12-17-1988

A New Computer Board for Distributed Real-Time Motion Control

Forrest Levin
Evergreen Design

Martin Buehler
Yale University

Louis Whitcomb
Yale University

Daniel E. Koditschek
University of Pennsylvania, kod@seas.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/ese_papers

Recommended Citation

NOTE: At the time of publication, author Daniel Koditschek was affiliated with Yale University. Currently, he is a faculty member in the Department of Electrical and Systems Engineering at the University of Pennsylvania.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/ese_papers/696
For more information, please contact libraryrepository@pobox.upenn.edu.
A New Computer Board for Distributed Real-Time Motion Control

Abstract
This article describes a modular computational engine that has become the workhorse for almost all real-time motion control tasks that we encounter in the Yale Robotics Laboratory. Roughly speaking, these tasks amount to the marshalling of various data - from external sensors; from user specified commands; from motor joint variables - and rapid computation involving their appropriate combinations all in time to produce motor commands that will ensure high performance coordinated movement of many coupled mechanical degrees of freedom…

Comments
NOTE: At the time of publication, author Daniel Koditschek was affiliated with Yale University. Currently, he is a faculty member in the Department of Electrical and Systems Engineering at the University of Pennsylvania.
A New Computer Board for Distributed Real-Time Motion Control

Preprint of an article to appear in
Electronic Systems Design

F. Levin
Evergreen Design

and

M. Bühler, L. Whitcomb, D. E. Koditschek *
Center for Systems Science
Yale University, Department of Electrical Engineering

December 17, 1988

*This work is supported in part by the GMF Robotics Corporation, INMOS Corporation and PMI Motion Technologies, and, in part, by matching funds from the National Science Foundation under the terms of a Presidential Young Investigator Award held by the last author.
1 Introduction

This article describes a modular computational engine that has become the workhorse for almost all real-time motion control tasks that we encounter in the Yale Robotics Laboratory. Roughly speaking, these tasks amount to the marshalling of various data — from external sensors; from user specified commands; from motor joint variables — and rapid computation involving their appropriate combinations all in time to produce motor commands that will ensure high performance coordinated movement of many coupled mechanical degrees of freedom. While our particular applications do not typify those found in the usual industrial setting, the general task of coordinating many degrees of freedom in real-time seems to be nearly universal. Moreover, since this architecture has proven to be extremely cost and time effective, an account of its design origins, development, and use might well be helpful to any small research and development team contemplating a custom board design and facing, as do we, unremitting shortages of manpower and money. Finally, the design is in its β-test phase at the Yale Robotics Laboratory, and we would like to announce its impending commercial debut.

Our solution to the exigencies of high performance motion control rests largely upon the INMOS “Transputer” family of microprocessors. To the powerful built-in computation and communications capabilities of their hardware, INMOS has added a facile and integrated software/development environment extremely well suited to the generation of parallel and concurrent versions of numerically complex algorithms in a message passing environment. Our contribution to the existing technology lies in the design and implementation of a two board set of printed circuits reflecting the I/O and memory requirements of the present applications. Our purpose in writing this article is to alert designers to the intrinsic capabilities of Transputer networks in general, and to provide concrete examples of the use and performance of the Yale XP/DCS board set in specific motion control applications.

1.1 Statement of Problem

Typical servo motors have time constants of between 10 and 100 msec., depending upon the load. While the theoretically prescribed Nyquist sampling rate is merely double the fastest frequency of the system to be controlled, practical wisdom in the control community has long dictated the choice of an order of magnitude higher [4]: we may thus take $1kHz$ as the desired sampling and control update frequency target. The coordination of multiply coupled degrees of freedom exacts a surprisingly high computational cost. For example, a typical nonlinear control law for tracking reference trajectories with a six degree of freedom industrial robot manipulator requires roughly $10^8$ floating point operations every sampling interval [5]. This algorithm, then, would require $1MFlop$ of raw computational capability for real-time implementation. Assuming that the controller must also sample joint positions and velocities and deliver torque commands to all six degrees of freedom, the algorithm assumes that 18 parallel I/O operations are performed at the rate of $1kHz$ as well.

In fact, the computational requirements of robotics are much more stringent than the pre-
vious paragraph suggests if we are interested in autonomous robots and other machines which possess some greater independence from human operators. For example, robot navigation feedback laws for environments cluttered with obstacles [6] involve real-time computation which grows exponentially with the number of degrees of freedom. Intuitively, it is clear that the closer we come to an anthropomorphic model of the machine, the more overwhelming the real-time computational load will become.

More relevant, perhaps, to the short term opportunities of the industrial world are recent developments in smart sensor and actuator technology. For example, a new class of variable reluctance motors promises relatively large torque/mass ratios with the full torque available at low speeds, suggesting great improvements in direct-drive positioning applications through the elimination of transmissions and linkages. However, satisfactory performance from these motors requires a great deal of attention to their commutation and current drive strategies, necessitating the dedication of a single processor to the production of torque itself in a single motor. Similar computational requirements in smart sensor applications will quickly convince the forward minded engineer that real-time motion control applications require not merely copious computational capacity but distributed processing and multiple channel I/O.

When faced with this problem over two years ago at the Yale Robotics Laboratory, the authors made a thorough review of the available technology. With an à priori commitment to parallel processing, and even present financial constraints, we sought a system consisting of relatively simple and affordable nodes, deployed in a reconfigurable network arrangement with high inter-node bandwidth. Since the theory regarding algorithm partitioning within a parallel processing environment is still in its infancy, and target control situations are legendary in their variability, we saw system reconfiguration flexibility as an important consideration. Furthermore, since robot sensor and actuator technology is in a state of rapid flux, each node would need to be easily adaptable to specific digital and analog interfacing problems.

A most important consideration centered around the software and development environment. We wanted to experiment with motion control algorithms and emphatically did not want to write real-time operating system kernels or develop our own parallel/concurrent code distribution tools. The system chosen would need a suitable high level language that supports parallel processing, and has capabilities for writing, distributing and debugging code. Most critically, the hardware associated with the development and target system system needed to be affordable.

1.2 Technology Review

The options available in the marketplace were numerous when we began our search for solutions two years ago: they are almost bewildering now. In this section we review the thinking that led to our developing the Inmos T800 based Yale XP/DCS boardset. Since we feel that this design remains a leader today with respect to price/performance criteria, we will attempt to contrast its capabilities with some of the very recent commercial products which were not available at the time of the original design.
Truly parallel systems employing a distributed architecture amenable to multiple instruction flow (e.g., Floating Point Systems, Intel Hypercube, Ncube, etc.) have been commercially available for some time. While these systems typically exhibit great processing power they often have limited internode communication bandwidth or inadequate real-time I/O capability. Cost represents a major consideration in our project as well as in industrial robot control systems: investments in the range of up to several hundred thousands dollars disqualify such commercially available distributed networks for our purposes.

On the other hand, Array processors, offered by companies like Systolic Systems, Marinco Computer Products or Mercury Computer Systems, represent an alternative means of boosting computing power for conventional mainframes or even personal computers. Even two years ago, they commonly featured rates of between one and five million floating point operations per second, substantial user libraries and cost around $5000. These array processors work well with certain classes of numerical problems (e.g. image processing, matrix manipulation) but the computational problems of robotics, as we have described them above, derive from their varied nature (i.e. I/O handling, interrupt latency considerations, floating point power, inter-processor bandwidth, etc.) and as such would benefit only marginally from such an architecture. The systolic array processors are best left to less "fuzzy" computational niches.

The inherent caveats of purchased systems (i.e. cost, manufacturer dependence, inflexibility, "non-transparency", hardware overhead) may be eliminated by recourse to customized design. An increasing number of such "homemade" distributed, real time controllers, were indeed being built two years ago, and continue to appear today. Generally, these systems evolve around off the shelf commercial microprocessors, e.g. one of TI's TMS320, Motorola's 68000 or National Semiconductor's 32000 family. For example, the TMS32010 is a RISC (Reduced Instruction Set Computer) signal processor using 16 bit instruction/data word operating at 5 MIPS (Million Instructions per Second). The outstanding features are the 200 ns (one instruction cycle) 16x16 Bit multiply, the 288 Bytes of On-Chip-RAM and capacity to access a total of 8 KBytes external memory at full speed, i.e. 200 ns. The MC68020 32 Bit microprocessor maintains a sustained rate of 2 to 3 MIPS and burst rates of more than 8 MIPS. It features a 256 Byte instruction cache that allows simultaneous data and instruction access and execution. More recently, a number of commercially developed single board computers based on these processors are becoming available as well, and we may consider their performance rather specifically.

Motion controllers designed as functional elements on one of the many shared bus arrangements are currently popular. The VME bus commands wide appeal in the industrial market place and its vendor support base is still expanding. The computational speed of the latest VME RISC based CPU boards is certainly adequate for high performance motion control. As an example consider: the offerings from Ironics Inc., Ithaca, NY) a VME board manufacturer specializing in multiprocessing applications. Their latest high-performance board, the IV-9001, is designed around AMD's RISC 29000 microprocessor. With a throughput of 17MIPS at 25MHz and 6MFLOPS with the 29027 coprocessor, the board has enough processing power to handle the task. The system memory held on a daughter board can have 2,8 or 16 Mbytes of static-column DRAM. For I/O applications they plan a similar daughter board with providing 100 Mbytes/sec throughput. Custom I/O cards are also possible. The IV-9001 base board costs $7995. To that
must be added memory at $1995 for the 2MB board. The price for the I/O daughter board has yet to be announced. Our particular application would require the three board set. The development system consists of a 68020 based Unix workstation coupled to a VME chassis via serial link. Code is written, compiled and then downloaded to the target boards in the development chassis. Monitors, and debuggers exist to aid in the process. The development system pricing with software starts at: $16,000.

Other VME boards using the Motorola 68020 and 68030 CISC parts are somewhat less expensive but offer less performance as well. Heurikon Corp. (Madison, WI) will provide a 20MHz 68030 VME cpu card with 4Mb DRAM and a 68882 coprocessor for about $5600. With the Motorola parts, the floating point performance becomes a question for our application. At about .3 MFLOPS the 68030/68882 won't provide the computational power we require and multiple processors would be needed. Although the VME bus has a bandwidth specification of 40 Mbytes/sec, a 680x0 based multiprocessor system would probably average no greater than about 3 - 5 Mbytes/sec of available inter-processor communications bandwidth. Difficulties arise concerning bus latency and the partitioning of code in such a system.

Similar considerations apply to the Multibus. As part of the IEEE 1296 specification for the Multibus II, the concept of message passing seeks to accomodate multiple microprocessor systems. A single chip termed the Message Passing Coprocessor was designed by Intel to enable any microprocessor to interface easily to the IEEE 1296 bus. Expected bus bandwidth is improved over that of the VME bus to a respectable 13 MBytes/sec in actual applications. Costs for Multibus II boards are still fairly high however. Heurikon's HK68/M220 board uses the 32 bit 68020 and has 4MB DRAM, 256k EPROM and a 68881 coprocessor and lists for $6295.00.

The high costs per node for these commercial VME and Multibus processor boards represent only one aspect of their deficiency regarding the intended application. For those problems requiring the power of a multiprocessor solution, the major limitations of these systems stem, ironically, from their fundamental strength, the shared bus architecture, or, as it has sometimes been described, the "von Neumann bottleneck." As more processors are added to a system, the 'bus' ability to service them diminishes. The point at which the number of processors becomes problematic varies with the application but problems with bus loading and latency are inevitable. Faced with these intrinsic limitations, designers have increasingly sought to minimize bus traffic as much as possible. Their solutions take the form of more autonomous "nodes" i.e. processors having more local resources e.g. dedicated memory and I/O ports and systems with more efficient bus handling e.g. the MPC chip on the Multibus II. The enhanced bus bandwidth (100MB/s) expected with the next generation VME bus, variously called Futurebus or Ruggedbus, will provide designers some relief but won't alter the more fundamental constraint limiting performance for multiprocessor systems.

The Bitbus from Intel is another possible candidate for real-time distributed control. The low cost and availability of control related processing nodes coupled with the simple interconnect scheme make this an appealing option. The two wire twisted pair RS-485 serial bus while being economical still retains the limitations mentioned above however. While tasks can run independently on the nodes of the Bitbus, inter-task communication must be arbitrated by
1 INTRODUCTION

the single Master on the network. The message passing protocol involving mailboxes requires
that the message pass from slave A to master (with handshaking) then from master to slave
B (again with handshaking). Although the bandwidth of the serial bus can be fairly high, the
overhead incurred though the message passing protocol reduces the effective node to node data
rate significantly. The power of the Bitbus is further constrained by the 12MHz 8044 used
by virtually all Bitbus vendors as the main system processor. The chip's 8-bit architecture
makes high precision floating point calculations prohibitively time consuming. The modularity,
flexibility, and affordability of this system together with expanding third party support make it
suitable, however, for a wide variety of distributed control chores and, aside from the performance
limitations mentioned, this system addresses many of our design criteria.

In summary, whichever of the conventional microprocessors is chosen as the basis of such a
customized distributed architecture, all share two big disadvantages. First, they are not designed
to be interconnected for parallel processing and thus, by themselves, do not afford inter-processor
communication. Thus one generally is forced to resort to a bus based approach — the basic
structure for almost all parallel real-time control systems built in the past. Unfortunately, the
bus communication bandwidth decreases at least linearly with the number of nodes and I/O units
which are attached in a parallel fashion. While it may be feasible to build such parallel systems
with only few nodes, expandability quickly becomes limited (depending on the communication
requirements of the specific application). Second, due to the lack of a suitable language, software
issues become more and more problematic as the hardware increasingly exploits parallelism.

In the remainder of this article we present a description of a system we feel represents a
"best fit" of technology to task. Unable to find an off-the-shelf solution to our needs we decided
to custom design the controller and based our system design on the Inmos Transputer.

1.3 Advantages of Transputer Technology

The choice of the INMOS product line represents a strategy which standardizes and places the
burden of parallelism — inter-processor communications support, software, and development
environment — around a commercial product, while customizing the computational "identity"
of particular nodes by recourse to special purpose hardware.

The Transputer is a 32-bit RISC microprocessor with fast on-chip RAM, interrupt and DMA
support, an internal architecture supporting multi-processing, and four high speed DMA serial
interprocessor communication links. A feature list of the T800 is given in Figure 1.

The latter capability represents the most important feature of this chip relative to its com-
petitors. The four links circumvent the constraints of bus based interprocessor communication
schemes both with regard to reconfigurability as well as bandwidth. The result is a topology
to which nodes are added or deleted simply by physically connecting a four wire serial cable
(and System Service connections). Through the parallel processing constructs of the associated
programming language, OCCAM, one can equally simply address the software requirements of
process concurrency. Whether multitasking on one transputer, or engaged in parallel implemen-
1.3 Advantages of Transputer Technology

tation on a network of transputers, the desired relationships between software processes and hardware processors may be specified with ease and flexibility.

The Transputer Development System (TDS), consisting of an evaluation board (B004) and supporting software and documentation, satisfies the need for a coherent prototyping environment. Using an IBM AT, or a more powerful engineering workstation (e.g. Sun, Apollo, Vax, etc.) as a host, the user can generate, debug, compile, and download code to a target node or group of nodes. The network configuration utility included in TDS minimizes the software changes attending the addition of a node: the new node’s name (i.e. PROCESSOR 5 T8) along with processes targeted to run on it are adjoined to an existing network configuration file; assignment of download link and interprocessor data transfer links completes the specification. The OCCAM compiler contained within the TDS supports the creation of processes that use “channels” for communication. These soft channels are mapped into physical links when a program is configured. Given this capability, programs intended for a particular interconnection scheme using a specified number of nodes can be simulated on variant networks, or even a single transputer, if desired.

In summary, from our point of view, the Transputer’s primary advantage over the other comparable CPU’s mentioned in the previous section is its intrinsic inter-processor communications capability. This capability is notable both with respect to hardware performance as well as the relative sophistication of the commercially provided development environment for parallel and concurrent applications. We now describe the present state of evolution of a board design for a real-time motion controller based upon this technology.
FEATURES

Integral hardware 64 bit floating point unit
ANSI-IEEE 754-1985 floating point representation
Sustained 1.5(2.25) Mflops
32 bit architecture with 15 MIPS† performance
Hardware and pin compatible with IMS T414-20
4 Kbytes on chip RAM for 120 Mbytes/sec† data rate
32 bit configurable memory interface
Directly addresses 4 Gbytes at 40 Mbytes/sec†
High performance graphics support
Sub-microsecond context switch & interrupt latency
Four 5/10/20 Mbits/sec INMOS serial links
Hardware scheduler for concurrent programs
Internal timers for real time processing
External event interrupt
Support for run-time error diagnostics
Boots from communication link or ROM
On-chip DRAM controller
Internal program continues during DMA
Optional external memory wait states
Single 5 MHz clock input
Single +5V ±10% power supply

Figure 1
2 XP/DCS Version 1.0

We set out in our board design to achieve both a general purpose computational node as well as a matched I/O board which would standardize the interface between our computational network and the physical world. The essential features of the resulting design include an INMOS T800 processor, 128 Kbyte (zero wait state) SRAM, fiber optic link interface, and modularization into a mother-daughter board set. The first feature provides reasonable computational power (1.5 Mflop) with no special purpose co-processors. The fiber optics interface turns out to be a vital step in so EMI hostile an environment as an electrical servo actuated robot. The main structural feature is the clear separation between computation and I/O on two separate boards. All I/O functions (and eventually, any coprocessors will) reside on a daughter card which plugs into a bus expansion connector on the mother board. The latter — the CPU board proper — can therefore be used alone as a floating point node or with an attached I/O card as a complete data acquisition and control node.

2.1 The XP/DCS CPU board

![Diagram of XP/DCS CPU board]

Figure 2: XP/DCS - CPU board

The high line density typical for 32 bit designs dictated the use of a four layer printed circuit board with the attendant benefits of reduced ground noise and enhanced signal integrity (the prototype was plagued by electrical and mechanical wire wrap failures). We standardized to the increasingly popular Eurocard form factor, using a board size of 100mm x 220mm, the so-called Single Extended Eurocard. The rear edge connector is pin compatible with INMOS' evaluation
cards for the ITEM system.

The CPU board supports both the T800 as well as the T414 at jumper selectable clock speeds of 15-20Mhz. A power-up reset signal provided on the board ensures that the processor is reset properly at power-up. Without this provision, random external bus accesses may occur until a program is downloaded. This is especially bad in a motion control application where I/O devices may be accessed arbitrarily and cause potential disaster.

The main system memory consists of 128 Kbytes of no wait state SRAM. This amount of memory may not seem adequate for some applications but one of the benefits of employing distributed processing is the reduction in program size running on any given node. If data logging is desired, the 2 Mbyte B004 host memory can be used. Our I/O board bus interface passes those signals necessary for additional memory, should it be required.

The Transputer's four high speed serial links are made available on the rear edge connector compatible with the INMOS ITEM boards. The link speeds are user selected. All inputs are Schottky diode clamped. For purposes of improving the impedance matching in critical cases the output lines' series resistances can be altered. To ensure signal integrity in harsh EMI, two of the links may be routed to fiber optic ports located on the board's front edge. Current hardware choices limit the fiber bandwidth to 5Mbps, a link speed which is supported by all transputer family products. The fiber optics cable lengths can extend to 17 meters.

To serve as visual software status indicators or as aids in system debugging, eight LEDs are located at the front edge of the card. The system service lines to and from the Transputer pass through the rear edge connector and have pin-outs compatible with those in the INMOS ITEM unit. All three, Error, Analyse, and Reset are visually displayed by front edge LEDs.

The I/O connector (DIN type 'B') located on the lower edge of the board, passes the 32 bit Data/Address bus with all interrupt, DMA, and bus control lines. Thus any kind of add-on board can be attached that interfaces to the transputer memory bus. However it is required that all signals be buffered after the I/O connector.

The performance of this CPU board is determined by the Transputer employed and the number of wait states for external memory cycles. As we use zero wait state memory (for CPU speeds up to 20MHz) all the performance measures given by INMOS in [7] for the transputer apply for our CPU board as well.

As mentioned above, the XP/DCS CPU board is pin compatible with all existing INMOS evaluation products using the B201-1 10 slot card cage. The I/O daughter board derives its power from the CPU board. Analog supply voltages are passed from the rear edge connector to the I/O connector on several pins. These rear edge pins are not connected in the standard INMOS ITEM chassis.
2.1 The XP/DCS CPU board

2.1.1 The XP/DCS I/O board

Figure 3: XP/DCS - I/O board

Virtually every I/O device is now available with tri state output, eliminating the need for a separate I/O latch for each unit, and permitting operation in a parallel fashion via the same I/O latch. In order to simultaneously maximize flexibility as well as ease of use, we chose a latched 32 bit bidirectional I/O bus which provides for a virtually unlimited number of I/O devices with minimal chip count. To further minimize the custom I/O effort when accommodating a specific I/O device, we provide six individually addressable sets of four latched handshaking output lines as well as a total of eight handshaking input lines. In order to prevent arbitrary latched outputs of the handshaking output lines at power up to cause disaster (for example, enabling a robot joint while the torque command is not under control), these outputs wake up in a high impedance state and can be jumpered to either polarity on the fabricated section of the I/O board. For programming and debugging convenience, all handshake output lines can be read back.

This implementation provides the user with a maximum of support for custom I/O needs. Most tristate I/O devices should be able to interface to this bus without any additional support chips at all. If desired, several devices can be accessed simultaneously: for example, two 16 bit or three 10 bit D-to-A or A-to-D converters could be accessed by attaching the different chips to different portions of the I/O bus.

We call the mode previously described the asynchronous I/O mode. Devices can be accessed independent of their speed. A complete I/O cycle with three I/O bus accesses in this default mode would take less than 1 μs, or roughly the time for one floating point operation. Recall
that a typical robotic application, as outlined in the first section, requires 18 I/O operations and 1000 flops per sampling time. Thus we can easily see, that with these timing considerations the I/O time requirements are negligible compared to the computational requirements. The typical three external memory cycles are: (1) enable device via handshaking out; (2) read from / write to device (if necessary, after delay); (3) disable device.

However, in general (for example for some ASICS, or if one needs the faster I/O cycle time — generally a gain of less than 1 µs), the board allows for direct bus interfacing to the transputer called synchronous I/O mode. This is possible by removing the bidirectional bus latches. However, no default latched mode is simultaneously possible and no further support is provided on board for this mode which is not considered the default use.
3 Performance

The first application of the XP/DCS boardset was for controlling a planar juggling apparatus (Figure 4) which was built to learn more about the underlying principles in modeling, analyzing and controlling robots that repetitively catch or throw, hop, run — or juggle [2],[1]. The physical system consists of a puck, which slides on an inclined plane and is batted successively by a simple "robot": a bar with a billiard cushion rotating in the juggling plane, driven by a DC servo motor from PMI Motion Technologies.

Figure 4: The Yale Juggler

The corresponding schematic representation of the previous picture is shown in Figure 5. It shows the concurrent processes operating on each processor.

In order to move the bar according to some puck dependent control algorithm, the puck's position and velocity in both directions on the plane have to be measured. This is done by placing an oscillator inside the puck and burying a sensing grid in the juggling plane, thus imitating a big digitizing table. The first XP/DCS node in the system functions as a smart sensor. In the tracking mode, it measures the induced grid voltages in a 10 inch by 10 inch window around the puck and computes the position from moment calculations. This information is used to both track the puck along its flight and to feed state observers to estimate the velocities and reduce the measured noise in position data. Each measurement of four puck states is communicated asynchronously via fiber optics to the motor controller. This process is performed at a rate of 1kHz, which is at least 16 times faster than possible if a standard video camera with a vision system was used to acquire the puck states.
The OCCAM code excerpt in Figure 6 shows the structural components of the sensor program.

The parameters passed to this sensor procedure are the link specification to the controller, the desired sampling time, state variable estimator poles and the juggling board angle. After library includes and declarations follows the procedure which scans one window around a given puck position. The main program consists of two parallel processes, a puck tracking program and a communication program (device driver) that communicates with the controller (Figure 6). This parallel construct serves two purposes: First, it exploits the fact that a transputer can simultaneously communicate on several links without seriously affecting the ongoing computation. Second, it decouples the timing of controller and sensor, allowing them to operate at different cycle times. If the sensor would communicate directly with the controller, without intermediate device drivers, both would run at the same cycle time — because in OCCAM, communication enforces process synchronization: communication on both sides of a transaction will only proceed upon completion, a feature very useful in many other instances. In our case the sensor performs a complete tracking cycle in a 1 ms time interval which is roughly twice the controller sampling time. These two parallel processes form the first process in a PRI PAR construct, which makes them both high priority processes. High priority processes have two main features: Their timers have 1 μs ticks as opposed to 64 μs ticks of low priority processes and they have priority over low priority processes. As opposed to a low priority process, a high priority process will not be interrupted, so in a parallel construct like ours, one has to make sure that a high priority process will not run continuously and exclude other processes from running. Processes that handle link communications should run in high priority, as external communication should be serviced promptly. The tracking program needs to run at high priority only because of microsecond timing requirements for I/O handling.

An OCCAM code excerpt that depicts the program structure of the controller processor is shown in Figure 7.

The program again consists of a parallel structure with two high priority device drivers and the low priority juggle control program. The two device drivers, one for communication between the B004 and controller for logging and one between controller and sensor, again, decouple the cycle times on the different processors and exploit communication/computation parallelism. The tasks performed in the control process are (Figure 8):

- Sampling time synchronization
  Ensures constant, fixed sampling time, here 640μs

- Motor position and velocity updates
  Reads optical encoder, derives and filters motor velocity, performs safety checks (overtravel, overspeed)

- Read the puck states from the sensor node, safety checks

- Implement the robot control algorithm which specifies a desired motor trajectory, perform high gain PD feedback motor control to achieve it
- Detect the puck status: Up, Top, Down, Impact
- Online logging to the B004 in the host PC

The advertised computational power for the 20 MHz T800 transputer is 1.5 sustained Mflops, which we have verified empirically. This number, of course, does not include external memory accesses that will be required for much of the application code. For example, a single precision operation $a := b \times c$, with variables in external memory (zero wait states) requires typically 1.3 $\mu$s. This is slower than the advertised floating point performance as it includes three external memory cycles. Longer expressions where the floating point unit can work off the evaluation stack without storing intermediate results in memory work much faster: The one line excerpt from a controller computation (Figure 9) was timed between 6.6 and 8.8 $\mu$s, depending on the data. All the variables are REAL32, except hit.dist, which is a constant (VAL REAL32). This figures agrees with the timings derived from INMOS literature [3],[7]: The computation involves six multiplications (typ. 550 ns/worst case 900), two divisions (800/1400) and three additions (350/450) as well as (max.) 13 external memory accesses (150) which sums up to 7.9/11.5 $\mu$s typ./worst case.

Of concern for real time control applications are external the I/O speed and interrupt latency. Our OCCAM test program for both is shown in Figure 10. I/O accesses are essentially external memory cycles. The timing using the PORT construct for one I/O operation was measured as 250 $\mu$s as depicted in Figure 11. The typical/worst case interrupt latency is a 19(78) cycles (assuming use of or-chip RAM) which for a 20MHz T800 translates into 0.95 (3.9) $\mu$s depending on the length of the presently executed instruction. For our test code we measured 500ns. Figure 12 shows the event request signal in the upper and the event acknowledge in the lower trace.
Figure 5: Processes/Processors

```
{{f SENSOR
  PROTOCOL format IS INT; (8)REAL32 :
  PROC SENSOR (CHAN OF format login, logout, 
                     VAL REAL32 delta.t, x.db1.pole, y.db1.pole, board.angle.deg) 
    #USE mblib
    #USE mathvals
    #USE snglmath
    ... declarations
    ... SC xyscan ( REAL32 x.center, y.center, BOOL search )
  PRI PAR
  PAR
    ... track puck
    ... device driver to controller
  SEQ
    SKIP
:
}}
```

Figure 6: The Sensor Occam Code
{{{F CONTROLLER
PROC CONTROLLER (CHAN OF ANY fromsens, tosens,
VAL BOOL logging,
VAL INT rows, interleaves, misc,
VAL REAL32 smol.time, board.angle.deg, alpha, x.d, y.dot.d,
set.k.1, set.k.2, set.k.3,
inertia, KP, KV, grav.torque, offset.torque)

... includes and declarations

PRI PAR
PAR
... sensor -- controller device driver
device driver
do not control

: : :

Figure 7: The Controller Code

{{: juggler control

... initializations

WHILE NOT abort
SEQ
  time.update()
  theta.omega.update()
  read.puck.states()
  robot.continuous()
  puck.phases()
  online.log()

  shut.down.motor()
  terminate.logging()
  flash (abort.mask)

}}}

Figure 8: The Juggle Control Code
result := ( ((x*y.dot)-(x.dot*y)) + 
( (sign.x*(hit.dist*((x*x.dot)+(y*y.dot)))) /div1 ) /div2

Figure 9: Sample Controller Computation

PROC test.io.event ()
    #USE mlib

    BYTE event.in:
    CHAN OF BYTE event:
    PLACE event AT #0:

    PORT OF INT HANDSHAKE.OUT.2:
    PLACE HANDSHAKE.OUT.2 AT #20000006:

PRI PAR
    SEQ
        WHILE TRUE
            SEQ
                event ? event.in
                delay(64)
            SEQ
        WHILE TRUE
            SEQ
                HANDSHAKE.OUT.2 ! #0 -- set and reset LSB of handshake.out port#2
                HANDSHAKE.OUT.2 ! #1 -- for timing test purposes
                HANDSHAKE.OUT.2 ! #0
                HANDSHAKE.OUT.2 ! #2 -- set event request
                delay(1) -- delay by 64 us (1 tick = 64 us)
                HANDSHAKE.OUT.2 ! #0 -- reset event request, event has been service
                delay(1) -- delay by 64 us (1 tick = 64 us)

Figure 10: I/O and Interrupt latency test code
Figure 11: IO access scope trace

Figure 12: Interrupt latency scope trace
4 Applications

The XP/DCS system was designed to be the general workhorse for real-time motion control experiments within the Yale Robotics Laboratory. Beyond the juggling system described above, the range of mechanical and electronic devices supported by this system is well illustrated by the following list:

- a four degree of freedom direct drive robot
- a camera field-rate (60 Hz) real-time vision system
- direct drive robotic actuators which require particularly intensive computation for good performance: e.g. an NSK variable reluctance motor; a Bridgestone "rubbertuator" (a novel rubber muscle-type pneumatic actuator with unusually good force/weight ratio)
- a three degree-of-freedom version of the previously described juggling robot that can juggle, catch and throw objects in space.

In this Section we review our experiences with the first application.

An advanced robot controller based on the XP/DCS is under construction for both testing new robot control algorithms and investigating issues of distributed real time control. For this task the GMF Robotics Model A-500, a four degree of freedom SCARA type arm shown in Figure 13, was chosen as the target mechanical unit.

![Figure 13: The GMF Model A-500](image)

The general structure of a robot/controller system takes the form shown in Figure 14. The robot is the actual mechanical unit shown in Figure 13, and the control system is to be implemented as a network of XP/DCS computers. The robot inputs are the terminals of the electric motors at each joint. Its outputs are sensors which measure the position and, perhaps, velocity of the individual robot joints. The controller has a more complicated I/O structure. First, the controller must be able to read the robot state in some fashion. Second, the controller may also have a reference trajectory input through which it receives commands. The controller output is
4.1 The Servos

![Control System Diagram]

$q_d$ → Control System → $\tau$ → Robot → $q, \dot{q}$

$q, \dot{q}$

Figure 14: Closed Loop System.

in the form of currents which drive the robot motor inputs. The purpose of the control system is to manipulate the robot inputs so that the actual robot state converges to equal the reference input.

4.1 The Servos

Like virtually all currently available robot systems, the original A-500 system controller provides an integrated high level user interface which serves admirably in industrial applications, but precludes the low level servo intervention which is needed in the research laboratory. For our experiments it is necessary to be able to directly and independently specify the torque being delivered by each joint of the robot. Since the original control system does not allow this type of interface at any level, it was necessary replace the manufacturer’s control system with our own system. For each of the robot’s joints, the new interface consists of a dedicated INMOS Transputer which directly commutates (in software) the currents in the DC brushless motors at the robot joints. The system block diagram for a single joint is shown in Figure 15

The servo transputer has five primary tasks:

- Receive torque commands from a higher level controller via standard INMOS links.
- Report current state information ($\theta_i, \dot{\theta}_i, \sin \theta_i, \cos \theta_i$) to the higher level controller.
- Perform low-level commutation of currents in the three phases of the dc servo’s windings.
- System monitoring and software safety interlock (a hardware safety interlock system is, of course, also implemented.)
- Servo initialization on power up.

The servo hardware of Figure 15 consists of six major functional modules:
XP/DCS Boardset: Motherboard and I/O board. Commutation, initialization, and error handling are implemented in software.


Shaft Encoder: Provides quadrature incremental readout of motor position.

Motor: Three phase DC brushless motor.

Servo Amplifier: Three channel bipolar PWM high power motor amplifier.

Yale I/O Interface: Six channel PWM generation and current feedback loop.

A primitive protocol has been defined for the interface between the servo transputer and the control network, thus abstracting them as "perfect actuators" which report their state in floating point units of radians, and receive torque commands in floating point units of Newton-meters. The simple servo I/O structure gives the designer a clean interface to experiment with high level control algorithm design and implement arbitrary network topologies using standard INMOS compatible nodes. We have found this approach to offer a powerful and flexible environment superior to bus-based multiprocessing environments. The speed and simplicity of processor interconnection has proven indispensable in the ease of rapid prototyping and testing of network concepts.

A prototype servo system realizing Figure 15 has been constructed and tested, and successfully commutates the dc brushless motor at a frequency of 10Khz. Implementation for the remaining joints is currently underway.

The servo transputers provide a clean interface to each actuator, thus freeing the designer of the control network from low level operational requirements of the the particular motors used. Thus the architecture of the control network will be dictated solely by the structure of the
4.2 A Sample Nine Node Control Network

The details of a particular control network topology for realizing the "computed torque" algorithm are depicted in Figure 17, and experiments have been reported in [8].

Our present implementation has each servo processor communicating directly to two processors: a dedicated computation processor and a dedicated communication server processor. The server processors communicate in a ring topology. Communicating directly with the \( i \)'th low level servo is the \( i \)'th computation node which computes the subexpression of the control algorithm associated with the \( i \)'th joint. The local computation consists of the entire feedback gain calculation, and the \( i \)'th rows of both the \( M(q) \) and \( B(q, \dot{q}) \) matrices, followed by the appropriate multiplication and summation. The local computation receives the \( i \)'th state information directly from the \( i \)'th servo, and the \( j, j \neq i \)'th state information as well as the reference information from a dedicated server node. This local computation is currently executed on a single T-800, but one might imagine further granularization of this process. Each low level servo node also reports its state directly to its local server node which forwards this data to the remaining servers. An additional command server receives reference trajectory commands from the host and forwards this data to the servers.
Figure 16: Implementation Gross Block Diagram.
4.2 A Sample Nine Node Control Network

Figure 17: The Control Network Topology.
References


