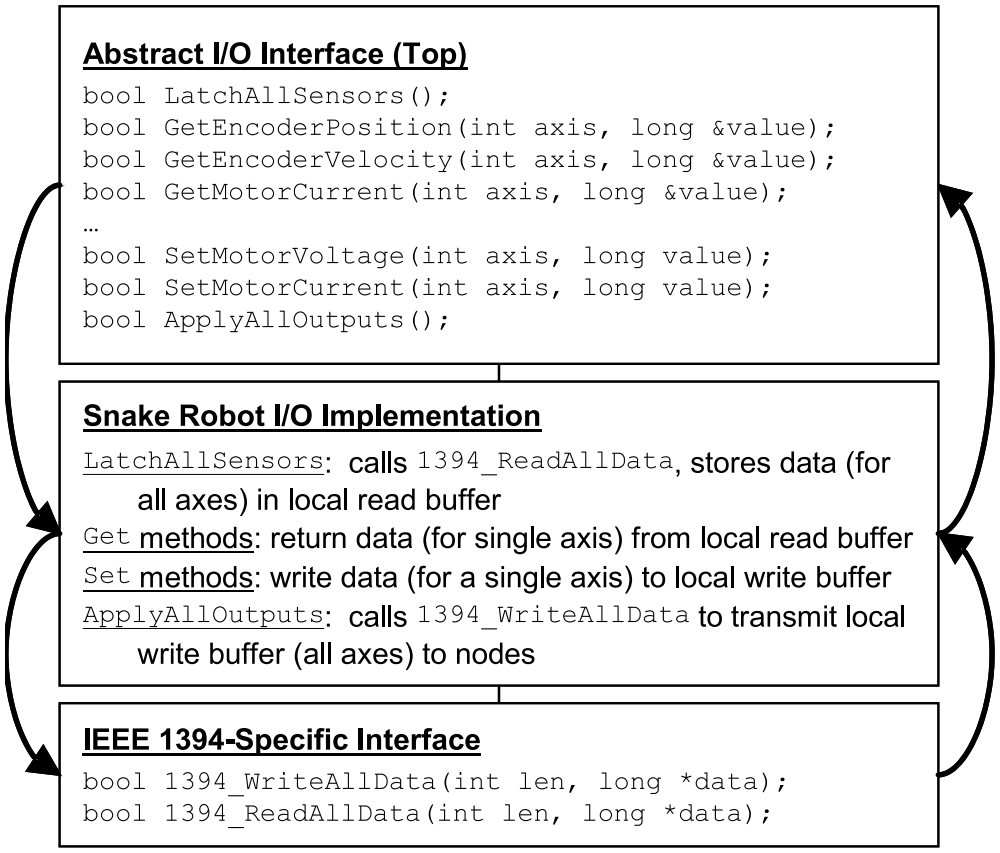

Figure 18: General robot control API

unsigned 14-bit integers by adding to it the midpoint value of the 14-bit range. Stated explicitly, the ADC views the range 0 to 2.5 V as -1.25 to +1.25 V, and reports a corresponding value in the range -8192 to +8191. The library adds 8192, resulting in an API return value in the range 0 to 16383, an unsigned 14-bit integer. The caller must treat this accordingly, as the sign is related to the direction of travel.

The DAC, through which motor commands and motor current limits are programmed, outputs voltages in the range 0 to 2.5 V. It accepts as its inputs 16-bit unsigned integers, making the mapping intuitive. Note however that the direction of the motor depends on whether the DAC output is above or below the midpoint (1.25 V), and its speed depends on the absolute difference from the midpoint. In this sense the motor command can be viewed as a signed value centered about this nonzero midpoint.

The controller provides two types of encoder counts. The main type counts the number of pulses, which can be used to measure incremental movement. Because an overflow flag is asserted when the count drops below 0 or increases beyond the 24-bit counter maximum, it is important to preload the counter with the midpoint value during initialization. The secondary measurement is a 16-bit counter representing the number of 1.302083333 μs (768

Bits 31-28	27-24	23	22-16	15-8	7-0
Unused(4)	Board ID(4)	WD Timeout(1)	Status(7)	EN Mask(8)	Amp EN(8)

Table 2: Status code definition

kHz) clocks elapsed between encoder pulses. This is potentially useful for measuring relatively slow motor velocities, as in this regime there is a resolution problem in simply dividing the number of encoder pulses by the elapsed time. This count should be interpreted as a signed value as it starts from zero and increments or decrements depending on the motor direction.

The digital potentiometer wiper value is an unsigned 10-bit value, where 0 sets the wiper to the A terminal and the maximum value sets it to the B terminal. Given the circuit connections, this means that the motor current feedback gain increases with the wiper value according to

$$G = \frac{12.1\,\Omega + 10\,\Omega \times (wiper/1023)}{1.5\,\Omega + 10\,\Omega \times ((1023 - wiper)/1023)}. \tag{1}$$

The $10\,\Omega$ factor that appears in this formula represents the nominal resistance of the digital potentiometer. Through our tests we have found this value to vary widely by device, so gains should be tuned per board-motor pair. The gain factor for this stage is nominally 2, so a reasonable initial wiper value for the Snake Robot would be 375.

An EEPROM is embedded within each channel, so accessing it requires two tiers of addressing, one to specify the channel and the other to specify the EEPROM location within that channel. Thus the index argument for EEPROM access is interpreted slightly differently. Bits 7-4 of the index now select the channel, and bits 3-0 select the EEPROM word within that channel. In contrast, bits 3-0 select the channel for the other devices. The API also allows higher level code to store single-precision floating-point values in EEPROM. It does so by splitting the 32-bit data type into two 16-bit words at the library level, using two EEPROM locations for their storage. Note that BCD and floating-point representations occupy the same memory space.

Each block read via `LatchAllSensors` fetches, in addition to ADC and encoder data, a status and a timestamp. The status (Table 2) is a 32-bit value that is a combination of amplifier enable bits and mask, amplifier status flags, watchdog timeout flag, and board ID (set by the onboard rotary switch). The timestamp, meanwhile, is a 32-bit value indicating the number of 49.152 MHz clock cycles elapsed since the previous `LatchAllSensors` call.

As of this writing the API has not been tested for CISST compatibility, but integration is in progress.

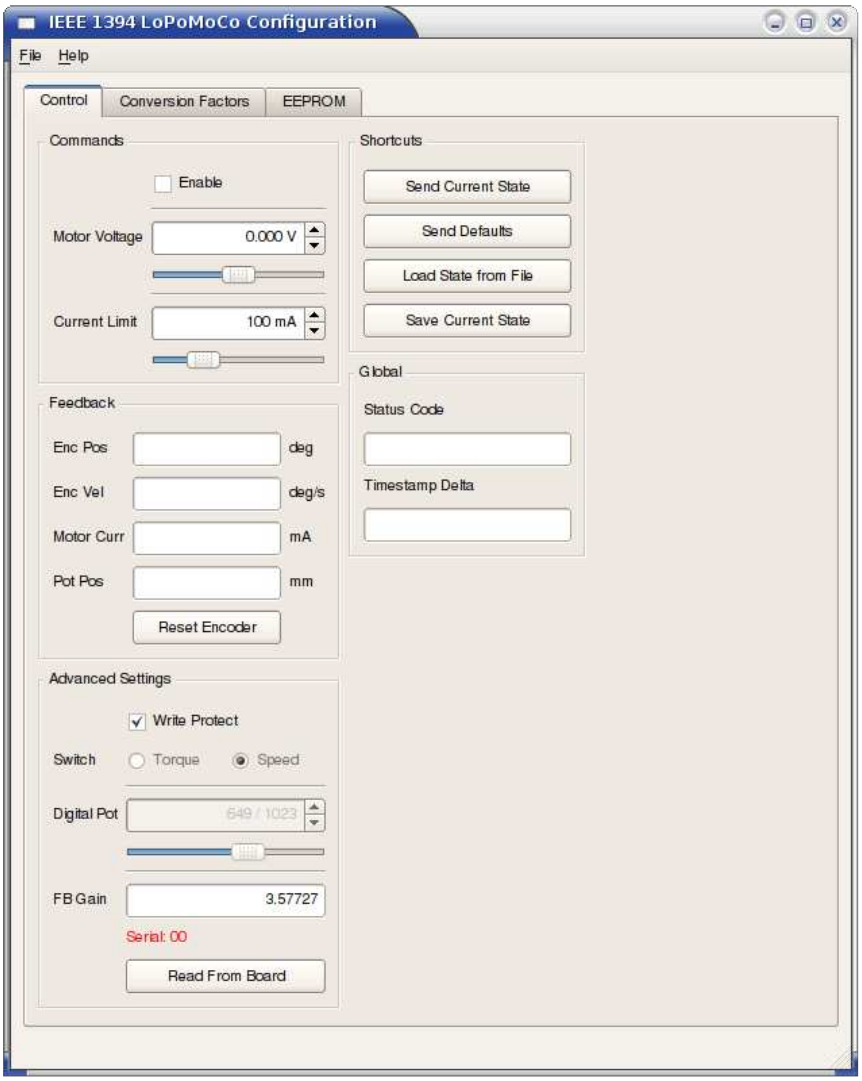

Figure 19: Test and configuration program

7.3 Demonstration

A Qt-based graphical user interface program (Figure 19) is being developed to enable testing and configuration of the various features of the board. It allows the user to control the motor voltage and current limit for each axis, as well as view their respective analog potentiometer, motor current, and encoder measurements. On the same screen the user can adjust the digital potentiometers and view the resulting feedback gains. The GUI offers a convenient mechanism for switching between speed and torque control as well. A dedicated EEPROM tab contains controls that allow for reading and writing the EEPROM on any axis, in both binary and floating-point representations.

This demonstration program is also intended to test the API in order to expose and address any remaining issues. Finally, the program serves as a reference implementation for usage of the API. At present the code is not compatible with the CISST libraries, but integration is expected to take place in the near future. All source code, design files, and documentation referred to in this paper are stored in the Electronics SVN repository [25].

8 Experiments

8.1 Setup and Verification

A conventional Linux desktop computer is used for the initial experiments reported below, while a real-time version of Linux, such as the Real-Time Application Interface (RTAI) and QNX, will be used to run the robot control software. The tests were run at 400 Mbps with one node connected to a 2 GHz Pentium 4 computer by a 6-foot IEEE 1394 cable.

As explained previously, the FPGA performs a read transaction as a concatenated read, which entails a request from the computer, an acknowledgment from the node, and a data response from the node (there is also a final acknowledgement from the computer to the node). There is no protocol delay (i.e. no busy wait) between receiving a read request and generating a response. Similarly, write transactions are unified, so there is no delay between the receipt of a write request and the execution of the request. Because I/O device access times are negligible relative to the system bandwidth, loopback tests of the DAC-to-ADC pair and digital I/O consistently return the appropriate values. Bus contention is not expected because (1) the computer is implemented as the bus master and the nodes as slaves; and (2) no devices unrelated to the control system will be attached to the bus.

The amount of various component-wise tests performed is voluminous and are not repeated here. Rather, the findings of these tests have been integrated into this report, while test code and their noteworthy results have been stored in the Electronics SVN repository. The demonstration GUI presented in the previous section represents the culmination of all verification of the hardware components, API, and their integration.

8.2 Quadlet Transfers

Figure 20 shows the timing results of 9,000 iterations of quadlet reads and writes.

The read times appear in three principal bands (30, 42, 49 μs) and the write times in two (27, 42 μs), possibly due to variability in discrete transaction sequences (e.g. request-ack-response); the average read and write times are 34.5 and 30.2 μs respectively—each respective low band is most common. We plan to investigate these regular distribution patterns should they arise under a real-time operating system.

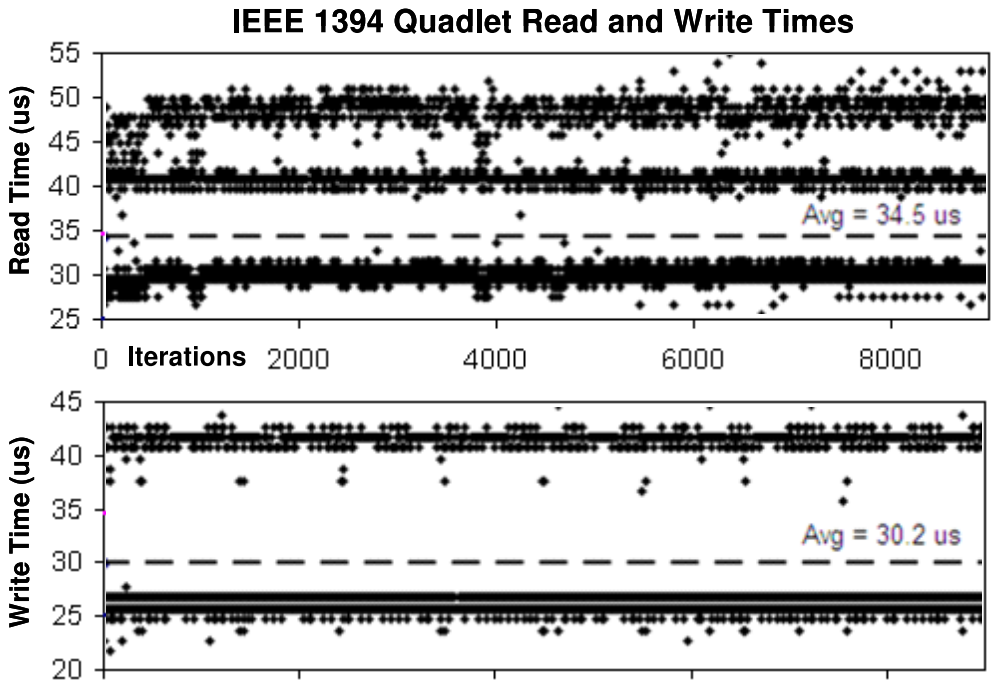

Figure 20: IEEE 1394 quadlet read and write transaction times

We note that there are significant latencies in the bus transactions [18, 19, 21]. In a standard servo control implementation (read-control-write) for a seven-axis robot, a combined read/write time of about 453 μs leaves only 547 μs for control computations at 1 kHz, and is not even feasible at 8 kHz. Given that the bus speed is 400 Mbps, and that the IEEE 1394 physical layer itself should not impose appreciable delays, we conclude that software overheads are a predominant factor in the latency, as in [13]. As we concluded in our previous work, it is necessary to bundle the data for multiple axes into blocks in order to overcome this limitation.

8.3 Block Transfers

Figure 21 shows a raw sampling of read and write times over the full range of allowable block sizes, up to the maximum of 512 quadlets for the 400 Mbps mode. Blocks sizes are measured in quadlets, as IEEE 1394 requires packet sizes to be in multiples of 32 bits. The plots indicate base latencies of about 33.2 μs for reads and 30.7 μs for writes, which closely matches our previous findings.

Based on the slopes of transaction time vs. block size, we compute average speeds of roughly 360 Mbps for block reads and 290 Mbps for block writes. Neither value reaches the nominal 400 Mbps rate due to protocol handshaking and other overheads, but it defies

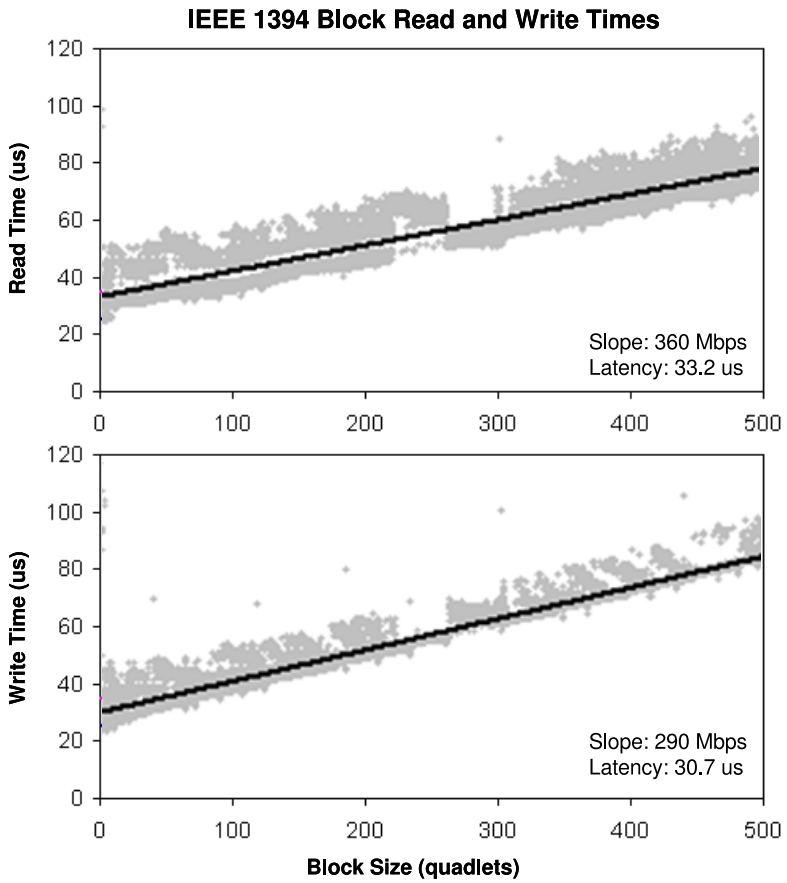

Figure 21: IEEE 1394 transaction times vs. block size

intuition that reads are faster than writes since the former is a slightly more complicated transaction type and incurs greater latency. A possible explanation may involve differences between how the computer and FPGA acquire bus access. We intend to resolve this anomaly in future work. At any rate, the results suggest that the number of axes can be scaled significantly with negligible incremental time consumption.

8.4 Discussion

Variability in the transaction times (about 20 μs, based on the noisiness of the plots) and sporadic but high-value outliers may be caused by underlying operating system activity, as we use conventional Linux for development purposes. The former observation may also be explained by variability in obtaining bus access. A real-time operating system will be used to run robot control software, and we would like to obtain performance measurements under such an environment as well.

Overall, the transaction times are well above theoretical maxima. For example, a quadlet read, 296 total bits, should only require a fraction of a microsecond to complete at 400 Mbps. We believe that software overhead is the primary source of concern in our system, 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 command signals. In a standard read-control-write implementation, a combined block read/write time of about 65 μs would leave 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 supports isochronous transfers, which occur at a frequency of 8 kHz and may therefore be more efficient for control at this frequency.

9 Ongoing and Future Work

As of this writing, the presented controller is complete and ready for integration. The remaining work involves the integration itself. First, the API must be integrated into the CISST Library [29] so that is compatible with existing robot control software, and to facilitate its future use. The next step is to modify the Snake Robot control software to use this API and controller. We hope that our careful efforts in crafting the API will allow for a smooth transition. We would like to experiment with QNX, a real-time Linux-based operating system that is reportedly being used by several robotics groups, by using it to run the new Snake Robot. It would be interesting to see whether usage of a real-time operating system has a significant impact on bus transaction times. Meanwhile, a heatsink-mount must be fabricated in order to physically attach the controller to the robot. While assembling the motor connectors, a key detail to remember is that the pinout of the motors changed after the circuit boards were fabricated.

Presently we expect to achieve real-time performance for one node, but it is uncertain whether this will hold when multiple nodes are connected. One way to alleviate this concern is to employ single transactions that encompass all nodes. This is largely if not entirely an FPGA implementation. The computer can send write commands by broadcasting large packets and allowing each FPGA to extract its own segment of data. Similarly-spirited read transactions appear to be somewhat more challenging to perform due to the nature of the architecture. The feasibility of using isochronous transactions for this purpose should be explored.

Ultimately the goal is to further surgical robotics research. One immediate project enabled by this work is the convenient deployment of an additional Snake Robot for tasks such as camera manipulation. Other potential applications include dexterous ultrasound imaging and ablation. The API will be made compatible with a standard medical robotics framework, the Surgical Assistant Workstation [5].

10 Conclusions

Though parallel buses such as ISA, Q-Bus, Multibus, and VME have become tried-and-true interfaces for robot control, they are increasingly deprecated with the emergence of IEEE 1394, PCI Express, and Ethernet-based protocols, which feature greatly simplified cabling. These high speed serial networks provide higher performance than traditional field buses, such as CAN, SERCOS, and RS-485, which have also been used for real-time control.

The ability of the IEEE 1394 bus to multiplex a large number of data channels helps to reduce wiring complexity, making systems more robust and scalable to many axes of control. Centralized processing eases intra-robot (e.g. master-slave) communication and allows systems to utilize ever-advancing computing power. Such a setup simplifies the software development environment, which is especially advantageous in areas such as research and education, where typical users are not proficient with software development using embedded microprocessors and associated tools.

This paper presents a scalable controller architecture based on IEEE 1394 for communication between the computer and actuated joints. 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 advantages of centralizing processing and distributing I/O via a high-speed serial network are discussed. The concept was demonstrated by a custom controller for the Snake Robot. As the design lends itself towards modularity, the theme served as an influence throughout development. With the potential need for many axes of control in surgical robotics research, modular hardware can help increase availability and flexibility while making the system as a whole easier to troubleshoot.

Our first prototype consisted of three boards: a commercial FPGA development board, a custom daughterboard with an IEEE 1394 physical layer chip, and custom motor power amplifiers. After running a number of tests on this setup, we then constructed a second prototype that combines the first two boards (FPGA and physical layer chip) into a single custom board. We also fabricated a new set of amplifier boards implementing several repairs and design changes.

Preliminary performance data, obtained under a conventional operating system (Linux), suggest that this approach is feasible for real-time control with rates up to several kHz; higher rates, such as 10 kHz, appear to be challenging with the current setup based on the measured latencies.

Though the described control system is not necessarily a novel design given existing technologies, we contend that it will ease the development of dexterous robots and allow researchers to experiment with more robot-assisted surgical tasks.

11 Acknowledgments

Special thanks go to Peter Kazanzides for his comprehensive advisement on this project. I would also like to thank Hamid Wasti of Regan Designs, Inc. (Coeur d'Alene, Idaho) for his expertise on the design and layout of the controller boards, and Ankur Kapoor for sharing his knowledge and experience. Finally I thank Russell Taylor for his overall support of my research and research-related efforts. This work was supported in part by the National Science Foundation (NSF) under Engineering Research Center grant #EEC9731748, NSF grant #MRI0722943, and by Johns Hopkins University internal funds.

References

[1] Simaan, N., R. Taylor, and P. Flint, "A dexterous system for laryngeal surgery," *IEEE Robotics and Automation*, vol. 1, pp. 351-357, 2004.

[2] 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.

[3] Kapoor, A., "Motion constrained control of robots for dexterous surgical tasks," Ph.D. dissertation, Johns Hopkins University, Sep 2007.

[4] Simaan, N., R. Taylor, and P. Flint, "High dexterity snake-like robotic slaves for minimally invasive telesurgery of the upper airway," *MICCAI*, Rennes-Saint-Malo, France, pp. 17-24, Sep 2004.

[5] Vagvolgyi, B., S. DiMaio, A. Deguet, P. Kazanzides, R. Kumar, C. Hasser, and R. Taylor, "The Surgical Assistant Workstation," *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions* (online at http://hdl.handle.net/10380/1466), Sep 2008.

[6] 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*, pp. 1416-1421, Beijing, China, Oct 2006.

[7] 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.

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

[9] 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.

[10] Hagn, U., M. Nickl, S. Jorg, G. Passig, T. Bahls, A. Nothhelfer, F. Hacker, L. Le-Tien, A. Albu-Schaffer, R. Konietschke, M. Grebenstein, R. Warpup, R. Haslinger, M. Frommberger, and G. Hirzinger, "The DLR MIRO: a versatile lightweight robot for surgical applications," *Industrial Robot: An Int'l Journal*, vol. 35, no. 4, pp. 324-336, 2008.

[11] Szydlowski, C., "Implementing PCI Express for industrial control," *RTC Magazine*, vol. 13, Sep 2004.

[12] Zhang, Y., B. Orlic, P. Visser, and J. Broenink, "Hard real-time networking on FireWire," *RT Linux Workshop*, Lille, FR, Nov 2005.

[13] Sarker, M., C. Kim, J. Cho, and 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, Jul 2006.

[14] Schneider, S., "Making Ethernet work in real time," *Sensors Magazine*, vol. 17, no. 11, Nov 2000.

[15] Kerkes, J., "Real-time Ethernet," *Embedded Systems Design*, vol. 14, no. 1, Jan 2001.

[16] Pratt, G., P. Willisson, C. Bolton, and A. Hoffman, "Late motor processing in low-impedance robots: impedance control of series-elastic actuators," *American Control Conference*, vol. 4, pp. 3245-3251, Jun-Jul 2004.

[17] Lewellen, T., C. Laymon, R. Miyaoka, K. Lee, and P. Kinahan, "Design of a Firewire based data acquisition system for use in animal PET scanners," *IEEE Nuclear Science Symposium Conference Record*, vol. 4, pp. 1974-1978, Nov 2001.

[18] Thienphrapa, P. and P. Kazanzides, "A distributed I/O low-level controller for highly-dexterous snake robots," *IEEE BioCAS*, pp. 9-12, Nov 2008.

[19] Kazanzides, P. and P. Thienphrapa, "Centralized processing and distributed I/O for robot control," *IEEE TePRA*, pp. 84-88, Nov 2008.

[20] Walls, B., M. McClelland, S. Persyn, and D. Werner, "Leveraging flight heritage to new CompactPCI space systems: a fusion of architectures," *Digital Avionics Systems*, vol. 2, pp. 8C41-8C47, 2001.

[21] Thienphrapa, P. and P. Kazanzides, "A scalable system for real-time control of dexterous surgical robots," *IEEE TePRA*, pp. 16-22, Nov 2009.

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

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

[24] Controller Area Network: http://www.can-cia.org/.

[25] Electronics SVN repository: https://svn.lcsr.jhu.edu/electronics/.

[26] Huang, R., P. Thienphrapa, and P. Kazanzides, "Calibration of the Snake amplifier board," Internal report, May 2009.

[27] Altera Corporation, "Cyclone II device handbook," 2008.

[28] Kazanzides, P., "Bridge motor amplifier with speed control," Internal report, Dec 2006.

[29] CISST Library Homepage: https://trac.lcsr.jhu.edu/cisst.