INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

AN EMBEDDED TEMPORAL EXPERT

FOR CONTROL OF A TANDEM ACCELERATOR

By

BRADFORD J. RODRIGUEZ, B.S., M.S.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree

Doctor of Philosophy

McMaster University

EMBEDDED TEMPORAL EXPERT

FOR CONTROL OF A TANDEM ACCELERATOR

DOCTOR OF PHILOSOPHY (1997)
(Electrical Engineering)

McMaster University
Hamilton, Ontario

TITLE: An Embedded Temporal Expert for Control of a Tandem Accelerator.

AUTHOR: Bradford J. Rodriguez, B.S., M.S. (Bradley University)

SUPERVISOR: Dr. W. F. S. Poehlman, Department of Computer Science and Systems

NUMBER OF PAGES: x, 136

ABSTRACT

Many process control applications are best solved by heuristic, or rule-based, control. Unfortunately, conventional expert systems are generally large and slow, centralized on workstation computers, and incapable of continuous operation. Furthermore, few expert systems are able to process time-varying data, or to reason about temporal relationships. Thus they are ill-suited to process-control, which is inherently a continuous and temporal problem, and which increasingly relies upon distributed networks of small embedded processors.

A new expert system has been developed to overcome these limitations. This system achieves extremely high inferencing performance on "low end" microcontrollers, and requires very little memory. A new "cooperative/advisory" model of distributed problem solving allows networks of processors to cooperate on a problem, while remaining able to work independently on distinct subproblems. Knowledge, in the form of facts or rules, may freely migrate around the network. The system incorporates a new algebra for time-valued data, and a formal temporal logic for reasoning about this data.

The capabilities of this system were demonstrated by automating, for the first time, the terminal charging subsystem of a model FN Tandem particle accelerator: a problem which is resistant to an analytic solution. Using the expert system, cooperating 68HC16 microprocessors have successfully operated the accelerator, performing as well as, or better than, an experienced human operator. During the course of these experiments, new techniques for technology insertion were devised, and a new local-area network for microcontrollers was invented.

ACKNOWLEDGMENTS

First and foremost, I thank my wife, Wendy, without whom this work would not have been possible. For encouraging me to enter the graduate program, for moral support through crisis after crisis, for paying the bills, for making it possible for me to devote my full attention to the project, for reviewing my manuscripts, for unflagging support and devotion throughout the whole program: "thanks" is too mild a word.

Second, I thank my supervisor, Dr. "Skip" Poehlman, who navigated me through the tortuous maze of a doctoral program, who facilitated my wishes and fought for my project, and whose patience, advice, and support sustained me through many a rough time.

I also thank my committee members, Dr. Dan McCrackin and Dr. Bill Garland, for their enlightened guidance, and for their frequent suggestions and helpful ideas, and particularly for telling me to "down tools" and start writing.

This work depended on much patient help and cooperation from the staff of the McMaster Tandem Accelerator Laboratory. I am particularly grateful to Jim Stark, Gary Mulligan, and Winston Williams, for countless days of their time and assistance.

Special thanks go to my two "unofficial" advisors, Dr. Nicholas Solntseff and Dr. Glen B. Haydon, who kept me "on track" and kept pushing me to finish the project!

Some of the hardware used in this project was funded by the Natural Science and Engineering Research Council of Canada.

Finally, I must express my gratitude to my close friend and fellow student, Robert Duffield. We started this journey together, and he has sustained me through some difficult times.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1. INTRODUCTION

1.1 Introduction

Computers are daily finding wider and wider use in the control of physical processes -- from automobile engines to automobile factories. Often the computer merely embodies in inexpensive silicon a "classic" control methodology, such as linear feedback or "ladder logic." But as more complex machines and processes are subjected to computer control, these conventional "algorithmic" techniques become intractable -- or may be totally unable to manage the control problem[1]. Thus there is increasing interest in adopting the "heuristic" techniques of artificial intelligence for process control. Of particular interest are the inference-driven techniques -- such as expert systems and fuzzy logic[2] -- because of their more predictable behavior.

Unfortunately, artificial intelligence was not invented in a process control environment. The state of the art in inference engines is a poor fit to the state of the art in process control.

Commercial expert systems (e.g. Nexpert, Kappa PC) typically run on large PCs or workstations.[3] Moreover, they typically run on a *single* PC or workstation, assuming that all "reasoning" shall be performed by one central unit. Also implicit in their design is the assumption of timelessness -- that all data shall be presented in an instant, and a conclusion drawn from (and for) that snapshot.

Process control, on the other hand, is increasingly the province of small embedded microprocessors. Large processes (such as an automobile assembly plant) no longer are automated with a single central mainframe computer; instead, a number of small computers are distributed throughout the plant and connected via a local-area network (Jones, 1988). And only the simplest of process control

---

1. (Zhang and Grant, 1990) cite the inverted pendulum as an example of a problem with a simple heuristic solution, but a forbiddingly complex analytical solution.

2. Throughout this paper, fuzzy logic systems shall be considered a subset of expert systems.

3. Some simple fuzzy logic engines *have* been developed for microcontrollers, e.g. (Motorola, 1992; Sibigtroth, 1991).

systems can function without some knowledge of time and change, even if only the first derivative of a feedback signal.

A process control expert system should reside on the distributed hardware used for such applications. For fastest response, decisions should be made as near to the point of control as is feasible, that is, in the "local" controllers. Centralizing all decisions limits the inferencing power to that which can be achieved by a single computer. Thus it is preferable to distribute the complex reasoning tasks over the network of computers. These systems commonly employ small microprocessors and local-area networks, so the expert system can expect only limited resources, and should use network communications to share information. Finally, the process control expert should be "aware" of time, and allow temporal relationships (such as the change of a data value) to be stated in its logical language.

1.2 High Performance Inferencing

Expert systems were originally developed on mainframe computers in languages such as LISP, a legacy which lingers. Most modern systems still require the resources of a large computer. Many are still written in LISP or similar languages, with their requirement for large "heap" memory and frequent garbage collection. This results in nondeterministic performance, which is unacceptable in a real-time control system.

Some systems (e.g. CLIPS) attempt to circumvent the problem by compiling the expert system to an efficient intermediate language, such as C. Fuzzy logic systems for microcontrollers (Sibigtroth, 1991) may be written entirely in assembly language. In either case, interactive and incremental development -- a major attraction of LISP -- is lost.

1.3 Distributed Inferencing

Contemporary expert systems typically assume either a single processor, or a small number of processors with a shared, central, "blackboard" memory where information can be posted and read. This "logical star" topology, illustrated by Figure 1-1a, is reminiscent of the multicomputer architectures of the 1960s (Hwang and Briggs 1984).

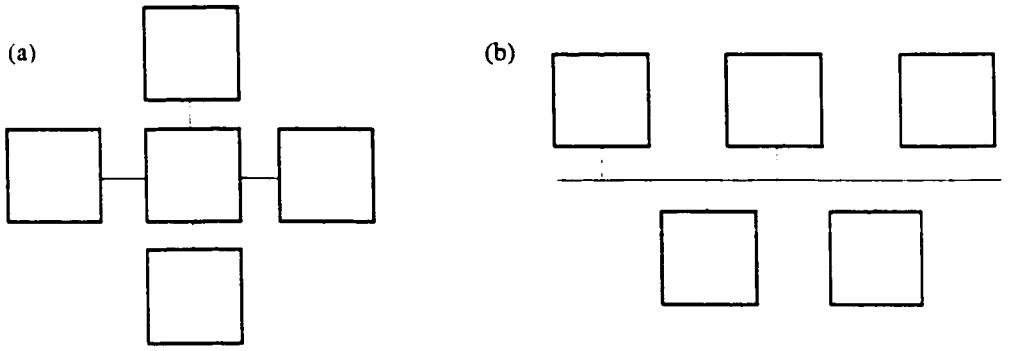


Figure 1-1. Multiprocessor Topologies. (a) star, (b) bus.

The trend in multicomputer systems is toward distributed processing. Even embedded control systems are increasingly using networks of small processors, rather than a cluster of large computers. This "logical bus" topology of a network, illustrated in Figure 1-1b, is a more suitable model for a control-oriented expert system for two reasons. First, it is a closer match to the physical networks commonly used in process control systems. More significantly, a bus topology avoids the processing limitations imposed by a central "bottleneck."

1.4 Temporal Logic

Unlike many subjects to which expert systems have been applied, process control is not a static problem. *Time* is frequently a consideration when making a control decision. Examples include:

- ensuring that events occur in the proper sequence
- monitoring inputs for a change in status
- keeping input data "current"
- computing rate of change of control variables
- recognizing and analyzing trends
- maintaining a history of input or output values

In short, information must be treated as having a temporal aspect. Data may have a time value, and a history log may be needed for some variables. Furthermore, the logical language of the knowledge base must be able to clearly express temporal relationships.

1.5 Control Requirements of the Tandem Accelerator

An example of a process control problem exhibiting all these requirements is the McMaster University Tandem particle accelerator. Operation of particle accelerators is largely heuristic (Poehlman, Garland, and Stark, 1991), with many temporal aspects. This demands a high-performance inference engine, running on a set of embedded control processors.

The High Voltage Engineering Corporation model FN Tandem Accelerator, as installed at McMaster University, is shown schematically in Figure 1-2. The purpose of this device is to accelerate

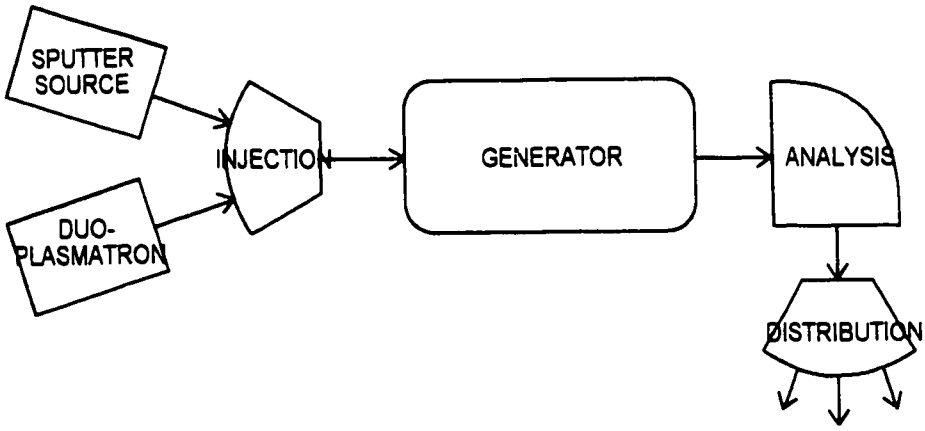


Figure 1-2. The Model FN Tandem Accelerator

various atomic nuclei to energies of several tens of MeV. Either of two ion sources (Duoplasmatron or Sputter Source) produces a beam of negative ions of the desired element. This beam is injected into the Generator tank. At the center of the tank is the high voltage terminal, which is charged positively to a potential of several megavolts by two Van de Graaff belts[4]. The negative ions accelerate towards and enter the positive terminal. Inside the terminal, the ions are stripped of some electrons as they pass through a thin foil or gas. The now-positive ions accelerate away from the terminal and out of the tank. The analyzing magnet bends the path of the ions having the desired energy 90 degrees: ions of lower or

4. In the McMaster University accelerator, chains are used rather than belts. The operating principle is the same.

higher energy are bent more or less and are blocked by a slit. This "tandem" arrangement doubles the effective acceleration, and allows both the ion sources and the beam targets to be at a low potential.

Heuristic control

In theory, the physics of this device are susceptible to an exact analytical solution. In practice, dimensions are never exact, alignment is never perfect, and currents and voltages are never precisely known. Many stray influences can perturb the particle beam. Thus the accelerator poses an intractable control problem. While some subsystems of the Tandem Accelerator have been automated (see Chapter 2), overall control of the accelerator is almost completely heuristic, and is usually performed by a human operator.

Temporal

Time is a factor in most aspects of accelerator control. For instance, it is not permissible to allow the generator terminal to charge too quickly; this requires knowledge of short-term rate of change. The stripper foil in the generator can wear out and require replacement; this is deduced from long-term trends in the measured beam current.

Embedded Multicomputer

Even a small accelerator such as the FN has hundreds of inputs and outputs.[5] It is expensive and impractical to wire all of these to a single computer, and the input/output processing alone would heavily load that computer. It is preferable to divide this burden among several processors. Ideally, these processors and their I/O wiring would be located near the accelerator subsystems, concentrating the data and reducing the number of cables to the control room. These considerations, and the hostile environment[6], suggest the use of embedded microcontrollers communicating via a local-area network.

1.6 Research Objectives

The task of the distributed expert system shall be to control a major subsystem of the FN

5. Just from Injection through Analysis, there are over 60 measurable quantities.

6. Among other problems: lack of air conditioning, unregulated power, electrical noise created by machinery, and huge electrical transients caused by megavolt sparks in the generator tank.

accelerator. As a test problem in process control, the accelerator is advantageous because most response times are measured in seconds, rather than milliseconds. Also, few "small" accelerators have ever been automated; still fewer with inference-driven control software (Lind, Poehlman, and Stark 1993).

This research project will make four contributions to the field:

a. High performance knowledge processing on simple CPUs. An expert system shell and inference engine, suitable for use on small microcontrollers, shall be developed. This software must use only the few kilobytes of memory available on these controllers. Heap storage, which is extravagant in memory use and which introduces nondeterministic delays, should be avoided. Speeds of hundreds of Inferences per Second, using a typical single-chip microcomputer, are desired.

b. Distributed reasoning. A method shall be devised to allow a set of microprocessors, each with an independent inference engine, to cooperate on the solution of a problem. This method shall be suitable for microprocessors communicating via a local-area network.

c. Integration of time into knowledge-based systems. The expert system shall be able to draw conclusions from time-constrained data. It shall also be able to represent, in its knowledge base, logical statements about temporal conditions.

d. Real-time control of a dynamic physical system. To demonstrate the efficacy of the distributed temporal inference engine, it shall operate a real-time process that has never before been automated: the FN Tandem Accelerator.

CHAPTER 2. LITERATURE REVIEW

2.1 Real-Time Expert Systems

Early work in expert systems was done with the preferred platform of the artificial intelligence community: mainframe computers, and the LISP language. This legacy can be seen today in such widely-used expert systems as OPS5, KEE, Kappa, Nexpert, and Personal Consultant Plus, to name just a few. The mainframe computer has been replaced by a LISP machine, a workstation, or a high-end PC, but massive processor power and slow performance remain the norm. However, in the last decade, there has been increasing interest in real-time[1] expert systems for embedded processors (Laffey et. al. 1988).

One of the first such systems was Hexscon, whose design goals included

> "(1) a capacity of 5000 rules in a microcomputer system with 512K memory, (2) a response time of 10 ms to 100 ms, (3) the ability to handle many objects (about 1000), and (4) the ability to continue functioning despite a lot of uncertainty" (Wright et. al. 1986).

Hexscon uses "progressive reasoning" to produce a quick approximate solution with conventional logic, followed by up to three successive stages of knowledge-based refinement as time allows. IF-THEN rules are compiled into a more efficient form for evaluation; procedural code can be referenced by the knowledge base. Both forward- and backward-chaining are employed by the inference engine. Uncertainty is supported by "belief" and "confidence" values, which are also used to resolve conflicting chains of reasoning. A simple "frame" mechanism allows rudimentary temporal reasoning about "past," "present," and "future," although past knowledge is stored off-line (on disk). Hexscon was implemented in Pascal on 8085 and 8086 microprocessors; when configured for maximum speed on the 8086, knowledge-based response times on the order of 0.25 - 0.5 seconds were achieved.[2]

MUSE is another package intended for real-time applications on embedded hardware (Reynolds

---

1. In the context of expert systems, "real-time" has been used to mean either "fast" or "deterministic." For the purposes of this survey, no distinction between the two is necessary.

2. It is not known how many Logical Inferences per Second this represents, but about 250 rules were used.

1988). It is written in PopTalk, an object oriented variant of POP-2; the resulting object code for the PopTalk virtual machine is interpreted on the target hardware (currently, a 68000 single-board computer). MUSE implements a forward-chaining inference engine similar to OPS, using a modified Rete (Forgy 1982) pattern matcher. A backward-chaining inference engine is also supplied. Procedural code may be freely used to qualify patterns, express rule firings, and perform input and output.

REAL-OPS is a real-time implementation of the forward-chaining OPS5 production language. It is written in Forth rather than LISP, and uses an improved "synchronous" garbage collection scheme to avoid unpredictable delays, for a substantial speedup:

> REAL-OPS running on a HP236 (8 MHz MC68000 cpu) runs a seven-tower problem in 25.7 s, while the LISP-based OPS5 takes 2 min on the MicroVAX II, 60 s on a VAX 11/780, and 18.9 s on Texas Instruments' Explorer. (Dress 1986)

This amounts to approximately 10 inferences per second. REAL-OPS includes provisions for external events (interrupts), input and output, and calls to procedural code.

Another OPS5 variant, OPS83, was used in the Embedded Inference Engine (EIE). OPS83 is a forward-chaining commercial package written in C, which is "five to 10 times faster than the equivalent LISP-based production system," and allows linkage to procedural C routines (Skapura 1989). EIE applies the concepts of concurrent programming to OPS83, to create "cooperating production systems," each of which works on a different subproblem (albeit coexisting a single CPU). This partitioning of the problem, combined with the performance of OPS5, yield in EIE benchmark speeds on a 10 MHz PC/AT which are 8.76 times faster than OPS5 on a VAX.

FORPS is a forward-chaining production system that was developed for a 68000 microprocessor.[3] It also uses compiled knowledge, with the ability to call procedural code from within rules. An explicit priority mechanism resolves rule conflicts. FORPS has been benchmarked at up to 221 inferences per second, but since the inference engine operates by exhaustive testing, speed is inversely proportional to the number of compiled rules in the knowledge base: "a system with more than 700 rules would require at

---

3. Implementations also exist for the 8086.

least .5 seconds per inference" (Matheus 1986).

At the performance extreme, speeds of 1095 Logical Inferences per Second (LIPS) have been attained with FORTES, a forward-chaining system for the IBM PC (Redington 1986).

An expert system described by (Lewis 1986) is noteworthy because it pioneered the use of a linked list for all rules dependent on a given antecedent[4] (as well as a list of all rules contributing to a given consequent). This provides fast forward- and backward-chaining. This system also used a compiled rule base, with procedural code.

The Prolog language is often used to implement rule-based systems. One commercial product, PDC Prolog, allows routines in a procedural language (C, Pascal, or assembler) to be called from Prolog, and allows Prolog predicates to be called from the foreign language (Murphy 1993, Elder 1993).

The Super expert system (Morizet-Mahoudeaux 1996) addresses almost the entire "wish list" of (Laffey et. al. 1988) for a real-time expert system.[5] Its backward chaining inference engine is capable of managing "rule interruptions, focus of attention, asynchronous event or input handling, multitasking, and temporal reasoning." Super translates a knowledge base to C++ source code, which can then be compiled with procedural C or C++ code for an embedded processor. Super is integrated with a real-time multitasking operating system; inputs and interrupts are communicated as messages to the inference engine. These can start the inferencing process, or rules can be scheduled for periodic activation. Events which occur during an inferencing cycle can suspend and restart the inferencing process. This process is quite fast:

> In the worst case, an Intel 80486/25-MHz microprocessor-based computer can completely analyze a compiled networked knowledge base of fifty thousand rules in less than 0.6 seconds. (Morizet-Mahoudeaux 1996)

Super is noteworthy in that knowledge is explicitly expressed as a logic network, in the manner of (Siddall 1990); this network is also used to enforce data consistency. A limited set of temporal relations is recognized by the system: current time, duration, frequency, and temporal correlation (succession of

---

4. This is equivalent to the "dependency list" described in this thesis.
5. See Chapter 3.

events).

Time Constrained Reasoning

Hexscon (described previously) uses "progressive reasoning" to achieve a solution within a given time constraint. This technique, also called "any time" search[6] (Lesser 1990), has also been used by (Broeders et. al. 1989), to successively apply up to five rule bases of increasing sophistication to a problem. Their process control expert system also achieves several other goals of a real-time expert system, including mixing of procedural and inferential code, a compiled rule base, direct input and output, and moderately high speed (e.g. response times ranging from 55 to 430 msec).

Rather than evaluate and then discard approximate solutions, an alternative technique attempts to decide in advance the best solution strategy for a given time constraint. This reasoning about the reasoning approach (or "meta-reasoning") may elect several approximations:

> (1) approximate search strategies explore a smaller portion of the search space than would be the case during normal processing, (2) data approximations provide an abstract view of data resulting in a simpler space being searched, and (3) knowledge approximations simplify the knowledge being applied in the system so that the search space can be explored quickly. (Lesser, Pavlin, and Durfee 1988)

Coping with these approximations requires a number of new knowledge representations, e.g., "filters" to eliminate unprofitable hypotheses and strategies (Decker, Lesser, and Whitehair 1990). This system was very effective in meeting specified deadlines for the Distributed Vehicle Monitoring Testbed (DVMT).

Another system which tries to produce a single, best solution within a given time constraint is RUM. Rules are written in KEE, using "both procedural and declarative knowledge" (Bonissone and Halverson 1990). The static rule set is then translated into a directed acyclic graph for efficient execution (in the same manner as Super, described above); the translator also computes static performance metrics for the network. The target (embedded) hardware receives this rule network, forward- and backward-chaining inference engines, and a metacontroller. The metacontroller uses the performance metrics to evaluate the best solution strategy for a given time constraint. The knowledge base is also statically

---

6. So called because a valid result can be obtained at any time after the first, approximate solution.

partitioned into subsets, allowing focus of attention. Multiple tasks with different priorities and deadlines are managed through an agenda, and interrupts can supersede the executing task and update the knowledge base. Although RUM is a single-processor implementation, its authors note that the delivered software is suited to distributed execution over multiple processors.

A different approach to meeting real-time constraints is taken by (Hayes-Roth 1990). The reasoning process is divided into small, well-defined operations. An agenda manager selects a subset of operations which are candidates for execution, depending on, for example, current deadlines or the occurrence of high-priority events. A scheduler then executes the highest priority operation from the (possibly incomplete) agenda. Scheduling and execution are well-bounded; thus the problem of controlling real-time performance becomes one of devising a well-bounded agenda manager.

Fuzzy Logic

By far the most explosive growth in real-time knowledge-based systems has been in the realm of fuzzy logic (Kosko 1997). A recent survey observed:

> For the past few years, particularly in Japan, approximately 1,000 commercial and industrial fuzzy systems have been successfully developed. The most successful domain has been in fuzzy control of various physical or chemical characteristics such as temperature, electric current, flow of liquid/gas, motion of machines, etc. (Munakata and Jani 1994)

The authors point out that most, but not all, fuzzy systems are knowledge-based systems.

When fuzzy rules are used for control, they are generally applied to very simple and limited problems, such as temperature control or automatic camera focusing, which can be solved with at most a few dozen rules. Typical performance of such systems might be

> [An OMRON FP-3000] dedicated-hardware fuzzy processor takes 650 microseconds to process a system of 20 rules with five antecedents and two consequents. A similar system would take up to 8 milliseconds on a 2 MHz 68HC11 . . . (Sibigtroth 1991)

Like the aforementioned FORPS system, the software fuzzy inference engine will usually just cycle through all of the rules in its knowledge base. This implies that inferencing time is inversely proportional to the number of defined rules. Perhaps for this reason, complex fuzzy systems are still rare (Sperry

1993).

2.2 Expert Systems in Control

Most of the aforementioned real-time expert systems were intended for some real-time application, such as data analysis or process control, but their published description focused on the expert system. What follows are descriptions of control applications which have used expert systems. This is but a sampling of the very large number of real-time control applications which have been subjected to AI techniques.

(Laffey et al. 1988) tabulate 43 applications of real-time knowledge-based systems, including 14 process control applications.

Even a rudimentary expert system can outperform a human operator at simple control tasks (Zhang and Grant 1990) describe a nine-rule system for the control of an inverted pendulum,[7] and compare its performance to that of human test subjects.

Starting from a rule-based solution to the inverted pendulum, Sammut and Michie evolved a rule-based attitude controller for an orbiting satellite. The final strategy involved 45 rules (15 rules for each control axis), and required no knowledge whatever of the dynamic model of the satellite. In tests with a "black-box" simulator,

> the amount of propellant used to bring the system under control compared favorably with the propellant use of a controller developed using traditional means and having full knowledge of the differential equations used in the model. (Sammut and Michie 1991)

To study the feasibility of expert systems for process control, (Francis and Leitch 1985) devised a nonlinear test problem consisting of two coupled water tanks: "a system which, although controllable, is surprisingly difficult to predict." The Artifact expert system, with a knowledge base of 21 rules, performed comparably to a PID feedback controller.

MCM is a material composition management system for chemical manufacturing, responsible for process monitoring, situation assessment, action planning, and plan monitoring. It is based on HCVM, a

---

7 A rigid pole hinged to a cart; the pole is to be balanced upright by moving the cart.

special version of Schemer, a real-time, event-driven, object-oriented expert system. Goals and plans are managed, and events are monitored, using an architecture very similar to a blackboard. MCM executes on a Lisp machine and communicates directly with separate (non-inferential) process control computers. It has successfully diagnosed and treated manufacturing process problems, but in an environment that admittedly does not demand great speed: "Admissible response times of the process management system ... are on the order of hours." (D'Ambrosio et. al. 1987)

Shallow knowledge (heuristic reasoning) is unable to cope with unforeseen circumstances, so many plant controllers also incorporate deep knowledge (model-based reasoning). MCM, just described, is one such system. Another, to control power plants, is described by (Suzuki et. al. 1990). Its Shallow Inference Subsystem (SIS) is a straightforward rule-based system, while the Deep Inference Subsystem (DIS) uses a frame-based Operation Generator to plan corrective actions, and a fuzzy-logic qualitative simulator to predict the likely outcome:

> A qualitative simulation based on a qualitative model is useful because qualitative modeling is easier and because the qualitative reasoning process better matches to the thinking process of a skilled human operator. (Suzuki et. al. 1990)

Unlike a quantitative simulation, a qualitative simulation cannot provide data suitable for numerical plant control, but it can provide information to a heuristic controller such as the SIS.

EXPRESS is a commercial process control system using a rule-based inference engine (Rather 1993). One successful application has been the automation of a lime kiln (Anway 1994).

Expert systems using progressive reasoning have been applied to environmental control (Kim and Zeigler 1990).

2.3 Distributed Expert Systems

Blackboard Systems

Most expert system research has involved single CPUs, although recently, distributed expert systems have become of interest. The prevailing model for multi-processor experts, by far, is the "blackboard" (Dodhiawala et. al. 1989). In such a system, the blackboard is readable and writeable by all

of the individual experts. Each can post hypotheses for other experts to examine, or results for other experts to use. (Corkill 1991, Englemore, Morgan, and Nii 1988)

One such system is the Distributed Vehicle Monitoring Testbed, mentioned previously. The DVMT is an artificial test problem:

> ... the purpose of building the testbed is to evaluate alternative distributed problem-solving network designs rather than to construct an actual distributed vehicle monitoring network (Lesser and Corkill 1988)

Tests were performed with a network of nodes[8], each running the Hearsay-II system. Hearsay-II allows multiple "knowledge sources" to contribute to a problem. For DVMT, "communication knowledge sources," which exchange hypotheses and goals with other nodes, were added.

A simple three-processor system has been developed by Park for real-time applications:

> The concurrent EXPERT-5 consists of two EXPERT-5 shells running on separate M68000-based co-processors mapped into the memory structure of a third EXPERT-5 based blackboard controller running under MS-DOS on a pc. (Park 1986)

This is a master-slave hierarchy using tightly-coupled processors, with the blackboard being managed by the master. No performance figures were published for the three-processor combination, but it was noted that each 68000 processor achieved standalone speeds of 9000 LIPS using the EXPERT-5 shell.

Blondie-III uses five cooperating blackboard systems to solve a telecommunications configuration problem (van Liempd, Velthuijsen, and Florescu 1990). Each agent has a separate blackboard, but all of the blackboards are globally visible. A set of distributed nodes is simulated on a Sun-3 workstation by making each node a separate Unix process, and confining all communication to interprocess messages.

(Larner 1990) describes DOSBART, a distributed real-time control system, one of whose goals is "to provide a single blackboard abstraction for control system design that can be partitioned into constituents running on multiple processors." Knowledge is distributed: stored where it is used most, but globally available to all nodes. "Data Server" objects,[9] in addition to storing knowledge, can perform dependent functions such as input, output, timed execution, and history recording. Interrupts are

---

8. In the test system, this network was simulated on a single computer.
9. Analogous to the "fact" objects described in this dissertation.

supported, as are "Trigger Activities" which cause forward-chaining, and "Theorem Provers" which implicitly backward-chain. Implemented with the Common Lisp Object System, DOSBART requires LISP processors or high-end workstations (e.g. SPARCstations).

Non-Blackboard Systems

(Green 1987), in discussing the limitations of blackboard systems, observes that the control task (scheduling knowledge sources based on the current state of the blackboard) is a limiting factor of system performance. His Activation Framework (AF) system uses messages, rather than a blackboard, for communication between many inferencing units, called AF Objects (AFOs) [10] Messages may be sent automatically (forward chaining), on demand (backward chaining), or "suggestively" (to accumulate evidence in the absence of complete data). An "activation level" gives priority to AFOs having high-priority messages or many lower-priority messages. AF also has some ability to reason temporally (sequence and relative timing of events), and can reason with uncertain data. The distributed AF system is written in C, and is simulated on a single multitasking CPU, using interprocess messages for communication.

The Distributed AI Shell (DAIS) operates on a network of MS-DOS PCs. It dispenses with the control mechanism of blackboard systems:

> The DAIS network is a collection of AI applications interconnected by DAIS primitives. DAIS does not provide coordination (central or distributed) among agents; instead, it provides primitives for interaction. Each agent can interact with any other DAIS agent driven by individual need. (Kannan and Dodrill 1990)

Each PC can support a single DAIS "agent." Communication between agents is via connectionless network messages, although a request-response mechanism is supported under this protocol. DAIS is a frame-based system implemented in C, both facts and rules are represented as objects. DAIS applications "can manipulate the entire KB[11] distributed over a network," however.

---

10. The individual experts may internally use a blackboard architecture, however.
11. Knowledge Base.

> Object behaviors are actually functions (sets of instructions), and exchanging such machine-dependent units present very tricky problems -- including compatibility at the machine instruction level, and dynamic loading of compatible functions.

Since rules (being objects) can be modified dynamically, the usual pattern-matching optimizations (e.g., Rete) are not applicable. Performance measurements for DAIS were not published.

The Distributed AI Toolkit (DAIT) supports intercommunication between independent reasoning agents, by defining languages and protocols for data sharing. These protocols are based on the Product Data Exchange Specification (PDES), the ISO specification language Express, and a new Agent Manipulation Language (AML). DAIT knowledge sources use the CLIPS expert system, and run on Sun 3s, 4s, and VAX workstations. "DAIT employs predicates for real-time control, distributed processing, and fault tolerance " (Goldstein 1994). No applications were described.

A system intended for embedded microprocessors is described by (Hendtlass 1991). Knowledge is distributed across nodes of a network, and each node operates an expert system which can use local or global knowledge. Each node's expert system may use fuzzy knowledge, and may also employ a subordinate neural network. Rule "subsets" allow focused attention on part of the rule base for improved evaluation speed. Inter-node messages are sent as text (specifically, source code for an application-specific language), allowing a heterogeneous network of mixed CPUs.

HESCPC is a hierarchical society of experts which advise the operator of a nuclear power plant, and also advise the designer of process control hardware. (Ionescu and Trif 1988). It is implemented using an expert system shell, Reshell, which has been "parallelized" on a VAX cluster consisting of one VAX 11/780 and eight MicroVAX's. Reshell supports both frames and production rules, and allows declarative knowledge to be written in M-Prolog, and procedural knowledge in FORTRAN.

The production system OPS5 has been implemented on parallel processors (Acharya, Tambe, and Gupta 1992). This is an attempt to "parallelize" a single computationally-intensive program, rather than to create a society of independent experts. Nevertheless, it does illustrate one mechanism for distributing an expert system.

2.4 Temporal Reasoning

Temporal logic as a field of mathematics is not new, and several books have been devoted to the subject. Its application to computer science is more recent, and mostly devoted to specification of computer programs and systems.

(Galton 1987) cites several examples of temporal logic in Artificial Intelligence, such as the "Time Specialist" of Kahn and Gorry. However, all of the examples cited represent attempts to explore models of temporal reasoning; that is, they were research vehicles and not applications.

BLOBS is an object-oriented blackboard system which addresses some of the requirements for a temporal reasoner: the ability to reason continuously with a continuous flow of data, the ability to obsolete old data, and the ability to reason within a time limit. BLOBS also allows rudimentary reasoning *about* time, by providing a system clock and allowing actions to be scheduled for specific times. There is no explicit representation of mathematical temporal logic. BLOBS is written in POP-11, and has been applied to the real-time problem of radar analysis. (Zanconato 1988)

A more complete temporal reasoning capability has been added to LES, the Lockheed Expert System shell. This frame-based expert system stores attribute values (data) with time tags. Rule antecedents can be written to include the temporal relations ANYTIME, ALLTIMES, or JUST, using the time qualifiers AT, DURING, BEFORE, and AFTER an absolute or relative time. Adding temporal reasoning to LES caused modest increases in data storage requirements and processing time:

> . . storage time increased by about 20 percent. Retrieval time increased by a factor of eight, and total time increased by a factor of two (for monitoring that uses temporal reasoning). (Perkins and Austin 1990)

Temporal LES has been used to diagnose real-time telemetry data from the Hubble space telescope.

The MOLTKE expert system has been extended to handle a specific temporal problem, Temporally Distributed Symptoms (TDSs) in diagnosis. (Nökel and Lamberti 1991) MOLTKE's representation language allows declaration of expected temporal relationships between events. The Temporal Matcher then examines the input data (possibly in response to test stimuli generated by the

expert system) until one such pattern is satisfied, in much the same manner as the match-resolve cycle of many expert systems. This can diagnose success or failure in a time- or sequence-sensitive test. The non-temporal functions of MOLTKE remain fully usable, with no degradation of performance.

(Tétreault, Marcos, and Lapointe 1992) describe a limited temporal facility for a Qualitative Simulation (QS) system. Simulation variables can be constrained at particular times or over time intervals; testing for temporal consistency reduces the number of branches in the simulation. While not strictly an expert system, the QS algorithm is similar. For example, this system is coded in Prolog.

Hexscon, AF, and to a greater degree the expert system Super (all described above), provide limited support for temporal reasoning.

2.5 Accelerator Control Systems

General Accelerator Controls

Accelerator laboratories have long used minicomputers and microcomputers for data acquisition and analysis. In recent years, computers have taken a more active role, i.e., control. Of particular interest is the increasing use of distributed processors to perform this real-time control task.

(Castellano et. al. 1989) describe a distributed control system for a superconducting LINAC accelerator. This system is perhaps novel in that it does not use a local area network; instead, interprocessor communication is by direct memory access between VME processor crates. At the time of writing, only manual control adjustments were supported, although a "model-driven" automatic control was anticipated.

A similar distributed system for cyclotron control (Weehuizen et. al. 1992) employs PC/386 computers linked by Ethernet. This too appears to be a manually controlled system, although a reference to "setpoints" implies at least a limited use of feedback control.

Except for a few systems noted below, control systems for accelerators have used only "conventional" techniques: analog or digital feedback loops, occasionally with adaptive control.

Tandem Accelerator Controls

Most computer control systems for Tandem accelerators could better be termed "control multiplexers" or perhaps "adjustment systems," since their primary function is to allow a human operator to manipulate a multitude of adjustments from a central location. A secondary goal is rapid data acquisition. The computer itself rarely provides active control. (McKay, Gingell, and Baris 1986; Keitel and Wilson 1987; Laycak 1989; Greiner and Caswell 1992)

A example of the "state of the art" in computerized Tandem control is described by (Kumar et al 1994). This system employs an 80386-based IBM PC, interfaced to CAMAC accelerator modules, as a control multiplexer, allowing many parameters to be manually adjusted from a single control point. The computer does not automatically control the accelerator, although it does provide alarms and safety interlocks, a logging database, and computational aids for accelerator parameters

A similar control multiplexer has been used at Chalk River Laboratories for their "TASCC" Tandem-Accelerator-Superconducting-Cyclotron (Greiner and Caswell 1992). This older system was based on a PDP-11, serving about 2500 CAMAC channels. Like the previous system, the TASCC control system was strictly manual.

Chalk River Laboratories has experimented with automatic computer control of beam steering in the Tandem accelerator (Davies and Howard 1993), but this was strictly a "conventional" feedback control with no heuristic elements. Similar experiments have been performed at McMaster University's FN Tandem accelerator (Wong 1986; Wong and Poehlman 1986; Lingarkar 1988). McMaster University has also implemented computer control of ion sources (Pollock 1985; Poehlman, Pollock, and McNaught 1985; Bokhari 1993) and the analyzing magnet (Lind and Poehlman 1994).[12]

Sweden's Svedberg Laboratory reports upgrading a model EN Tandem accelerator and its ion source to use microprocessor controls (Possnert, Carlsson, and Widman 1992), but no mention is made of the control methodology, and it must be presumed that this is a conventional control system.

A recent survey article (Lutz and Marsaudon 1994) summarizes the requirements for new

12. This research concerned the analyzing magnet of a model KN Van de Graaff accelerator, but the principles are directly applicable to the model FN Tandem accelerator.

Tandem control systems (including the need for distributed processing and optical communication). The authors note that the adoption of "modern controls" in small electrostatic accelerators (such as the Tandem) lags far behind the state of the art found in large high-energy installations. Surprisingly, their vision of a future Tandem control system suggests only one use for heuristic systems: off-line "operators' assistants" for design and repair.

Heuristic Advisory Systems

Expert systems have been developed which do not attempt active process control, but instead act as advisor or assistant to a human operator (Stark and Poehlman, 1988). While this does not eliminate the need for trained operators, it allows expert knowledge to be easily replicated, without placing stringent real-time requirements on the expert system.

The Accelerator Operator's Companion (Poehlman and Stark 1989; Berg 1989; Tam 1989) is an operator's assistant for the McMaster FN Tandem accelerator. In addition to heuristics for routine operation and fault diagnosis, it provides a database (logbook) of past runs and a spreadsheet calculator. The first prototype was developed in C, and the second using Personal Consultant Plus, both on an IBM PC.

(Clearwater et. al. 1990) describe a diagnostic expert system to monitor the performance of a data acquisition trigger subsystem. For this application it was found necessary to have a "fast" inference engine, written in C, to analyze trigger events in the allotted 4 msec. A "slow" expert, written in LISP, performs detailed diagnosis of the trigger hardware. Both experts executed on a 20 MHz 80386 PC.

Advisory systems are not limited to particle accelerators; for example, (Garland et. al. 1989) describes a heuristic advisory system for nuclear power plants. This is a problem whose complexity is far greater than the Tandem accelerator, but whose real-time requirements are similar.[13] Another such system is DIAREX (Naito et. al. 1987), a specialized operator's assistant for the diagnosis of power-plant faults.

---

13 As evidenced by the fact that both use human operators.

Heuristic Control Systems

No Tandem accelerator to date has employed heuristic techniques for operations control. Indeed, heuristic controls are rare in accelerators of any description. In 1989 it was observed that

> "Expert Systems, though discussed for some years, have not yet found many significant [control] applications. So far only fairly trivial cases have been attempted." (Gurd and Crowley-Milling 1989)

and in 1994

> "In recent years there has been expectations that AI techniques would be applied widely to accelerator operations, but these expectations have not been fulfilled. While there have been a few uses of expert or knowledge-based systems reported, these have not become widespread." (Crowley-Milling and Busse 1993)

The "fairly trivial cases" of rule-based control usually involve very limited problem domains. One reported example (Perreard and Wildner 1994) employed fuzzy logic in a control loop to condition high voltage cavities, a problem similar to that of conditioning the high-voltage terminal in a Tandem accelerator. The cavity-conditioning system used ten rules written in OPS5, with fuzzification and defuzzification performed by a separate module.

A popular problem for expert system solutions is beam line steering adjustments or "tuning." This is perhaps because

> "There are characteristics common to beam lines. Most have a large number of a few types of interacting devices. Each device has several related devices, the effects of which are not always well understood. While a mathematical model of the theoretical beam line may have been used to design the beam line initially, often that model is not precise enough to predict accurately what a given change will do to the real beam." (Schultz and Brown 1990)

The TRIUMF laboratory in Vancouver created a beam line expert using first NEXPERT, and then KEE, on a VAXstation 3200 (Schultz and Brown 1990). In trials this system was not entirely successful, as it was "unable to find an acceptable solution during the allotted time." Lawrence Livermore Laboratories, for the Advanced Test Accelerator, had greater success with a custom inference engine (Lager et. al. 1990).

A somewhat more ambitious application of expert systems is the SETUP system of (Daems et al 1994), which automates the cold-start and run-time testing of components in the CERN Proton

Synchrotron. This object-oriented expert system represents the knowledge base as "goal procedures" in directed graphs, can chain forwards or backwards, and is able to backtrack to previous goals if a subgoal cannot be achieved.[14] The application is essentially a diagnostic expert system, and does not replace the existing control system. The authors do suggest possible future uses of the expert system, including automatic run-time control.

Of greatest relevance to this work is the automation of the McMaster University model KN Van de Graaff accelerator (Stark et. al. 1992; Lind et. al. 1993). This control system used an expert system operating on a PC/386, communicating with an embedded 8051 single-board computer (SBC) that performed simple I/O functions. The 8051 also incorporated a simple neural network to analyze beam status. Two commercial expert system shells were tried in the PC, but yielded inadequate performance; the final system used a custom-developed expert system in Turbo Pascal (DeMooy and Poehlman 1991, DeMooy 1991, Lind et. al. 1991, Lind and Poehlman 1992). Accelerator startup, steady-state operation, accelerator shutdown were all performed automatically by the expert controller; in addition to fault detection and diagnosis.

The prior work with the McMaster KN accelerator has strongly influenced the goals of this project. The inadequacy of commercial expert system shells for real time control suggested the need for a new inference engine. The KN control system was limited to the inferencing power of a single CPU, and the I/O processing power of a single SBC, thus the desire for a distributed system with multiple inferencing agents and multiple I/O controllers.

---

14. This backtracking, similar to that used in pattern matching, can in fact be emulated by a goal-driven system such as that implemented in this project; see for example (Rodriguez 1989).

CHAPTER 3. AN EMBEDDED TEMPORAL EXPERT SYSTEM

3.1 Objectives

This chapter describes the development of an expert system suitable for real-time process control. (Laffey et al. 1988) and (Coyle 1990) have identified several objectives for such a system.

1. High speed.
2. The use of compiled, rather than interpreted, code.
3. Continuous operation without system-introduced delays (such as garbage collection).
4. A predictable response time, or a guaranteed maximum response time.
5. Focused attention -- the ability to select one of several working contexts.
6. Integration with "conventional" (procedural) software.
7. Direct interface to sensors and actuators.
8. Use of embedded hardware, such as single-chip microcontrollers with limited resources.
9. Handling of asynchronous events (interrupts).
10. Time scheduling of events.
11. Expiration or invalidation of input data.
12. Expiration or invalidation of knowledge.
13. Maintenance of a data history.
14. Ability to reason about temporal relationships.

In addition, the trend towards the use of networks of microcontrollers suggests these additional goals for the embedded expert system:

15. Ability to share data (input data or deduced facts) with other processors.
16. Ability to share knowledge (rules) with other processors.

No expert system to date has achieved all of these goals.[1] The expert system created during this research achieves all but goals 4, 5, and 12; and the potential exists to accomplish these as well.[2]

3.2 Efficient Inference Engines

A "production system" is an expert system whose knowledge base is represented in the form of IF-THEN rules (Townsend and Feucht 1986). For instance, an expert system to classify animals may have as one of its rules

```
IF (ANIMAL HAS-HAIR)
   OR (ANIMAL GIVES-MILK)
THEN (ANIMAL IS-MAMMAL)
```

The "IF" clause (*predicate*), contains the *antecedents* of the rule. The "THEN" clause (*conclusion*) contains the *consequents* of the rule, i.e., the new facts that are known if the rule is satisfied. The expert system matches known facts to rules until some ultimate *goal* is reached (in this example, identification of the animal.)

Traditionally the matching of facts to rules has been viewed as a pattern matching problem. In LISP, the patterns may be represented as lists of symbols; in other implementations, text strings have been used. Exhaustively searching the knowledge base for a given pattern is a slow process, and is the principal factor limiting the performance of expert systems. Two different techniques have been used to accelerate this.

Much research has been devoted to accelerating the pattern-match. For example, (Sedgewick 1983) describes how pattern matching can be performed by a state machine; (Siddall 1990) suggests organizing rules as a network. The prevailing algorithm, Rete, constructs a network from the knowledge base which maps new facts directly to the affected rules (Forgy 1982).

The second technique is to represent facts not as symbolic patterns, but as Boolean variables, as

---

1. Although the recent "Super" expert system (Morizet-Mahoudeaux 1996) comes close.

2. Predictable response time might be achieved by continuous performance monitoring. Guaranteed response time (using progressive reasoning) and multiple working contexts have been demonstrated in Forth by (Broeders et. al. 1989). Forth's "vocabulary" mechanism is another possible approach to context control. The technique described here for data expiration can also be applied to knowledge (rules); but this capability was not required by the accelerator control system.

in (Matheus 1986). For example, the previous rule could be written in C language:

```
int mammal() {
    return( hashair() | givesmilk() )
}
```

and in the postfix language Forth[3]:

```
: MAMMAL   HASHAIR GIVESMILK OR ;
```

This converts the evaluation of predicates from a pattern matching problem, to the evaluation of a logical expression. Since Boolean logic is a fundamental operation on conventional CPUs[4], logical expressions can be quickly evaluated.

In this research a third and different approach is developed, offering the advantages of both of these techniques, and more. Predicates are represented as logical expressions, and a graph is constructed to efficiently test the rules. What is novel is that the edges of the graph (paths through the network) are represented as subroutine calls, and this graph is traversed by the instruction-execution mechanism of the CPU[5]. Thus the rule network is converted to a directly-executable computer program, and the CPU to an optimal matching engine.

3.3 Backward Chaining

Given a set of inputs, the matching engine (or "inference engine") traverses the rule network and attempts to reach one or more conclusions. A "backward chaining" inference engine takes each conclusion in turn, and works backwards through the network to discover the required input conditions. If all of the required inputs are true, the conclusion is true. In this mode, each rule is a subroutine that returns a truth value by evaluating a logical expression.

---

3. Other than the niceties of syntax -- such as using : and ; to enclose a function declaration in Forth -- the principal difference in these two program fragments is that the OR operator is *infix* (between the operands) in C, and *postfix* (after the operands) in Forth.

4. Processors do exist which are designed for the LISP language; these are significantly faster at pattern matching.

5. A similar technique has been described for combinator graph reduction (Koopman 1990), but this is its first application to expert systems.

Code Generation

The postfix Forth syntax has been chosen for its ease of compilation (Kogge 1982). For example, the previous Forth fragment can be translated directly to the machine code sequence:

```
CALL HASHAIR
CALL GIVESMILK
CALL OR
RET
```

This code can be compressed 33 to 50 percent[6] through the use of threaded code (Bell 1973; Dewar 1975), which dispenses with the CALL instructions, and compiles only the addresses of the subroutines, thus:[7]

```
CALL INTERPRETER
DW HASHAIR
DW GIVESMILK
DW OR
DW EXIT
```

Since the interpreter need only execute a list of addresses, its speed penalty is tolerable (typically on the order of 100%, i.e., half the speed of the equivalent machine code). This penalty is more than offset by the advantage of portable code representation.[8] A further refinement is to use "tokenized" references to object-oriented "methods."

Applying a conventional language translator to the logical expressions of the rules will yield a subroutine network which implements a depth-first, backward-chaining search of the knowledge base. Calling any rule will cause its subordinate rules to be called, recursively, until supporting facts are found.

Inputs and Outputs

Inputs are simply subroutines which return a truth value from the external world (by reading a sensor or asking an operator, for example). These subroutines may be used freely as terms in the logical expressions of rules. The subroutine of a rule may also perform some output action, such as driving an actuator, depending upon the outcome of its logical expression.

---

6. 50% on the 68HC16 processor; 33% on the 8086 processor.

7. To be specific, this example is *direct-threaded* code. For the differences between direct-threaded and indirect-threaded code, refer to (Bell 1973; Dewar 1975; Kogge 1982). On most CPUs, direct-threaded code is faster.

8. For example, there are no machine instructions in the address list.

Procedural Interface

Similarly, *any* procedural code may be called from within a rule. For example, a subroutine could be called which performs numerical computation, or which accesses a database, e.g. to determine a desired goal. Indeed, numeric and logical expressions may be freely mixed within a rule. Procedural subroutines may also be called as output functions.

Since each rule is itself an executable subroutine, it is possible to call individual rules from "outside" the expert system, in order to determine their current status -- even partially activating the inference engine if needed. This provides much finer control than other expert systems, which at best may allow starting the entire inference engine, returning only when a conclusion is reached and the engine halted. Selectively accessing rules and facts, while the inference engine continues to run, is invaluable for process control.

Dynamic Memory

Pattern-matching expert systems, such as those written in LISP, make heavy use of dynamically allocated "heap" memory, as patterns (lists) are constantly created and destroyed during the inferencing process. This can lead to unpredictable delays for garbage collection or defragmenting. On small systems, the inferencing process may halt due to lack of memory. In contrast, this inference engine requires only a modestly-sized subroutine stack, which is always reclaimed perfectly after a rule is evaluated. This allows the inference engine to run continuously, with no system-introduced delays, on small microcontrollers with only a few kilobytes of RAM.

Pruning

Evaluation of a rule network can be accelerated by "pruning" irrelevant branches of the logic tree. For example, it is known that the animal has hair, it must be a mammal -- there is no need to ask (or sense) whether it gives milk. When Boolean expressions are compiled, this pruning may be achieved by the use of "lazy evaluation" logic operators. These are operators which terminate evaluation of an expression as soon as a result is known. In C, the operator || is the "lazy" OR:

```
int mammal() {
    return( hashair() || givesmilk() )     /* note the change */
}
```

and in Forth, similar operators have been devised (Bartel 1990; Rodriguez 1990):

```
: MAMMAL   HASHAIR || GIVESMILK ;     ( || is an infix operator )
```

Rule Memory

The backwards traversal of the network may follow convergent paths, causing an intermediate rule to be evaluated more than once. These superfluous evaluations may be eliminated by storing the result of the logic expression the first time the subroutine is executed (the *evaluate* phase), and returning this stored result on subsequent calls (the *invoke* phase). This requires an auxiliary flag to indicate whether the rule is "known" or "unknown" (not yet evaluated).

Redefinition of Rules

Since this technique uses the language compiler to translate the knowledge base to executable code, there is no need for a separate "rule compiler." This implies that the knowledge base is fixed at compile time, a reasonable restriction for many applications. A further advantage of the Forth language is that its integral compiler allows new code to be added at run time. New rules may thus be added, or, if the subroutines are vectored through pointer variables[9], old rules may be modified.

Uncertainty

It has been assumed that each rule returns a Boolean (true or false) result. Rules may as easily return a numeric result representing a range of certainty from false to true. The logic operators AND, OR, and NOT can be redefined to support strict probability (Siddall 1990), MYCIN uncertainty (Townsend and Feucht 1986), or other algorithms.

Fuzzy Logic

A numeric result may as easily represent a degree of membership in a fuzzy set (Mercaldo 1992; Kosko 1997). Fuzzy logic may be represented with the model just described if two conditions are met.

---

9. Most Forth compilers support this through the defining word DEFER.

First, each membership function is made a separate rule. Thus, instead of a single rule TEMPERATURE which has fuzzy "classes" VERY-HOT, HOT, COLD, and VERY-COLD, four rules must be defined: TEMPERATURE-VERY-HOT, TEMPERATURE-HOT, etc. Second, a new logic operator "defuzzify" must be created that combines the outputs of these rules, e.g. by computing a centroid.

3.4 Forward Chaining

For process control applications a "forward chaining" inference engine is more suitable. Given a set of inputs, this engine attempts to establish the truth or falsehood of intermediate conclusions, working forward through the tree until the conclusions have all been reached.

Many forward chaining engines (Johnson and Bonissone 1983, Matheus 1986, Sanderson and Shackleford 1986) operate by repeatedly re-evaluating the entire rule base. A significant acceleration can be achieved by maintaining, for each node in the graph, a list of other nodes which are immediately dependent on its outcome (Lewis 1986). When a node changes, only its immediate dependents need be revisited. Applied recursively, this yields a "depth first" forward traversal.

This research further accelerates the matching process by representing the forward-chain links as subroutine calls. Thus the CPU becomes the inference engine, in the same manner as the backward chaining technique just described. In this mode, the function of each subroutine is to re-evaluate the logic expression of the rule, store the result, and then call each of the dependent subroutines in turn (the forward links). It is convenient to separate this action into an *evaluate* phase[10] and a *propagate* phase.

Object Orientation

If both backward and forward chaining are to be supported, each rule requires two subroutines, or a subroutine with two entry points (Rosen, 1982). Object-oriented languages facilitate this by allowing multiple "methods" for a given object. A rule can be represented as an object, for which four distinct methods are useful:

---

10. If enough information about the logic expression is known, it may not be necessary to recompute it -- for example, if it is known to be an AND conjunction, and an input has just gone false. However, it is simpler, usually faster, and always more general to reevaluate the entire expression.

invoke - return the current truth value of this rule.

evaluate - recompute the logic expression of this rule.

assert - store the new truth value of this rule.

propagate - visit and re-evaluate the dependents of this rule.

Ordinarily these steps will occur in this order, as shown in Figure 3-1.

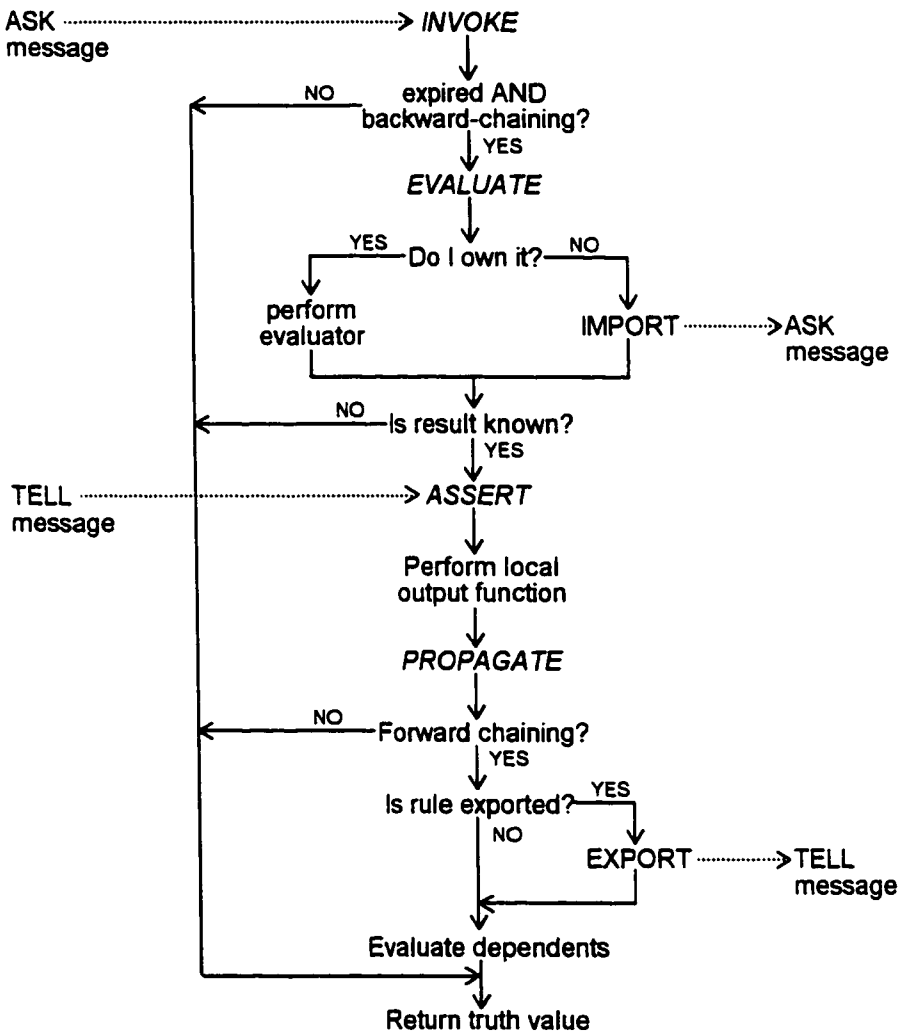


Figure 3-1. Flowchart of the Inference Engine

The object-oriented feature of "polymorphism" allows these four methods to be defined differently for different classes of rules.

Asynchronous Events

If forward-chaining is enabled, a chain of inferences is initiated whenever a new truth value is *asserted* for a rule. This provides a means for the system to respond to asynchronous events. For example, a new data value arriving from the external world (the system being controlled) can be asserted as a new value; then all rules dependent upon that value will be re-evaluated. Since *assert* can be used as a procedural subroutine, it is even possible to initiate inferencing in response to a processor interrupt.

Time Scheduling

Most real-time operating systems have the ability to schedule a routine for execution at a given time.[11] This may be used with the *assert* operation to schedule inferencing events. Events can also be linked to a real-time clock interrupt; for example, the accelerator control system (Chapter 4) contains a fact which is asserted 18.2 times per second. Inferences which are dependent on this fact are thus regularly re-evaluated.

3.5 Time-Valued Data

An essential adjunct to forward-chaining is persistence of data (rule memory). No benefit can be expected from forward chaining if every input change requires all inputs and intermediate conclusions to be reevaluated.

For process control, a reasonable assumption is that each input is valid for some finite time. Each rule returns a 2-tuple: a truth value, and an associated "expiration time." When the current clock time exceeds the expiration time, the rule is deemed to be unknown, and will be re-evaluated.

Temporal Algebra

This expert system is unique in that expiration times are calculated automatically when the rule's Boolean expression is evaluated. This is done by "overloading" the integer arithmetic and Boolean logic

---

11. Either an absolute clock time, or a given number of milliseconds in the future.

operators,[12] that is, defining a new set of operators which act on time-valued data. These operators, invented during this research, comprise a new mathematics -- a "temporal Boolean algebra."

Unary operators carry the time unchanged; e.g., if X is known until time t, NOT X is also known until time t. For most binary operators, the resultant expiration time is the lesser of the two inputs; if X is known until time t1, and Y until an earlier time t2, the sum X+Y is only valid until t2. Logical operators may have a more complex behavior: if X is known to be false until t1, and Y is known until an earlier time t2, then X AND Y will be false until t1, regardless of any possible change in Y at t2. Figure 3-2 illustrates the temporal relationships for AND and OR.

AND	unknown	FALSE,t2	TRUE,t2
unknown	unknown	FALSE,t2	unknown
FALSE,t1	FALSE,t1	FALSE, max(t1,t2)	FALSE,t1
TRUE,t1	unknown	FALSE,t2	TRUE, min(t1,t2)

OR	unknown	FALSE,t2	TRUE,t2
unknown	unknown	unknown	TRUE,t2
FALSE,t1	unknown	FALSE, min(t1,t2)	TRUE,t2
TRUE,t1	TRUE,t1	TRUE,t1	TRUE, max(t1,t2)

Figure 3-2. Truth Tables for Temporal Boolean Algebra.

Pruning

When expiration time is significant, the logic tree can no longer be pruned. For example, an early term in an AND expression may be false (rendering the expression false). But a further term may be false with a greater expiration time, increasing the expiration time of the result.

Data History

It may be desirable to maintain a "history" of a rule or an input value. This may be accomplished by redefining the *assert* method to record old values when they are replaced. For example, one suitable technique records "decades" of data in progressively larger intervals, storing a year's worth of data in 81

12. Overloading refers to the programmer's ability to define multiple actions for a language operator; for example, adding new behaviors to the operator "+" to add vectors, and OR Boolean values.

samples (Garland et. al. 1989). Object-oriented programming allows different "classes" of rules to be defined, each using a different method for history storage. Methods can be added as required to these classes, to retrieve or operate on this historical data.

3.6 Temporal Logic

Temporal logic is an extension of mathematical logic which allows descriptions of temporal relationships. (Gabbay, Hodkinson, and Reynolds 1994) define eight temporal relations which may pertain to statements A and B:

GA	"A will always be true (*henceforth*)"
HA	"A was always true (*heretofore*)"
FA	"A will sometimes be true (in the *future*)"
PA	"A was sometimes true (in the *past*)"
U(A,B)	"B will continuously be true *until* A is true"
S(A,B)	"B was continuously true *since* A was true"
TA	"A will be true *tomorrow*"
YA	"A was true *yesterday*"

The last two operators imply a discrete sampling period. They might better be called "previous" and "next" to emphasize that the period need not be daily.

A process control system needs to reason about current and prior observations, thus requiring the four "past" relations. (Ostroff 1989) shows that the fundamental future operators are *tomorrow*, *henceforth*, and *until*. Similarly, all past relations may be derived from *yesterday*, *heretofore*, and *since*.[13]

Implementation

No expert system to date has implemented the operations of mathematical temporal logic. When temporal operators are provided, they are typically "ad hoc" devices to address the needs of a particular

13. Strictly speaking, *since* is not fundamental, since it may defined in terms of *heretofore*, *and*, and *not*: $S(A,B) = \mathbf{H}(\neg(\neg B \wedge \neg\mathbf{H}(\neg A)))$. In loose English translation, the statement "B has been continuously true since A was first true" is restated "there has never been a time when B was false, but A was at some previous time true."

problem. This research has yielded a simple and efficient implementation of the relations listed above.

A simple modification to the *assert* method allows the expert system to evaluate "past time" temporal expressions. The *yesterday* (or *previous*) operator requires storing the "old" truth value of a rule, whenever it is updated. *Heretofore* requires a flag which is set if ever the rule evaluates false; if this flag is clear, the rule has always evaluted true. Similarly, a *past* flag may be set if ever the rule evaluates true.

A more general solution stores the previous value, the maximum integer value, and the minimum integer value for each rule. Boolean rules represent true and false as integers -1 and 0, respectively, so the minimum and maximum yield *past* and the inverse of *heretofore*. For integer-valued rules, the previous, maximum, and minimum values are useful in their own right, and allow computation of first difference and range.

3.7 Distributed Facts

When rules are implemented as subroutines which return truth values, they may be distributed across a network of processors through the use of Remote Procedure Call (Tanenbaum 1989). However, RPC requires blocking the caller until the reply is received, an undesirable trait in a real-time program. Also, a rule which is used frequently may cause excessive network traffic.

A better solution is to cache the truth value of each rule in each processor. The expiration time indicates when the cache is no longer valid. Only one processor, the *owner*, possesses the subroutine to evaluate any given rule. When its truth value expires in an different processor, that processor will request the new value from the owner; meanwhile it may either return "unknown," or block until the owner replies. This communication involves two network messages,

```
ASK  rule#

TELL rule#, truth value, expiration time
```

Rules have global identifiers; that is, all processors refer to the same rule by the same rule number.[14] A subset of rule numbers is reserved for "private" rules, that is, rules which are known to only one processor

14. When rules are defined as objects, the object identifier is an integer. The internal object format is common across all processors. See Appendix C for a description of the object implementation.

(and are thus never the object of an ASK or TELL message).

When chaining backward through the logic tree, an ASK message is generated during the *evaluate* phase for every unknown, "external" rule. Each ASK message will cause a TELL message in reply, requiring a total of two messages to transfer a new fact. When the TELL message is received, the *assert* method stores the fact.

Forward chaining requires the *propagate* phase to notify other processors of a new truth value, by sending a TELL message. Processors receiving the TELL message *assert* the new fact. If the network supports broadcast or multicast messages (Tanenbaum 1989), a single message suffices to update all dependent processors. Thus forward chaining may require less than half as much network traffic as backward chaining.

3.8 Distributed Rules

It is also desirable to move rules about the network. For example, a heavily loaded processor may wish to shift some of its inferencing burden to other processors.

When rules are compiled to machine code, they cannot be easily moved. The destination may be a different CPU, or may have a different load address for the routine. To solve this, the rule subroutines are compiled to a tokenized intermediate language (Apple 1979; Pelc 1992; IEEE 1994). This language, ITL (Inferencing Token Language) is a zero-operand, postfix virtual machine, and can be efficiently interpreted.[15] Tokens are represented in either 8 or 16 bits. Appendix D lists the current ITL primitives.

In addition to the previous classification of private vs. public, rules are divided into *bound* vs. *unbound*:

Private rules are known to only one CPU, and can be used only for that CPU's inferences.

Public rules are known to all CPUs; their truth values are cached in every CPU.

Bound rules must execute on a specific CPU; for example, a rule which uses a physical input.

---

15. Tests with the expert system have shown "token-threaded" code to be approximately half as fast as direct-threaded Forth code; however, optimization of the rule methods has regained this lost speed. See Chapter 5.

Unbound rules may execute on any CPU.

For a rule to be unbound, it must use only public facts, and it must be written entirely in ITL. Bound rules may use private facts and incorporate machine-language subroutines.

Four network messages support rule transfer across the network:

```
DEFINE-RULE rule#, ITL statement
I-OWN       rule#
WHO-OWNS    rule#
```

The owner of a rule may send it to another CPU with the DEFINE-RULE message. That CPU replies with a broadcast I-OWN message, which both acknowledges receipt and notifies other CPUs of the change of ownership. A WHO-OWNS broadcast allows any CPU to determine the current owner of a rule: the owner replies with an I-OWN message.

3.9 Implementation

The inference engine has been implemented in F-PC Forth for the IBM PC, and MPE-Forth forthe 68HC16, using the object-oriented extensions described in Appendix C. The basic rule object is defined as shown in Figure 3-3. Other rules may inherit this definition and add instance data, e.g. for temporal logic or data history.

The methods `:build`, `:dependent`, and `:evaluator` are used by the rule compiler.[16] The other methods have been previously described, except `:scrub` whose purpose is to reset the rule to its initial state (normally "unknown").

Both the evaluator and the dependency list (the backward and forward chains) are stored and executed as ITL statements. An ITL "escape" token calls machine-language or Forth subroutines. An optional "proponent" routine may be defined to perform an output action whenever the rule is asserted with a new value. The use of a proponent function or the "escape" token automatically flags the rule as

16. The rule compiler is available at all times, even while the application is running.

```
0 METHOD :build        \ initialize new rule object
1 METHOD :scrub        \ reset rule to "unknown"
2 METHOD :invoke       \ get rule's truth value
3 METHOD :assert       \ set new truth value
4 METHOD :evaluate     \ recalculate truth value
5 METHOD :propagate    \ recalculate dependent rules
6 METHOD :dependent    \ add rule#n as dependent
7 METHOD :evaluator    \ store ITL thread as evaluator
INHERIT COMMON
    CELL ITEM .Expiration   \ expiration time
    CELL ITEM .Value        \ current value
    60   ITEM .EvThread     \ ITL thread for the evaluator
    CELL ITEM .DepThread    \ pointer to dependency list
    CELL ITEM .Proponent    \ pointer to output routine
    CELL ITEM .Pending      \ flag: awaiting TELL message
    CELL ITEM .Owner        \ id of this rule's owner
    CELL ITEM .Export       \ export to this CPU (0=all)
    \ Performance monitoring data
    CELL ITEM .#Evals       \ # times Evaluated.
    CELL ITEM .#Asserts     \ # times Asserted.
    2 CELLS ITEM .EvalTime  \ accumulated evaluate time
    2 CELLS ITEM .PropTime  \ accumulated propagate time
CONSTANT FactSize
```

Figure 3-3. Definition of a Rule Object.

"bound" (owner=0). Newly asserted values may be exported to a single CPU, all CPUs (export=0), or no CPU (export=-1).

3.10 Future Work

Three desirable objectives have not yet been achieved.

Focused attention allows the knowledge base to change, depending on external conditions. This may involve storing multiple definitions for a rule, and dynamically using the definition appropriate to the current "milieu." One possible technique for dynamically changing subroutines has been described in (Broeders, Bruijn, and Verbruggen 1994).

Predictable or guaranteed response time has been addressed in many ways. The current system logs average evaluation and propagation times for each rule; this is intended to provide a rough prediction of response time. With the addition of focused attention, the "progressive reasoning" technique described by Broeders et. al. could be implemented, wherein progressively more complex rules are applied until the available time is consumed.

Expiration of knowledge would involve marking rules "invalid." This may perhaps be desirable if a rule has not been used for some time. A simple solution would employ an *invalid* ITL primitive, which would return an unknown result and generate an error alert if the expired rule was invoked. A more complex solution would reclaim the memory used by the expired rule; this would probably require expiration of its dependent rules as well.

Other promising areas for future research include:

"Just in time compilation" of ITL threads. Currently the ITL evaluator for a rule is interpreted, with a 100% speed penalty over compiled code. It would be possible, when a rule is transferred to a new CPU, to translate the ITL statement to a machine language subroutine (retaining the ITL definition so that the rule could be moved again). This would provide the speed of machine code with the portability of ITL, with a compilation cost only when a rule is moved.

Solicited rule transfer. At present only the owner of a rule can cause it to be transferred to a new CPU. Another network message, SEND-ME, could allow another processor to request ownership of a rule. This may be useful if that processor is lightly loaded, or if it uses the rule frequently.

Automatic load balancing. Rule transfer is performed manually in the existing system. A mechanism should be devised to allow the network to automatically determine the best location for each rule in the system, ideally in response to changing conditions. This is an ambitious optimization problem, and will doubtless require much more performance monitoring data for each rule and each processor.

CHAPTER 4. CONTROLLING THE FN ACCELERATOR

4.1 Description of the Problem

Figure 4-1 is a schematic diagram of the High Voltage Engineering model FN Tandem Accelerator, as installed at McMaster University.

Either of two sources -- Duoplasmatron or Sputter Source -- can provide an ion beam for acceleration. Each can produce negative ions of various elements, and accelerate them to a modest voltage (kilovolts).

The desired ion beam is selected by a steering magnet in the Injection subsystem. This subsystem also contains components to focus, steer, and analyze the beam, before it enters the main accelerator tank via evacuated stainless steel tubes ("beamlines").

The Generator is a large tank which contains a Van de Graaff generator. This generator charges a central electrode to several megavolts positive, accelerating the incoming ion beam. At the center, electrons are stripped from the ions, changing them to positive ions. The electrode then provides an additional several megavolts of acceleration as it drives the positive ion beam out of the tank and into the high-energy beamline.

The Analysis subsystem uses a steering magnet to turn the beam and guide it through a set of slits. Since the beam deflection is a function of the ion energy, charge state, and mass, this provides accurate control of beam energy.

The Distribution subsystem contains additional focusing elements, and a steering magnet to route the ion beam to one of eight "beam lines." Each beam line ends in a different experimental "target."

Figure 4-1 also shows the devices which are used to monitor the beam. The "cup" is a Faraday cup, which safely collects the beam and directly indicates beam current. Since this blocks the beam, it cannot be employed while the beam is in use; it doubles as a safety device to stop the beam. The "BPM" is

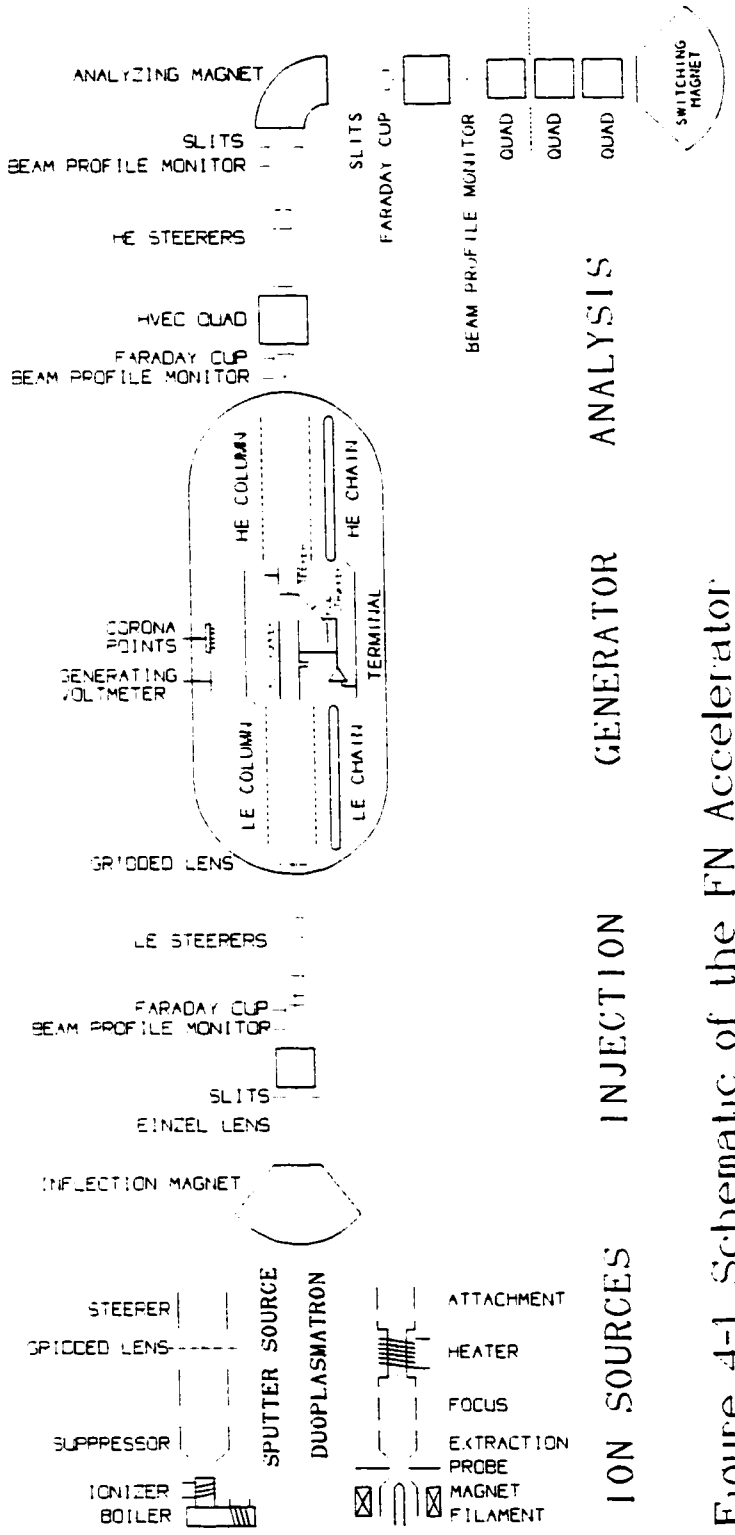


Figure 4-1 Schematic of the FN Accelerator

a Beam Profile Monitor. It observes the beam with relatively little disruption, and can be employed while the beam is in use; however, its interpretation is much more complex.

4.2 Basic Operating Sequence

The basic operating sequence of the FN accelerator, as documented by the manufacturer (High Voltage Engineering 1968), is as follows:

a) Activate the generator, and adjust for the desired terminal voltage.

b) Activate the desired ion source, and adjust for desired output.

c) Adjust the injection subsystem to select the ion source and for desired output.

d) Adjust the generator subsystem for desired output.

e) Adjust the analysis subsystem for desired output.

f) Adjust the distribution subsystem (switching magnet) to select the desired beam line, and adjust for desired output.

The phrases "adjust" and "desired output" are deceptively simple in this description. For example, the injection subsystem has seven adjustments, which must select the desired ion source and the desired ion mass, steer the beam into the accelerator, and achieve the desired beam profile and intensity. This description also omits the problem of keeping the beam in adjustment, once the accelerator is running. However, it does illustrate an important characteristic: the accelerator subsystems are largely independent. For instance, the injection subsystem can be adjusted without reference to the analysis subsystem.[1]

4.3 The Terminal Charging Subsystem

An example of the complexity of even one subsystem can be seen in the terminal charging mechanism, Figure 4-2. Two variable high-voltage power supplies deposit charge on the charging chains. The chains, moving in an endless loop, carry this charge to the terminal. As charge accumulates on the terminal, its voltage increases. Charge bleeds off from the terminal through the two fixed column

1. There is some interaction between subsystems, which becomes important in the "steady-state" operation of the accelerator.

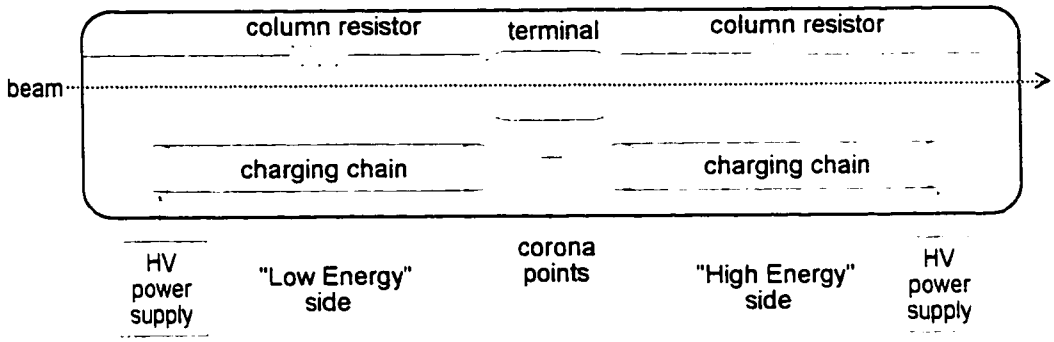


Figure 4-2. Terminal Charging Mechanism

resistors, and also by corona discharge to the adjustable corona points.

Mathematically this can be modeled by assuming:

a) The DC high voltage output of the supply is some function f_{hv} of the AC input voltage.

b) The charge current deposited on the chain is proportional to the DC voltage.

c) The motion of the chain introduces a fixed time delay (about 0.5 second) in the charging current.

d) The terminal has a fixed capacitance to ground.

e) The corona discharge current is proportional to terminal voltage, for any given position of the corona points.

f) There is a time delay from the terminal to the corona points, which is fixed for any given position of the points.

g) There is a fixed time delay in the terminal voltmeter (not shown in Figure 4-2).

The charging model using these approximations is shown in Figure 4-4. This model can be subjected to a classical feedback system analysis.

4.4 The Distributed Processor Hardware

Figure 4-3 illustrates the control system for the FN Tandem Accelerator. It includes six "local" controllers, corresponding to the six subsystems of the accelerator, plus a central control station.

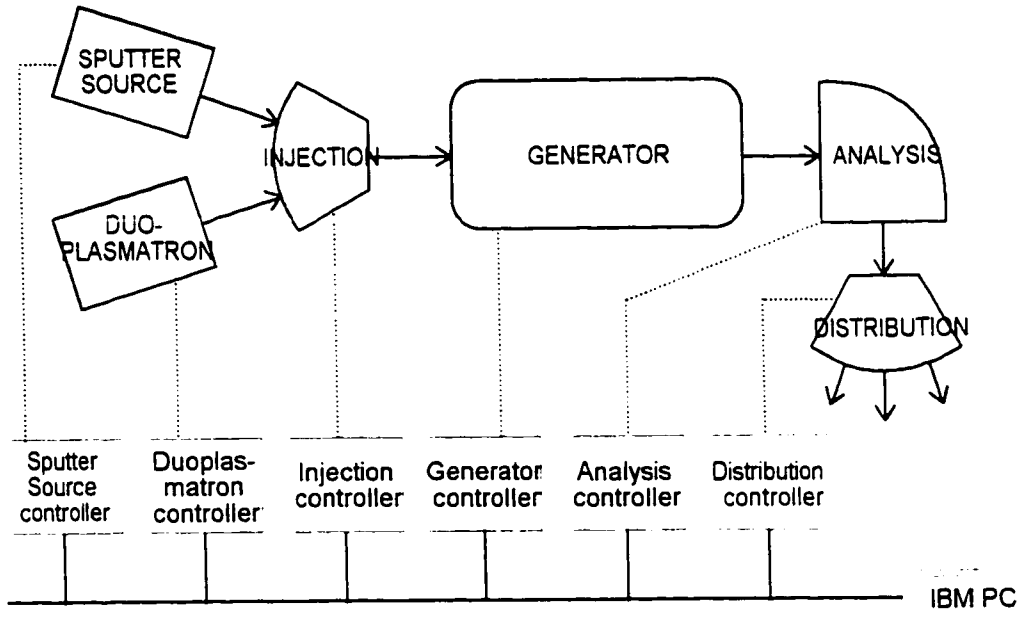


Figure 4-3. The Distributed Processor Hardware

Local Controllers

Each local controller is a New Micros Inc. NMIX-0026 single-board computer, employing the Motorola 68HC16 microprocessor. Two New Micros interface boards are added to provide the "standard" I/O complement for each controller:

Eight 10-bit analog-to-digital converters

Four 12-bit analog-to-digital converters

Two 12-bit digital-to-analog converters

Six TTL inputs

Three TTL outputs

The local controllers incorporate both simple active control algorithms, such as PID[2] loops, and inference engines for rule-based reasoning.

---

2. Proportional-Integral-Derivative, or PID, refers to a feedback system in which the control signal is a linear combination of the error signal, its time integral, and its first derivative.

Operator's Console

All system functions are controlled from the operator's console, an IBM PC with 80386 processor and a VGA graphics display. This system is responsible only for operator input, display, and network supervision. No interencing tasks are performed by the PC.

Network Communications

Eventually it will be desired to install the local controllers as close as possible to their respective subsystems, i.e., at the accelerator machinery. Since some of this machinery is at a potential of several kilovolts, the staff of the Accelerator Laboratory requested the use of fiber optic communications. This led to the development of a token-ring network using the asynchronous ports of the 68HC16s and the IBM PC (described in Appendix B). The development hardware uses point-to-point RS-232 serial cables to simulate the fiber optic links.

Safety Considerations

Licensing regulations under the Atomic Energy Control Board (AECB) require that no accelerator functions be left solely under control of the computers. It must be possible for the human operator to gain control of the accelerator at all times, regardless of any system malfunction. Prior work at the McMaster Accelerator Laboratory has used mechanical linkages from the computer to the physical control knobs of the accelerator (Lind, Poehlman, and Stark 1993). This project uses entirely electronic controls with "fail-safe" operation.

Most sensor readings are displayed on panel meters in the Tandem electronics rack. The local controllers were designed and developed to noninvasively sense the meter readings through the "Meter Repeater," a circuit board which mounts on the back of each panel meter and monitors the current or voltage at the meter terminals. Each board includes a simple first-order RC filter to reduce high-frequency noise.

The Tandem accelerator uses Variacs for many of its control actuators. These variable autotransformers provide from 0 to 110 VAC to a number of accelerator subsystems, such as high voltage

power supplies. For computer control each is replaced with a "solid state Variac," a phase-controlled Triac. The Triacs can be operated manually on command or if the computer fails.

These interfaces, and others devised for this project, are described in more detail in Appendix A.

4.5 I/O Processing Software

Additional conditioning of the sensor inputs is performed in software by the local controllers. A second-order infinite-impulse response (IIR) filter, operating at a sample rate of 18.2 Hz, attenuates high-frequency noise on the inputs. The "zeros" of this filter are adjusted to eliminate any aliased 60 Hz noise. These filters use the DSP capability of the 68HC16 microprocessor.

Control outputs are also assisted by numerical software. For instance, a Proportional-Integral-Derivative (PID) feedback loop achieves a desired DC charging voltage by adjusting the AC input to the HV power supply. The nonlinear transfer functions of the Triac control and the HV power supply are

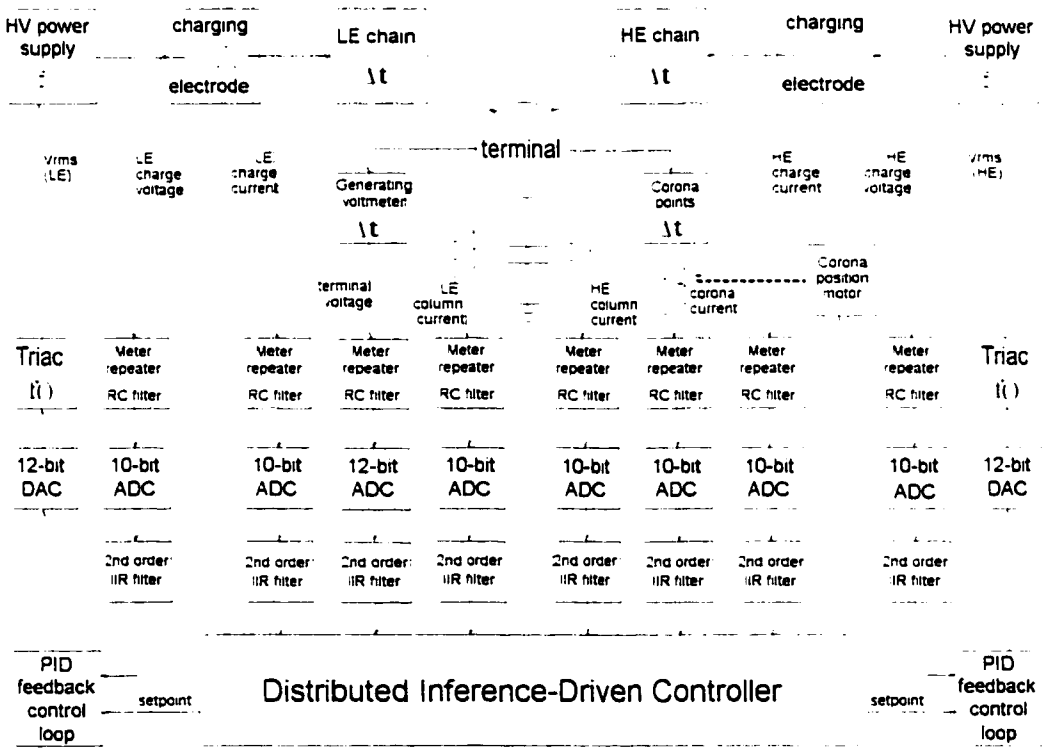


Figure 4-4. Terminal Charging Model

"hidden" from the expert system, which only need to specify a desired charging voltage "setpoint."

These processing functions are illustrated in Figure 4-4.

4.6 Operator Interface

Since the purpose of this research is to investigate a control technology, and not to produce a finished product for accelerator control, only a rudimentary operator interface is included. Sensor data is displayed in tabular form on a text-only screen. A command interpreter resides on the PC and also on the local controllers. The operator may issue commands to any or all local controllers via the network; this is a great aid in development and debugging.

4.7 The Terminal Control Strategy

The objective of the terminal control subsystem is to achieve a specified voltage on the terminal, with a usable corona current, while avoiding sparks. The corona current must be within a limited range (typically 50 to 100 uA) in order for the terminal stabilizer hardware to function (Cairns, Greene, and Kuehner 1974). Excessive corona current must be avoided, as it will cause potentially damaging sparks from the highly charged terminal. Sparks may also occur if the terminal voltage is excessive, or if the terminal is charged too quickly (rate of change of terminal voltage is excessive).

The terminal voltage is increased by supplying charge faster than the column resistors and corona points are bleeding off charge. Corona current is increased by moving the corona points inward, and

Terminal Voltage / Corona Current	Low	OK	High
Low	Increase charge	Extend points	Extend points
OK	Increase charge	Do nothing	Reduce charge
High	Retract points	Retract points	Reduce charge

Figure 4-5. Terminal Charging Heuristic

decreased by moving the points outward. There is a strong interaction between these two adjustments. Corona current is proportional to terminal voltage. Also, withdrawing the corona points will cause a rapid increase in terminal voltage -- as the corona current decreases, the net charging current will increase.

After discussions with Jim Stark, Director of Operations of the McMaster Accelerator Laboratory, the decision matrix of Figure 4-5 was devised. This represents the following heuristic reasoning.

a) If the terminal voltage and the corona current are both low, increasing the charge will increase both.

b) If the terminal voltage is low and the corona current is OK, more charge is required.

c) If the terminal voltage is low and the corona current is high, retracting the points will both increase the terminal voltage, and decrease the corona current.

d) If the terminal voltage is OK and the corona current is low, it is necessary to extend the points.

The remaining four adjustments are the "mirror image" of these. The ninth case, when the terminal voltage and the corona current are both OK, requires no action.

The effectiveness of this control strategy, and some modifications to it, are discussed in Chapter 6.

CHAPTER 5. EXPERT SYSTEM PERFORMANCE

5.1 The Benchmark Program

Expert system performance is generally measured in number of Logical Inferences per Second (LIPS). Accurate measurement requires a test problem which involves a large number of inferences. One such is the Towers of Hanoi, the classic problem of moving a stack of disks from one tower to another, moving one disk at a time, and never putting a large disk above a small one. The expert system solution involves only a three rules (Matheus 1986):

a) If the current goal is "move a tower of N disks," remove the current goal, and push three new goals: move top disk to temporary location, move tower of N-1 disks, and move top disk from temporary location.

b) If the current goal is "move tower of 1 disk," replace it with the goal "move disk."

c) If the current goal is "move disk," remove the goal and move the disk.

A "goal stack" keeps track of intermediate goals.[1] These three rules[2] will be evaluated repeatedly until the problem is solved (the goal stack is empty). To solve an eight-disk problem, the system must fire 512 rules[3] and make 255 moves.

5.2 Uniprocessor Performance

The expert system invented during this research (and described in Chapter 3) has been christened TEXMEX -- Threaded EXecution Micro EXpert -- to emphasize that it compiles a rule network to executable, threaded[4] code, and that it can run on small ("micro") processors. Several versions of this software were developed as the research progressed.

---

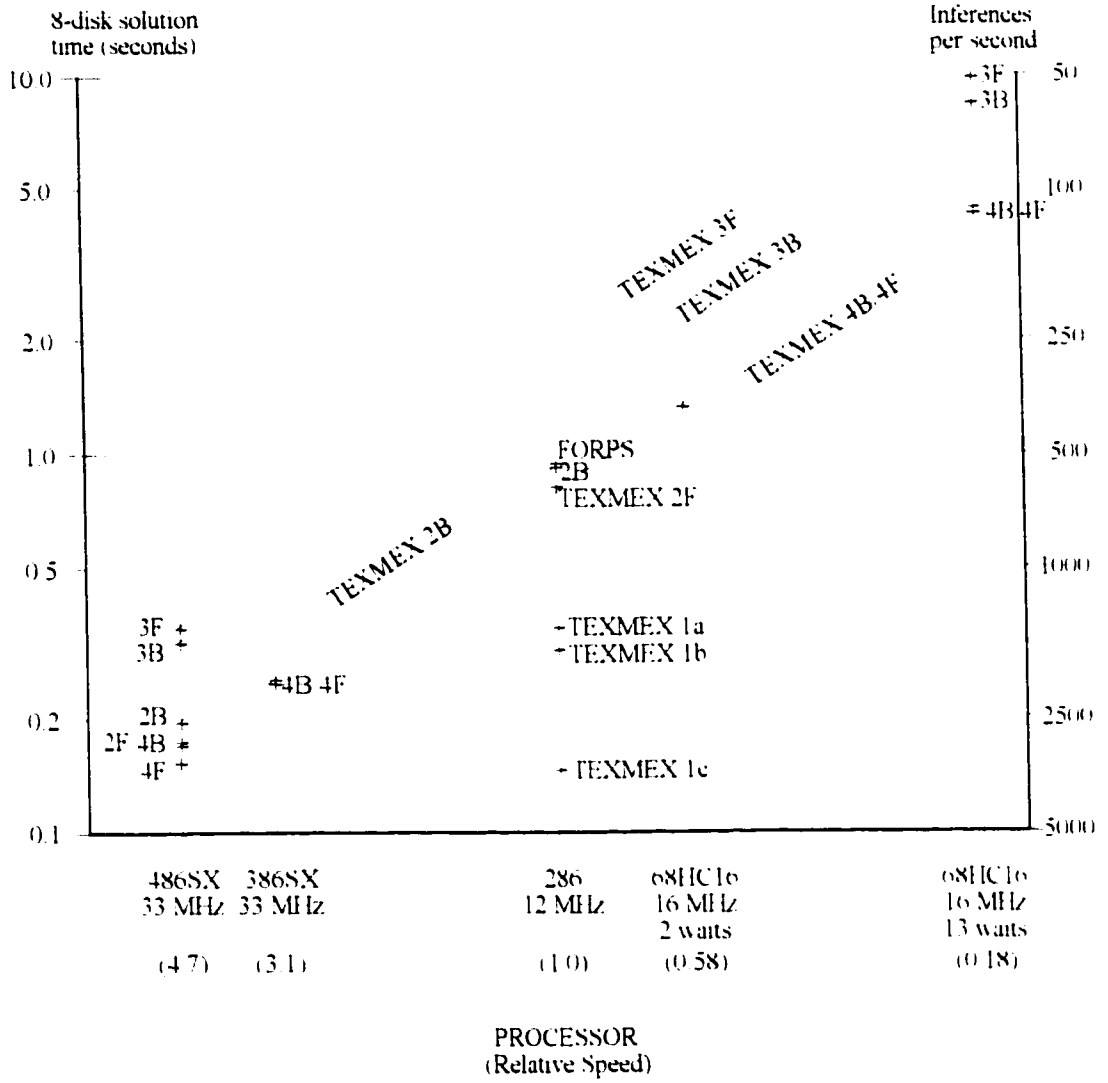
1. In a LISP-based expert system, this stack must usually be simulated with a list. See Appendix F for a CLIPS version of the Towers of Hanoi.

2. Matheus includes a fourth rule which places the original goal on the stack.

3. This number comes from Matheus, who seems to count only firings of rules (b) and (c); it is used here for consistency with Matheus' reported results.

4. See Chapter 3 for an overview of threaded code, or (Kogge 1982) for a detailed treatment.

Figure 5-1 summarizes the performance of this system on a single processor. Data from FORPS, the FORth Production System of (Matheus 1986), is included in this table for comparison.



Notes:

TEXMEX 1a is Direct Threaded Code, with pruning; backward chaining only
TEXMEX 1b is as 1a, with critical sections converted to machine code (hand optimized)
TEXMEX 1c is as 1b, with an optimized goal stack (using Forth's data stack)
TEXMEX 2, 3, and 4 are given for both forward (F) and backward (B) chaining
Processor speed is given relative to a 12 MHz 80286, based on benchmark results

Figure 5-1. Towers of Hanoi Solution Times, One Processor

TEXMEX 1, the first version of the expert system, was written in F83 for the IBM PC. It performed backward chaining only, and compiled the rule network to direct threaded code. Three successive refinements are shown in Figure 5-1. Variant (a) used rule memory and pruning[5] to speed evaluation. In (b), key rule subroutines and logical operators were rewritten in assembly language for improved speed. Finally, (c) dispensed with the auxiliary stack used in the FORPS example to record goals; instead, the Forth data stack was used.

TEXMEX 2, written in Pygmy Forth for the IBM PC, was the first version to allow backward ("/B") or forward ("/F") chaining. It was also the first version to use temporal algebra and expiration times; this prevented "pruning" of the logic tree. This, plus the more complex logic of the inference engine, made TEXMEX 2 about one-third as fast as the comparable TEXMEX 1a. Like TEXMEX 1, rules were compiled to direct threaded code. No speed is lost by forward-chaining; indeed, it is slightly faster than backward chaining.

TEXMEX 3, the first to operate on distributed processors, introduced the Inferencing Token Language "ITL," and compiled rules to a tokenized program rather than direct-threaded code. It was written in F-PC for the IBM PC, and MPE Forth for the 68HC16 microprocessor. The use of tokenized code makes TEXMEX 3 roughly half the speed of TEXMEX 2.

TEXMEX 4 was the object-oriented implementation of TEXMEX 3. The token interpreter and some key methods (e.g. *assert, evaluate*) were rewritten in assembly language for improved performance. This regained the speed lost by TEXMEX 3; even though it still uses a tokenized representation, TEXMEX 4 is slightly faster than TEXMEX 2.

TEXMEX 4 is also twice as fast as TEXMEX 3 on the 68HC16 processor. The speed of this processor was tripled when its internal wait-state generator was reduced from 13 (the reset default) to 2 (the number actually required for the memory in use).

The "inferences per second" statistic was computed from the 8-disk problem. No one system was

---

5. See Chapter 3, section 3.3.

evaluated on all processors; however, enough comparable tests were performed to allow a relative speed metric to be computed for each CPU. These are shown at the bottom of Figure 5-1, with the 12 MHz 80286 arbitrarily[6] used as the "reference" speed.

5.3 Distributed Performance

To test the performance of TEXMEX on a distributed system, a "worst case" test was devised. The Towers of Hanoi problem was divided among three processors as follows:

1) The "tower expert" knows only how to move a tower (rules a and b).

2) The "disk expert" knows only how to move a disk (rule c).

3) The "goalkeeper" keeps track of the current goal (the goal stack).

Thus no processor can perform more than one inference before having to refer to another processor. This is intended to maximize the amount of network traffic, and provide a pessimistic measure of system performance. A "real" problem would be factored so as to *minimize* network traffic, so that each processor could solve, unassisted, the largest possible subproblem.

Figure 5-2 summarizes the results of these tests.

Forward Chaining

At first glance, the reduced performance of the distributed expert system is striking. Using a 486SX/33 MHz, 386DX/16 MHz, and 8088/8 MHz,[7] TEXMEX 3/F solves the 8-disk problem in 44.0 seconds (as compared with 0.353 seconds for the 486 alone). However, instrumenting the network reveals that 2040 packets of 15 bytes each are sent in the solution of this problem. Since the division of tasks ensures that no two sequential messages originate from the same processor, each packet is followed by a two-byte token. In the *best* case (assuming no idle network time) this involves 34,680 bytes, or 36.13 seconds at 9.6 Kbaud. Thus the inference engine is consuming *at most* 12 seconds,[8] as shown in the

---

6. Primarily because all of the early benchmarks used this processor.

7. Three processors of widely varying speeds were chosen to create a more demanding test of the network communications software.

8. More likely, the inference engine is consuming only about two seconds, and network traffic is taking 42 seconds (i.e., a network utilization of 86%).

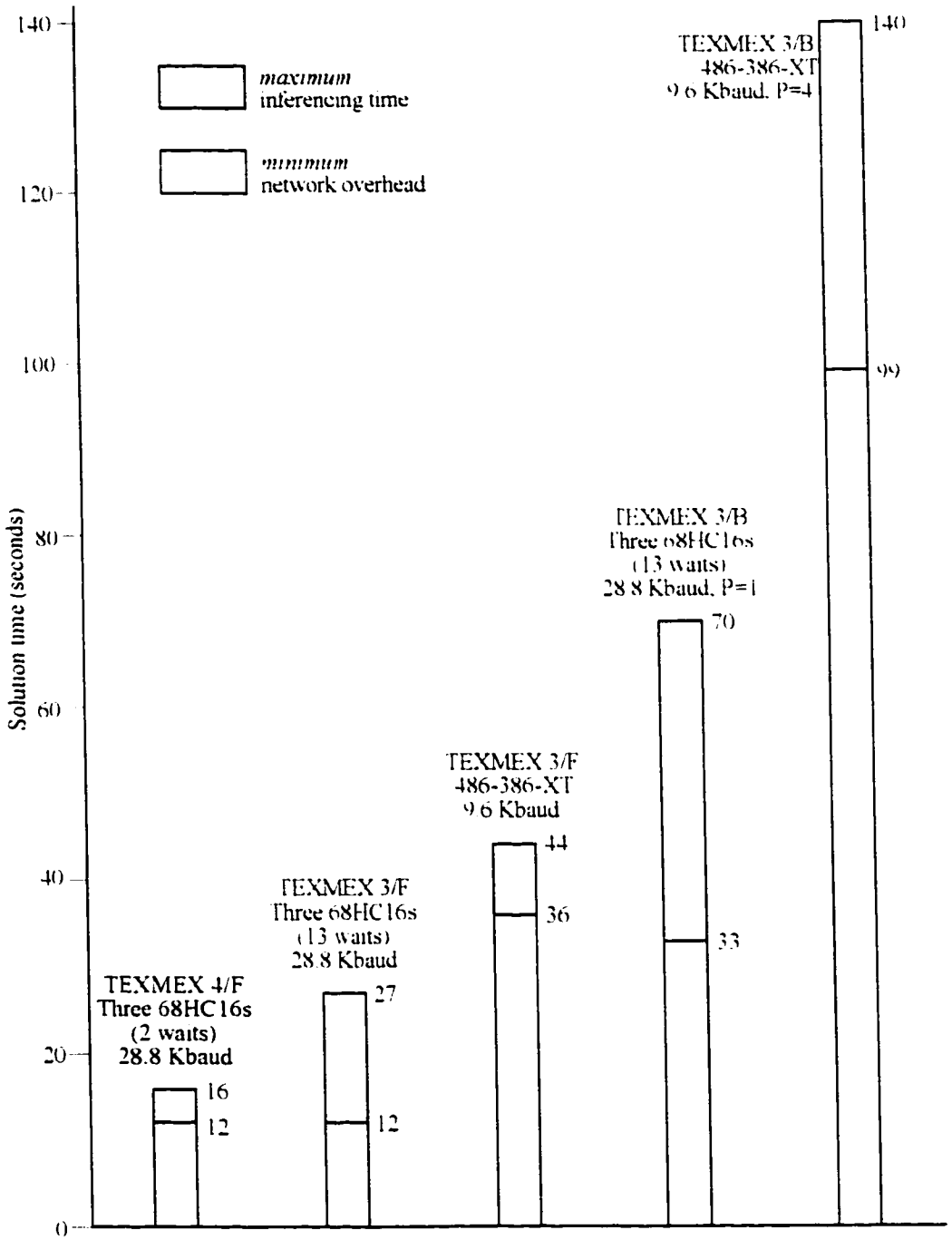


Figure 5-2. Towers of Hanoi Solution Times, Three Processors
(Eight disk problem)

middle column of Figure 5-2. Thus the limiting factor appears to be the network, a conclusion which is confirmed by the tests at 28.8 Kbaud. The first column, for TEXMEX 4/F running on 68HC16s with two wait states, shows a minimum message time of 12 seconds, and a maximum inferencing time of 4 seconds. A single 68HC16 can solve this problem in 1.35 seconds. It is reasonable to assume that the "missing" 3.65 seconds are largely due to less-than-perfect network utilization.[9]

On the 68HC16s, TEXMEX 3/F (13 wait states) solved the 8-disk problem in 26.8 seconds, while TEXMEX 4/F (2 wait states) took 16.6 seconds. This 10.2 second difference is comparable to the 8.6 second difference on a single 68HC16 (10.0 vs. 1.4 seconds). On a 68HC16 with 13 wait states, it appears that about half the total solution time is consumed by inferencing and internal message processing. It appears that the expert system will not be entirely network-limited on embedded processors. Raw CPU performance is still a significant concern, especially if slower (e.g., 8-bit) CPUs are to be used.

Backward Chaining

The backward chaining solution was problematic, since in this case there is no way for one processor to tell the others that a new goal is present. Instead, the tower and disk experts must wait for their current goal to expire, upon which they will query the goalkeeper for a new goal. This expiration time is critical: if too long, the system will spend time waiting idly. If too short, the experts may ask for a new goal before a new goal is available, increasing network traffic with useless messages. Various values of "persistence" time (given in 18.2 Hz clock ticks as "P" in Figure 5-2) were tried. With P=3, repeated 8-disk trials required from 6337 to 7663 packets on the network. The number is more than the expected double of the forward-chaining solution[10], because the backward chaining system cannot use broadcast messages to reduce traffic. The average message length is 13 bytes, plus two bytes for the token; assuming the "best case" of 6337 packets to solve the problem, this represents a message passing overhead of 99 seconds at 9.6 Kbaud, or 33 seconds at 28.8 Kbaud.

---

9. A network utilization of 82% would account for this discrepancy.

10. A doubling of messages is expected because backward chaining requires two messages (ASK and TELL) to transfer a new fact, while forward chaining requires only one (TELL).

Once more, the 68HC16 with 13 wait states spends about half of its time (about 37 seconds) on internal processing. This compares with about 15 seconds for TEXMEX 3/F on the same hardware configuration. Since the single-processor solution times were nearly identical for TEXMEX 3/B and TEXMEX 3/F, it seems likely that the increase in "internal" processing is due to the larger number of messages; that is, message processing (exclusive of inferencing) represents a significant load.

5.4 Comparison to Other Systems

Figure 5-3 compares the performance of the TEXMEX embedded expert system with several other published and commercial packages.

System	Processor	Time (in seconds) to solve 8 disks
TEXMEX 4/F	486SX/33 MHz	0.251
TEXMEX 4/B	486SX/33 MHz	0.258
FORPS	486SX/33 MHz	0.404
FORPS	80286/12 MHz	0.96
TEXMEX 4/F	68HC16/16 MHz	1.36
CLIPS 5.1	486SX/33 MHz	1.42
FORPS	68000/10 MHz	2.31
TEXMEX 4/F	three 68HC16s	16.6
OPS5	TI Explorer	37.8*
REAL-OPS	68000/8 MHz	51.4*
KAPPA PC	486SX/33 MHz	81
OPS5	VAX 11/780	120*
OPS5	MicroVAX II	240*

*this figure extrapolated from data supplied by (Dress, 1986).

Figure 5-3. Comparison of Expert Systems

TEXMEX 4 running on a single processor is two to three orders of magnitude faster than the LISP-based expert system shells, OPS5 and KAPPA, running on comparable hardware. Even the use of a dedicated LISP processor (TI Explorer) does not accelerate OPS5 to one-hundredth of the speed of TEXMEX. Not evident from this table is the fact that the resources required by the commercial expert shells -- megabytes of RAM and a hard disk -- render them useless for embedded control.

The C Language Interactive Production System, CLIPS, is an example of the "state of the art" in real-time expert systems. TEXMEX 4 is almost five times faster than CLIPS. Furthermore, CLIPS requires the resources of a personal-computer: a demonstration CLIPS program compiled to over 155 kilobytes, far in excess of the memory available on an embedded microprocessor.

The only truly comparable system is the FORth language Production System, FORPS. This expert system is well suited to resource-limited processors, and appears only slightly slower than TEXMEX. But since FORPS does not create a network of rules, it must exhaustively search its rule base on every rule firing, with the result that the inferencing speed is inversely proportional to the number of rules defined.[11] The Towers of Hanoi benchmark, having only four rules, is especially favorable to FORPS; a "real" problem would run much more slowly.

5.5 Observations

The statistics quoted for TEXMEX in Figure 5-3 are for a system using a token interpreter. The results obtained for TEXMEX 1 (Figure 5-1) indicate how much more speed is available by compiling to a more efficient code representation (direct threaded code). Compiling directly to machine code would be even faster. "Just in time" compilation gives the benefits of machine code with the portability of tokenized code. The postfix token language ITL is well suited to this, since is can be quickly and easily compiled. This is a promising area for future research.

It has been said that forward chaining is better than backward chaining for process control, since process control is driven by the arrival of new data (e.g. from sensors). From this work, it can now be

11. (Matheus, 1986) estimates 0.7 ms per rule per inferencing cycle, thus a system with 700 rules would perform approximately two inferences per second.

added that forward chaining is also better for distributed inferencing, due to several aspects:

a) A new fact can be transferred with one network message rather than two.

b) A new fact can be broadcast to several CPUs with a single message.

c) Backward chaining requires that queries continually be sent to discover changes in a previously "known" fact. If the query period is too short, the network will be cluttered with useless messages. If the query period is too long, the new fact will not be discovered promptly, and the effective inferencing speed is reduced.

Fortunately, the forward and backward chaining techniques devised in this project achieve equivalent inferencing speeds. There is no incentive to use the "wrong" chaining method for the application when real-time performance is critical.

CHAPTER 6. ACCELERATOR CONTROL PERFORMANCE

6.1 Performance Criteria

The distributed expert system was applied to the problem of controlling the terminal charging subsystem of the FN Tandem Accelerator (see Chapter 4). This is a desirable test for several reasons:

a) It is presently solved heuristically by a human operator.

b) It is susceptible to an analytical solution (see Figure 4-4).

c) It involves a complex interaction between control variables (charge and corona position).

d) It requires cooperation between processors.

e) It can be compared with similar previous work (Lind and Poehlman 1992).

Terminal charging is also a useful test because it has four clearly measurable performance criteria:

a) Response time to a requested change in terminal voltage.

b) Amount of overshoot of desired terminal voltage.

c) Coarse regulation of desired terminal voltage.[1]

d) Maximum rate of change of terminal voltage.

The rate of change statistic is important because increasing the voltage too rapidly may cause sparking within the accelerator.

6.2 Manual Operation

Figure 6-1 is the record of a typical manually operated accelerator run (16 April 1996, Winston Williams, operator). The terminal was charged to 6.4 MV for a period of roughly 30 minutes for "conditioning;" then it was raised to the operating voltage of 7.6 MV. The expanded view in Figure 6-2 shows that it took almost 270 seconds to raise the voltage by 1.2 MV. First the charging voltage was raised, and the terminal reached equilibrium near 7.2 MV (at t=2500 seconds). Then the corona points

1. For fine regulation, a completely separate "terminal stabilizer" subsystem is activated by the operator.

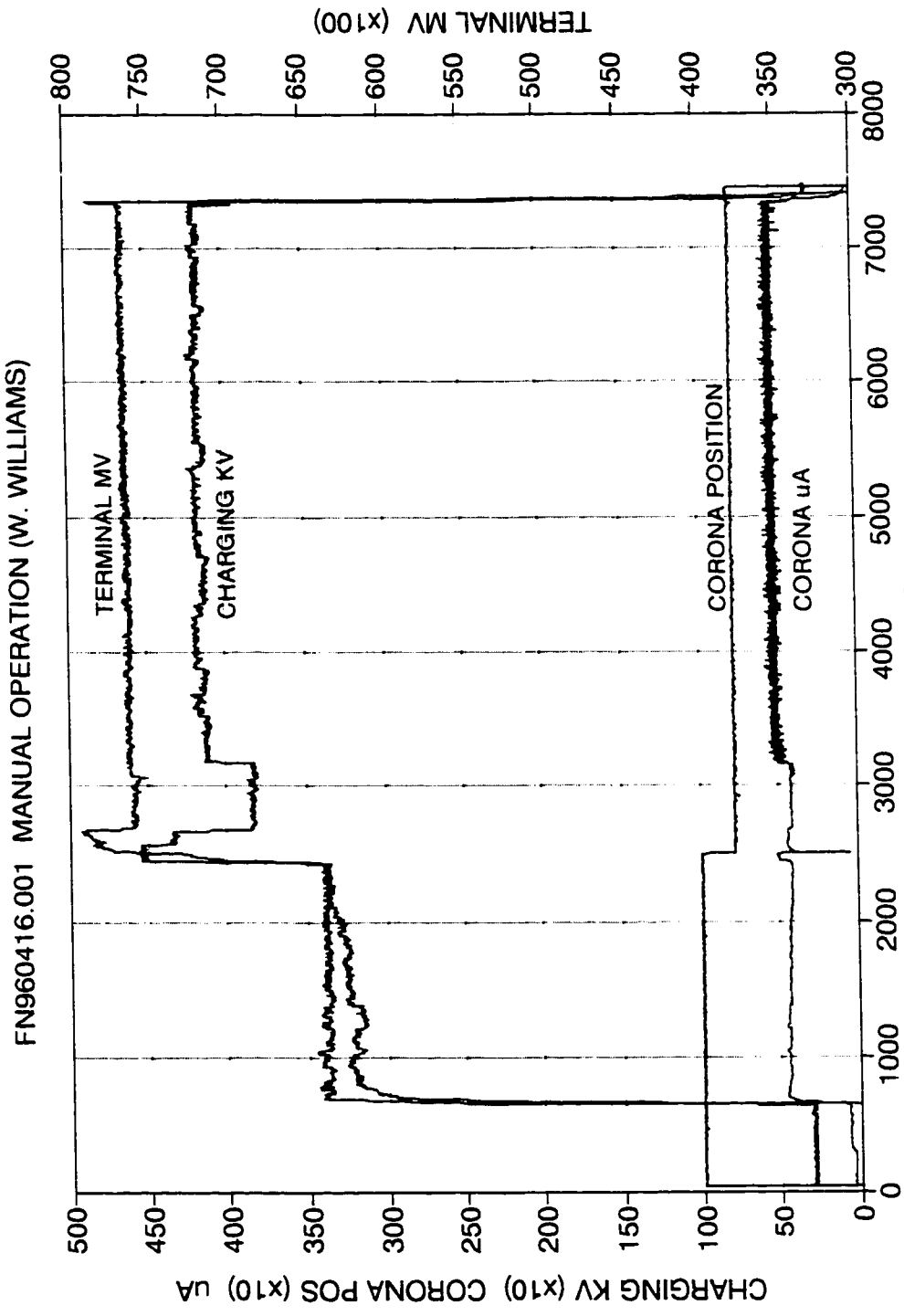


Figure 6-1. Manual Operation

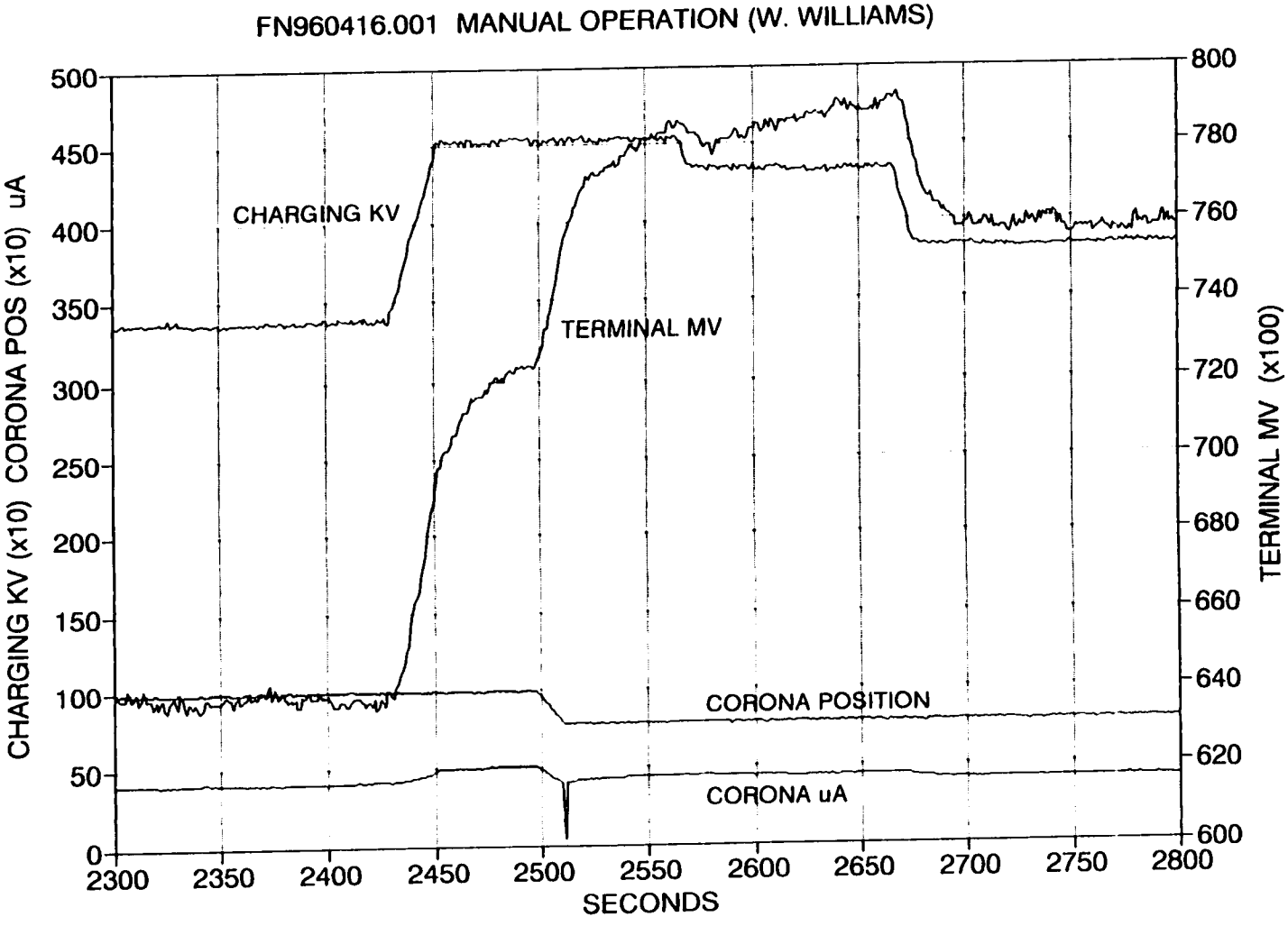


Figure 6-2. Manual Operation, Enlarged

were withdrawn about two inches; this caused a sudden rise in voltage to 7.8 MV. Two reductions of charging voltage were needed to bring the terminal voltage back to 7.6 MV; the peak voltage was slightly over 7.9 MV. The maximum rate of change was 160 kV/sec at t=665 seconds.

6.3 Control Strategy #1

The first automatic control trials made no attempt to adjust corona position. Only the terminal charging was controlled, to achieve a desired terminal voltage.

Figure 6-3 shows the simplified decision matrix of strategy #1. This strategy is actually

Terminal Voltage	Low	OK	High
Action	Increase charge	Do nothing	Reduce charge

Figure 6-3. Control Strategy #1 Heuristic

represented in six rules.[2] Three are intermediate conclusions about the current terminal voltage:

ChargeUp is true if the terminal voltage is too low.

ChargeDown is true if the terminal voltage is too high.

DeltaMV indicates the change in terminal voltage since the last inferencing cycle (1 second ago).

It is unique in that it returns a numeric result, and that it has a temporal aspect.

Two rules are conclusions which have an output action:

MoreCharge increases the charging setpoint, if the voltage is too low *and* DeltaMV is too low.

LessCharge decreases the charging setpoint, if the voltage is too high *and* DeltaMV is too high.[3]

The output action of these rules is to change the setpoint of a PID controller; this is accomplished by embedding a "conventional" procedure call in the CONCLUDES phrase of the rule. The sixth rule is a "safety" rule to prevent excessive rate of change of terminal voltage:

OverCharge decreases the charging setpoint, if DeltaMV is excessively high.

---

2. In retrospect, these rules could have been given better names.
3. DeltaMV is a signed value, the threshold is set so that any positive value is "too high."

Several additional "rules" perform digital filtering on the sensor inputs, or execute the PID control software for the actuators. This is done by embedding procedure calls (*2ndFilter* or *ApplyPID*) in the evaluator function of each rule. These "I/O rules" are re-evaluated periodically by making them dependent upon the rule *18Hz*, which in turn is fired 18.2 times per second[4] by a timer routine.

For the first test of this control strategy, the setpoint for both high voltage charging supplies was incremented by 1 unit (0.1 kV) whenever the terminal voltage was more than 2 units (0.02 MV) less from the desired value. As can be seen in Figure 6-4, this caused a large oscillation in charging setpoint, and a smaller oscillation in the terminal voltage, as the system constantly overcorrected. Subsequent tests showed that this could be cured by decreasing the increment to a fractional value,[5] or by requiring a larger error threshold before making an adjustment. These tests were greatly facilitated by the fact that the increment and threshold were stored in ordinary Forth variables, and that interpretive access to the language was available during testing.

Figure 6-5 shows the greatly improved performance using an increment of 0.5 units (0.05 kV). Generally, a smaller increment gave better regulation, but at the cost of a slower slew rate. Another variant, designated Strategy 1a, incremented the two charging supplies alternately, so as to give finer control of charging voltage.[6]

6.4 Control Strategy #2

The second series of tests attempted simultaneous control of terminal voltage and corona current, using the decision matrix of Figure 4-5. Normal noise fluctuations in the corona current reading caused the system to "hunt" back and forth for a stable position. Also, an error in the control rules caused the system to simultaneously change the charging voltage and corona points position; it was quickly discovered that these two interacting variables should *not* be adjusted together. A modification, Strategy

---

4. This is the standard clock interrupt of the IBM PC, adopted on the 68HC16s for consistency.

5. Fractional values are accumulated until a change of one whole unit occurs. This "unit," 0.1 kV, is the smallest change which can be output to the charging power supplies.

6. Incrementing one supply by one unit (10 kV) yields half the effective increase of incrementing both supplies by one unit.

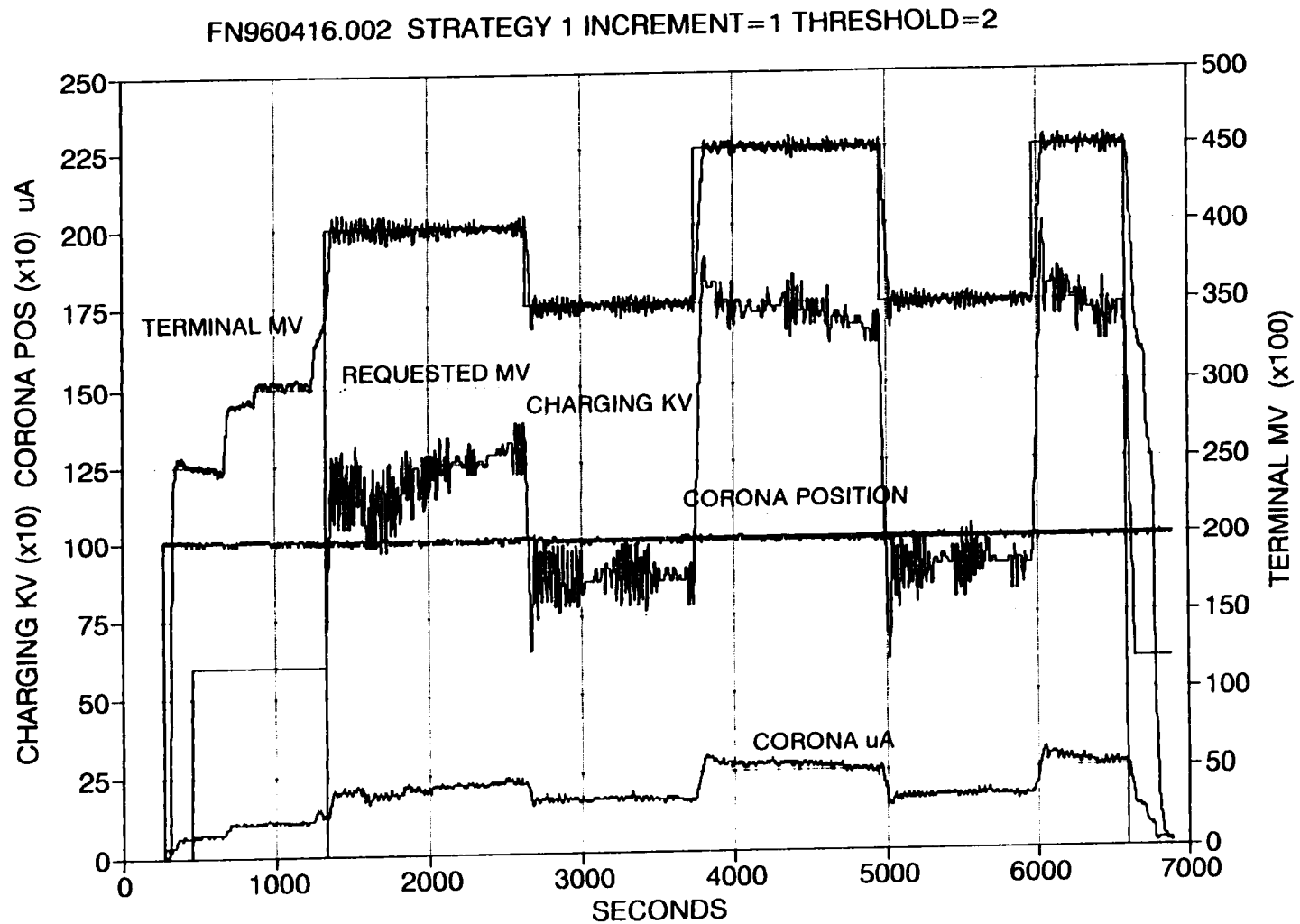


Figure 6-4. Strategy #1 Oscillation

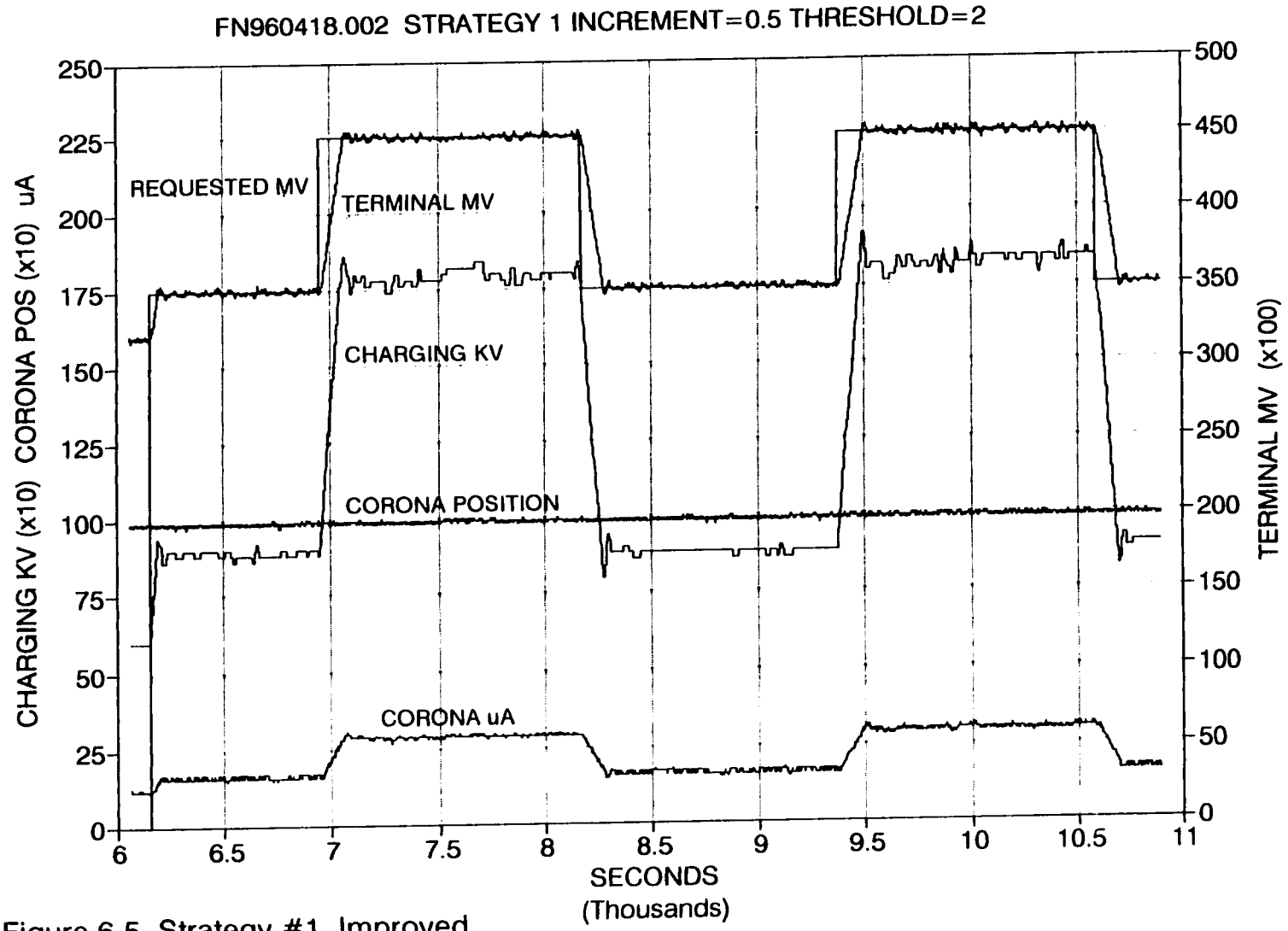


Figure 6-5. Strategy #1, Improved

2a. solved the noise problem by adding hysterisis (in the form of a dual threshold) to the *CoronaLoad-low* and *CoronaLoad-high* rules. The rules were rewritten so that *only* corona position *or* charging voltage is adjusted -- never both together.

In addition to simultaneous control of two interacting variables, this was the first strategy to employ two cooperating processors. CPU "0D" measured terminal voltage and controlled the charging supplies with eight rules, exporting the facts *TermMV-high* and *TermMV-low* to the other processor. CPU "0C" measured corona current and controlled corona position with five rules, exporting the facts *CoronaLoad-high* and *CoronaLoad-low*.

Two new safety rules were added. One stopped charging if a terminal voltage limit was reached. The other stopped charging if the terminal voltage suddenly dropped more an 0.5 MV -- evidence that a spark has occurred in the generator tank.[7]

6.5 Control Strategy #3

The combination of corona and charge adjustment can cause a large overshoot of terminal voltage; this is undesirable near the desired voltage. Also, a large increment and large threshold will reach the desired voltage sooner, but small increment and threshold will yield better regulation. This

Terminal Voltage / Corona Current	Low	Slightly low	OK	Slightly high	High
Low	Large charge increase	Small charge increase	Do nothing	Small charge decrease	Extend points
OK	Large charge Increase	Small charge increase	Do nothing	Small charge decrease	Large charge decease
High	Retract points	Small charge increase	Do nothing	Small charge decrease	Large charge decrease

Figure 6-6. Control Strategy #3 Heuristic

7. The accelerator formerly employed an electronic spark detector, but now only this heuristic method is used.

suggests using one strategy when far from the desired voltage, and another when near. This hybrid approach was implemented as Strategy #3.

Appendix E gives the program listing of the rules for strategy #3.

Good results were obtained using a "large" increment of 0.5 unit, and a "small" increment of 0.2 unit. Previous tests revealed that a corona adjustment could cause a jump of 0.2 MV, therefore, when less than 0.2 MV from the desired voltage, the "fine adjustment" rules take effect. The result is shown in Figure 6-7

6.6 Discussion of Results

The terminal voltage control experiments are summarized in Figure 6-8. In addition to the automatic control strategies just described, five test runs were performed with human operators to provide comparison data.

Two principal figures of merit are voltage regulation (RMS error), and maximum rate of change of terminal voltage (slew rate). Both of these statistics are somewhat compromised by noise in the terminal voltmeter reading. To estimate this noise, two tests were performed using the independent "beam stabilizer" circuit, which regulates the terminal voltage to within 2 kV (Cairns, Greene, and Kuehner 1974). These suggest about a 10 kV RMS error in the terminal voltage reading, corresponding to one count of the external digital voltmeter.

The "refined" expert control strategies exhibited measured variations of about 15 kV RMS Assuming that voltmeter noise and terminal voltage fluctuations are statistically independent[8], this corresponds to a regulation of approximately 11 kV RMS. "Unregulated" manual operation exhibits variations of 20 to 43 kV RMS.

Since slew rate is computed as the difference between successive measurements, it is strongly affected by random fluctuations. It is reasonable to assume, over a long series of samples, that the largest difference will be exaggerated by one count (10 kV); this accords with casual observation of the voltmeter

8. In this case, the total variance is the sum of the two independent variances; RMS is the square root of the variance.

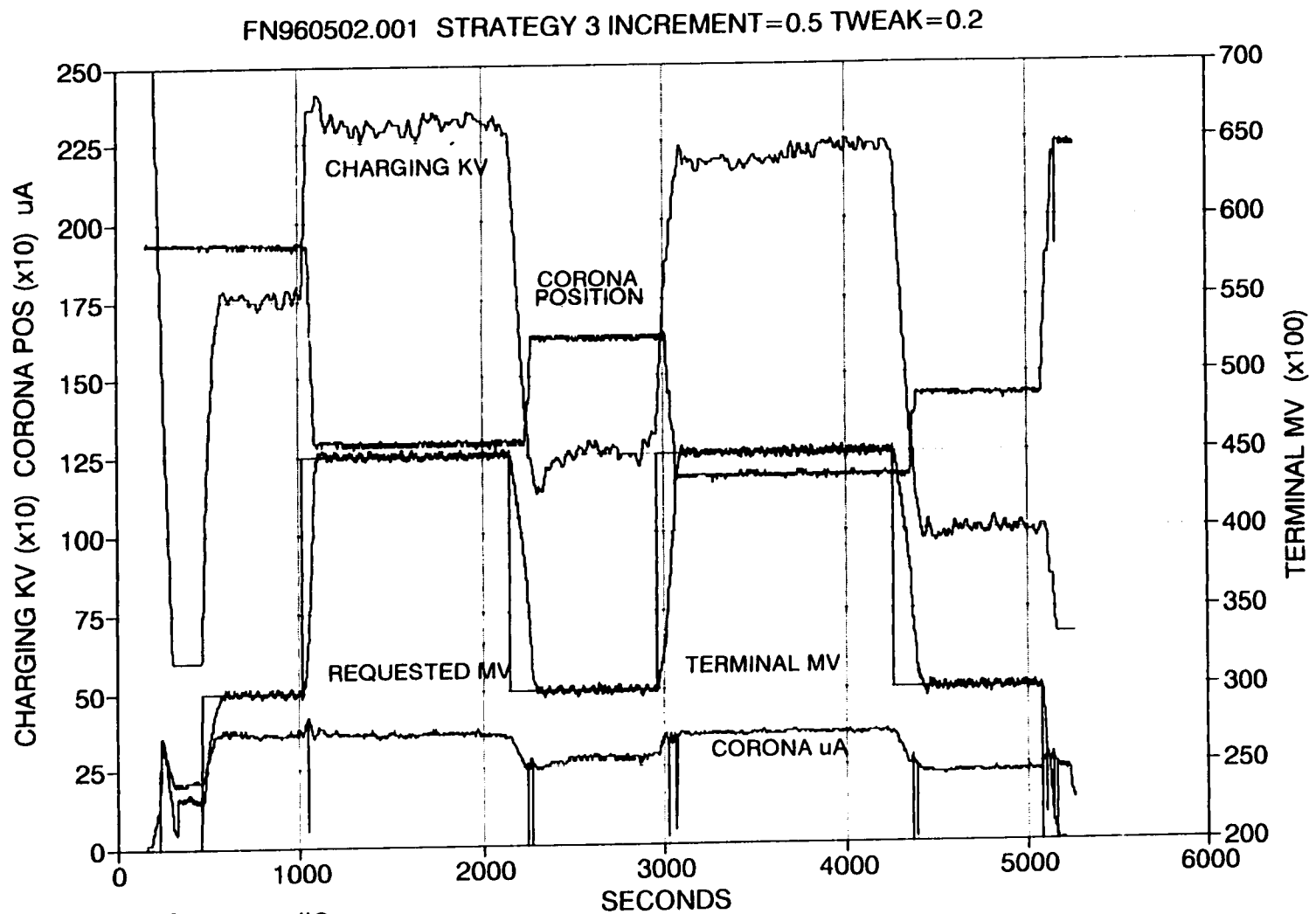


Figure 6-7. Strategy #3

Experiment Number	Control Strategy	Terminal Voltage Threshold	Belt Charge Increment	Belt "Fine" Increment	Corona Current Threshold	Regulation (RMS error)	Maximum Slew Rate
960416.001	manual					13.8 kV [1]	160 kV/s
960417.001	manual					n/a	70 kV/s
960501.001	manual					44.2 kV	150 kV/s
960501.002	manual					10.9 kV [1]	n/a
960501.003	manual					21.8 kV	180 kV/s
960416.002	1	0.02 MV	0.1 kV			20.8 kV	60 kV/s
960416.003	1	0.04 MV	0.1 kV			24.2 kV	50 kV/s
960418.001	1	0.02 MV	0.02 kV			16.0 kV	50 kV/s
960418.002	1	0.02 MV	0.05 kV			15.3 kV	40 kV/s
960418.003	1 [2]	0.02 MV	0.05 kV			11.6 kV	40 kV/s
960419.001	1a	0.02 MV	0.02 kV			12.7 kV	60 kV/s
960419.002	1a	0.02 MV	0.05 kV			14.3 kV	50 kV/s
960426.001	2	0.02 MV	0.05 kV		5 uA	n/a	70 kV/s
960426.002	2a	0.02 MV	0.05 kV		10 / 7 uA	11.8 kV	50 kV/s
960426.003	2a	0.02 MV	0.05 kV		5 / 2 uA	15.6 kV	90 kV/s
960430.001	3	0.02 MV	0.05 kV	0.01 kV	5 / 2 uA	15.7 kV	50 kV/s
960502.001	3	0.02 MV	0.05 kV	0.02 kV	5 / 2 uA	15.6 kV	50 kV/s
960502.002	3	0.02 MV	0.1 kV	0.02 kV	5 / 2 uA	17.1 kV	60 kV/s

Note 1. For these tests the automatic terminal voltage stabilizer was activated.

Note 2. This was a duplicate of the previous test, to observe the effect on terminal voltage of manual changes in the corona points position

Figure 6-8. Summary of Terminal Control Trials

reading. Then the "better" control strategies limit slew rate to about 40 kV/second[9], compared with 60 to 170 kV/second for different human operators.

Comparing the experimental runs of Figure 6-1 and Figure 6-7 shows that the expert system is faster to reach a new voltage (80 vs. 250 seconds, for a 1.5 MV change), and has less overshoot (30 kV vs 300 kV). These are also important figures of merit for terminal control. Some test runs were recorded in which a human operator was faster than the expert system to set a new voltage. This usually involved the human operator using his judgment to exceed the recommended maximum rate of increase of terminal voltage; presumably, this knowledge could be incorporated into additional rules which allow the expert system to use a higher slew rate under specified conditions (such as terminal voltage below 5 MV). Invariably, though, the human operators overshoot the desired terminal voltage by substantially more than the expert system.

6.7 Observations

It is noteworthy that control strategy #3 was programmed in approximately 30 minutes. This "rapid prototyping" is a tribute both to the ease of writing rule-based control algorithms, and the advantage of implementing the rule base as an extension of the programming language (thereby allowing all of the interactive programming tools to be applied to the knowledge base, as well). Control strategy #2 required roughly half a day to develop, but this was largely the "design" phase, in consultation with accelerator lab staff. Once a strategy is designed, actually writing the rules to embody that strategy is a quick and straightforward task.

All of the expert control strategies operated with a "major inferencing cycle" of one second (that is, all of the control rules were evaluated once per second). This was felt to be a reasonable choice for initial experiments, since it accords roughly with a human reaction time. It would be instructive to explore control strategies with a longer inferencing cycle, to discover a lower bound on expert system response time. Reducing the cycle time would reduce the inferencing load on the processors, and thus

9. The 90 kV/s recorded for experiment 960426.003 is probably a statistical fluke, given the repeatability of the slew rate limits for all other test runs.

permit larger and more complex problems to be handled. It may also be useful to run different parts of the knowledge base on different schedules: perhaps charging current can be adjusted every five seconds, but safety rules such as terminal overvoltage should be checked every second.

Corona current and terminal voltage are strongly coupled, and should properly be controlled by the same CPU. That they were separated here was an accident of the input/output assignment -- a fortuitous accident, because it offered a very effective test of two processors cooperating on one problem.

This may well be carried to a greater extreme in future control systems. One can readily visualize a network of more and smaller CPUs, with -- for instance -- one CPU located at the corona points, dedicated solely to measuring corona current and controlling corona position.[10] The extent to which the problem can be distributed is limited mainly by the amount of interprocessor communication required, which in turn depends on how many facts must be exported and imported. A problem for future research will be to discover where the tradeoff between inferencing (each rule in a different processor) and communication (all rules in one processor) is optimum.

---

10. From an installation standpoint this is advantageous, because it can greatly reduce the amount of cabling required in the physical plant.

CHAPTER 7. SUMMARY AND CONCLUSIONS

7.1 Summary

There are many real-world process control problems for which heuristic, or rule-based, control is preferable. An example of such a problem is a Tandem particle accelerator, which is mathematically intractable, and typically is under the control of an experienced human operator. The cost and difficulty of training operators has spurred efforts to capture and automate this expert knowledge.

Conventional expert systems, alas, are ill-suited to process control. Most expert systems are large and slow, requiring fast processors, large memories, and frequent delays for garbage collection. Few have a knowledge of time; most are limited to reasoning about the present instant. In general, the only way to solve a larger problem is to buy a larger computer. Those systems which do allow a problem to be subdivided across multiple computers, retain a central "blackboard" machine as a bottleneck.

This research has developed a novel expert system for process control, which makes four original contributions to the field:

a) The expert system achieves extremely high inferencing performance on "low end" processors, by compiling backward and forward inference networks to procedural code. The inference engine and knowledge base require less than 6 Kbytes of program storage, and no dynamically-allocated data storage. Speeds in excess of 300 Logical Inferences Per Second have been achieved on a 68HC16 microcontroller, and over 3000 LIPS on a 33 MHz 486SX computer.

b) The system uses a unique model to perform cooperative reasoning across a distributed network of inferencing agents: the "consultant/advisory" model. A new portable representation of knowledge (rules) has been devised, as well as a portable object-based representation of data (facts).

c) The system can manipulate time-valued data, and automatically handles the expiration and

renewal of input data. A conceptual extension incorporates, for the first time, the formal relations of temporal logic, allowing statements to be made about temporal relationships in the knowledge base.

d) The system has successfully operated a process control problem never before solved with inference-driven techniques: controlling the terminal voltage and corona current of an FN Tandem Accelerator. In doing so, it achieves comparable or better speed, better control of slew rate, less overshoot, and better long-term regulation than a human operator.

Three incidental, but no less important, contributions have been made in the course of this research.

e) The expert system allows an unprecedented freedom to mix procedural and inferential programming. Inference routines may freely call procedures and may directly interact with sensors and actuators. Procedural code may access facts and invoke selected rules. One benefit is extremely rapid prototyping of control strategies.

f) A new local-area network, the Asynchronous Token Ring (Appendix B), was developed during this research. This is an open-ended, deterministic network that requires no special hardware beyond that offered by most microcontrollers, and which can be directly used with fiber optic transmission. It is unique in that it can sequence and acknowledge broadcast messages.

g) New hardware has been developed for "technology insertion" into the 1960's-vintage control electronics of the Tandem Accelerator (Appendix A). The new interface techniques include a transparent monitoring of panel meters, and a retrofit to high-voltage power supplies to allow computer control.

7.2 Conclusions

Heuristic control problems are now within the scope of inexpensive, embedded microcontrollers. This new expert system, TEXMEX, provides ample inferencing speed with limited resources. The problem of nondeterministic response time, commonly associated with expert systems, has been overcome. Furthermore, a central inference engine is not necessary; process control problems of greater complexity

can easily be distributed over a network of small processors.

Commercial interest has been expressed in TEXMEX, and development will likely continue with the intent of converting this research software into a marketable product.

7.3 Future Work

Unfortunately, the McMaster University Tandem Accelerator was closed down permanently, just after the control experiments described herein. There was no opportunity to automate more than the terminal charging subsystem. It would be highly instructive to bring more of the accelerator under expert control -- both to test the performance of the system with a more complex problem, and to study the more subtle interactions that can occur between the accelerator subsystems.

The implementation of formal temporal logic is still immature. Research is needed to explore different means to achieve the temporal relations described in Chapter 3. These relations must also be tested for utility in a variety of control problems; perhaps a less formal logic will suffice.

Chapter 3 also suggested several other areas for further development of the expert system. For example, automatic rule relocation is likely to be a large and fruitful research subject.[1] "Just in time" compilation, offering the speed of compiled code with the portability of tokens, is another likely focus.

Only a minimal Inferencing Token Language has been implemented, and the design of this language has been driven by the immediate needs of the accelerator control problem. This is inadequate for the design of what should be a general-purpose language. This class of problem has recently become of significant interest,[2] and touches upon both language design and computer architecture (instruction representation).

This work was originally intended to be portable to a variety of embedded microprocessors. At present, only the 68HC16 and 8086 (IBM PC) microprocessors have been used. The system should be

---

1. One complex aspect of this problem is balancing the often-conflicting demands of processor time and network bandwidth. For example, moving a rule from a heavily-loaded processor to a lightly-loaded processor may be desirable from the standpoint of CPU utilization, but may cause an undesirable increase in network traffic.

2. Primarily, thanks to the advent of the Java language and its tokenized representation.

tested on more platforms, particularly the ubiquitous but resource-limited 8-bit microcontrollers such as the 8051 and 68HC11.

The Asynchronous Token Ring network is still in a primitive state. While only incidental to the main goals of this research, this network revealed itself to be quite an interesting -- and a potentially profitable -- problem. Further development, as outlined in Appendix B, could produce a valuable networking product for embedded processors.

BIBLIOGRAPHY

Acharya, Anurag, Milind Tambe, and Anoop Gupta. "Implementation of Production Systems on Message-Passing Computers." IEEE Transactions on Parallel and Distributed Systems 3, no. 4 (July 1992): 477-487.

American Radio Relay League (ARRL). The ARRL Handbook for Radio Amateurs. Seventieth Edition. Newington, CT: American Radio Relay League, 1992.

Anway, Allen. "Running a Lime Kiln with PolyForth's EXPRESS." In Proceedings of the 1994 Rochester Forth Conference, by the Institute for Applied Forth Research. Rochester, NY: Institute for Applied Forth Research, 1994, 1-3.

Apple Computer Inc. Apple Pascal Reference Manual. Cupertino, CA: Apple Computer, 1979.

Bartel, Joe, and Marla Bartel. "Data Structures in Forth." The Computer Journal no. 41 (November-December 1989): 9-11.

Bartel, Marla. "Lazy Evaluation." The Computer Journal no. 43 (March-April 1990): 26-27.

Bell, James R. "Threaded Code." Communications of the ACM 16, no. 6 (June 1973): 370-372.

Berg, D. "A Computer-Based Companion for Operation of the McMaster University Tandem Accelerator." M.Sc. dissertation, McMaster University, 1989.

Bokhari, Asghar Ali. "An Investigation of a Real-Time Multitasking System for Ion Source Operation Within a Particle Accelerator." M.Sc. dissertation, McMaster University, 1993.

Bonissone, Piero B., and Peter C. Halverson. "Time-Constrained Reasoning Under Uncertainty." The Journal of Real-Time Systems 2 (May 1990): 25-45.

Bradley, Mitch. "Structured Data with Bit Fields." In Proceedings of the 1984 Rochester Forth Conference, by the Institute for Applied Forth Research. Rochester, NY: Institute for Applied Forth Research, 1984, 188-192.

Broeders, H. M. T., P. M. Bruijn, and H. B. Verbruggen. "Real-time direct expert control using progressive reasoning." Engineering Applications of Artifical Intelligence 2, no. 2 (June 1989), 109-119.

Brown, R. J. "Dreams." Journal of Forth Application and Research 6, no. 4 (1994), 279-314.

Butler, Jim. "A Simple RS-485 Network." Circuit Cellar INK no. 21 (June/July 1991)

Butler, Jim. "Embedded Controller Networking Alternatives." Circuit Cellar INK no. 26 (April/May 1992)

Cairns, J. E., M. W. Greene, and J. A. Kuehner. "A Terminal Stabilizer for a Tandem Accelerator." Nuclear Instruments and Methods 114 (1974): 489-494

Castellano, M., L. Catani, G. Di Pirro, S. Guiducci, C. Milardi, P. Patteri, M. Serio, and L. Trasatti. "A Distributed VME Control System for the LISA Superconducting LINAC." IEEE Transactions on Nuclear Science 36, no. 5 (October 1989): 1619-1623

Clearwater, S., W. Cleland, F. Provost, E. Stern, and Z. Zhang. "A Real-Time Expert System for Trigger-Logic Monitoring." Nuclear Instruments and Methods in Physics Research A 293 (1990): 491-495.

Corkill, Daniel. "Blackboard Systems." AI Expert 6, no. 9 (September 1991): 40-47

Coyle, Frank. "Expert Systems - Ready for Real Time?" IEEE Expert 5, no. 5 (October 1990): 12.

Crowley-Milling, Michael, and Winfried Busse. Preface to "Proceedings of the Third International Conference on Accelerator and Large Experimental Physics Control Systems, Berlin, Germany, October 18-23, 1993." Nuclear Instruments and Methods in Physics Research A 352 (1994): viii-xi.

Daems, G., V. Filimonov, V. Homutnikov, F. Perriollat, Yu. Ryabov, and P. Skarek. "A knowledge-based control method: application to accelerator equipment setup." Nuclear Instruments and Methods in Physics Research A 352 (1994): 325-328.

D'Ambrosio, Bruce, Michael R. Fehling, Stephanie Forrest, Peter Raulefs, and B. Michael Wilber. "Real-Time Process Management for Materials Composition in Chemical Manufacturing." IEEE Expert 2, no. 7 (Summer 1987): 80-93.

Davies, W. G. and R. E. Howard. "Program to Automatically Steer the Beam in TASCC Beam Lines." Excerpt from TASCC status report. Chalk River, Ontario: AECL Chalk River Laboratories, 1993. (private communication)

Decker, K. S., V. R. Lesser, and R. C. Whitehair. "Extending a Blackboard Architecture for Approximate Processing." The Journal of Real-Time Systems 2 (May 1990): 47-79.

DeMooy, S., and W. F. S. Poehlman. "Using Expert Systems for Control Purposes." Proc. 3rd Symposium on Applications of Expert Systems in DND, May 2-3, 1991. Kingston, Ontario: Royal Military College, 1991, 17-36.

DeMooy, Samuel. "Development Towards a Control Program for a Model KN Van de Graaff Particle Accelerator." M.Sc. dissertation, McMaster University, Hamilton, Ontario, 1991.

Dewar, Robert B. K. "Indirect Threaded Code." Communications of the ACM 18, no. 6 (June 1975): 330-331.

Dodhiawala, Rajendra, Vasudevan Jagannathan, Larry Baum, and Tom Skillman. "The First Workshop on Blackboard Systems." AI Magazine 10, no. 1 (Spring 1989): 77-80.

Dress, W. B. "REAL-OPS: A Real-Time Engineering Applications Language for Writing Expert Systems." Journal of Forth Application and Research 4, no. 2 (1986): 113-124.

Elder, R. Bruce. "Linking C and PDC Prolog," AI Expert 8, no. 4 (April 1993): 30.

Englemore, R. S., A. J. Morgan, and H. P. Nii. Introduction to Blackboard Systems, ed. Robert S. Englemore and Anthony Morgan, 1-22. Reading, MA: Addison-Wesley, 1988.

Forgy, Charles L. "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem." Artificial Intelligence 19 (1982): 17-37.

Francis, J. C., and R. R. Leitch. "Intelligent Knowledge-Based Process Control." In Control 85, vol. 2. London: Institution of Electrical Engineers, conference publication #252, 1985, 483-488.

Gabbay, Dov M., Ian Hodkinson, and Mark Reynolds. Temporal Logic: Mathematical Foundations and Computational Aspects. Oxford: Oxford University Press, 1994.

Galton, Antony, ed. Temporal Logics and Their Applications. London: Academic Press, 1987.

Garland, W. J., W. F. S. Poehlman, N. Solntseff, J. Hoskins, and L. Williams. "Intelligent real-time system management: towards an operator companion for nuclear power plants." Engineering Computations 6, no. 2 (June 1989): 97-115.

Green, Peter E. "AF: A Framework for Real-Time Distributed Cooperative Problem Solving." In Distributed Artificial Intelligence, ed. Michael N. Huhns, 153-175. London: Pitman, 1987.

Greiner, B. F. and D. J. Caswell. "Evaluation of a Commercial System for CAMAC-based Control of the Chalk River Laboratories Tandem-Accelerator Superconducting-Cyclotron Complex." IEEE Transactions on Nuclear Science 39, no. 2 (April 1992): 210-214.

Goldstein, David. "The Distributed AI Toolkit." AI Expert 9, no. 1 (January 1994): 34-37.

Gurd, D. P., and M. Crowley-Milling, eds. Preface to "Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems, Vancouver, British Columbia, Canada, October 3 - November 3, 1989." Nuclear Instruments and Methods in Physics Research A 293 (1990): vii-xii.

Hamilton, Jennifer. "A Model for Implementing an Object-Oriented Design without Language Extensions." ACM SIGPLAN Notices 31, no. 1 (January 1996), 36-43.

Hayes-Roth, Barbara. "Architectural Foundations for Real-Time Performance in Intelligent Agents." The Journal of Real-Time Systems 2 (May 1990): 99-125.

Hentdlass, Tim. "Embedded Node Collectives." In 1991 FORML Conference Proceedings, by the Forth Interest Group. Oakland, CA: Forth Interest Group, 1991, 77-82.

Hewlett-Packard Corporation. Optoelectronics Designer's Catalog. Palo Alto: Hewlett- Packard, 1985.

Hwang, Kai, and Faye A. Briggs. Computer Architecture and Parallel Processing. New York: McGraw-Hill, 1984.

High Voltage Engineering Corporation. HVI-593 Instruction Manual: Model FN Tandem Van de Graaff Accelerator. High Voltage Engineering Corporation, 1968.

Institute of Electrical and Electronics Engineers (IEEE). Token Ring Access Method and Physical Layer Specifications. IEEE Standards for Local Area Networks, ANSI/IEEE Standard 802.5-1985. New York: IEEE, 1985.

Institute of Electrical and Electronics Engineers (IEEE). Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices. IEEE Standard 1275-1994. New York: IEEE, 1994.

Intel Corporation. "8272, 8273-4, 8273-8 Programmable HDLC/SDLC Protocol Controller." Chapter in Intel Component Data Catalog. Santa Clara: Intel, 1981.

Ionescu, Dan, and Ioan Trif. "A hierarchical expert system for computer process control." Engineering Applications of Artificial Intelligence 1, no. 4 (December 1988): 286-302.

Johnson, Harold E. Jr., and Pieor P. Bonissone. "Expert System for Diesel Electric Locomotive Repair." Journal of Forth Application and Research 1, no. 1 (September 1993): 7-15.

Jones, Vincent C. MAP/TOP Networking. New York: McGraw-Hill, 1988.

Kannan, R., and W. H. Dodrill. "DAIS: A Distributed AI Programming Shell." IEEE Expert 5, no. 6 (December 1990): 34-42.

Keitel, R., and L. Wilson. "TICS - Control System Software for the TISOL Facility at TRIUMF." IEEE Transactions on Nuclear Science NS-34, no. 4 (August 1987): 849-852.

Kim, Tag Gon, and Bernard P. Zeigler. "AIDECS: An AI-Based, Distributed Environmental Control System for Self-Sustaining Habitats." Artificial Intelligence in Engineering 5, no. 1 (1990): 33-42.

Kogge, Peter M. "An Architectural Trial to Threaded-Code Systems." IEEE Computer 15, no. 3 (March 1982), 22-32.

Koopman, Philip J. Jr. An Architecture for Combinator Graph Reduction. San Diego: Academic Press, 1990.

Kosko, Bart. Fuzzy Engineering. Upper Saddle River, NJ: Prentice-Hall, 1997.

Kumar, B. P. Ajith, J. Kannaiyan, P. Sugathan, and R. K. Bhowmik. "The NSC 16 MV tandem accelerator control system," Nuclear Instruments and Methods in Physics Research A 343 (1994) 327-330.

Laffey, Thomas J., Preston A. Cox, James L. Schmidt, Simon M. Kao, and Jackson Y. Read. "Real Time Knowledge-Based Systems." AI Magazine 9, no. 1 (Spring 1988): 27-45.

Lager, Darrel L., Hal R. Brand, William J. Maurer, Fred Coffield, and Frank Chambers. "MAESTRO - A Model and Expert System Tuning Resource for Operators." Nuclear Instruments and Methods in Physics Research A 293 (1990): 480-485.

Larner, Daniel L. "A Distributed, Operating System Based, Blackboard Architecture for Real-Time Control." In The Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Charleston, South Carolina, July 15-18, 1990, by the Association for Computing Machinery. New York: ACM, 1990, 99-108.

Laycak, John F. "Computer Control for a Particle Beam Accelerator." IEEE Transactions on Nuclear Science 36, no. 5 (October 1989): 1616-1618.

Lesser, Victor R., and Daniel D. Corkill. "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks." In Blackboard Systems, ed. Robert S. Englemore and Anthony Morgan, 353-386. Reading, MA: Addison-Wesley, 1988.

Lesser, Victor R., Jasmina Pavlin, and Edmund Durfee. "Approximate Processing in Real-Time Problem Solving." AI Magazine 9, no. 1 (Spring 1988), 49-61.

Lesser, Victor. "Guest Editor Introduction." The Journal of Real-Time Systems 2 (May 1990): 5-6.

Lewis, Steven M. "Tokenized Rule Based System." Journal of Forth Application and Research 4, no. 1 (1986): 29-45.

Lind, P. C., W. F. S. Poehlman, and J. W. Stark. "Implementation Considerations for PACES: the KN3000 Particle Accelerator Control Expert System." In Proc. Third Symposium on Expert Systems in the Dept. of National Defence, May 2-3, 1991. Kingston, Ontario: Royal Military College, 1991, 17-36.

Lind, P. C., and W. F. S. Poehlman. "Design of an Expert System-based Real-time Control System for a Van de Graaff Particle Accelerator." In Proc. AIENG '92: Applications of Artificial Intelligence in Engineering, July 14-17, 1992, edited by D. E. Grierson, G. Rzevski, and R. A. Adey. New York: Elsevier, 1992.

Lind, P. C., W. F. S. Poehlman, and J. W. Stark. "The KN-3000 Particle Accelerator Control Expert System (PACES)." IEEE Transactions on Nuclear Science 40, no. 6 (December 1993): 2030-2036.

Lind, Peter C., and W. F. S. Poehlman. "Artificial Intelligence-Based Control of a Particle Accelerator's Analyzing Magnet." In Proc. Sixth Symposium on Expert Systems in the Dept. of National Defence, May 5-6, 1994. Kingston, Ontario: Royal Military College, 1994.

Lingarkar, Ravi. "Automation of the Low-energy Steerers for the FN Tandem Accelerator." M.Sc. dissertation, McMaster University, Hamilton, Ontario, 1988.

Lutz, J. R., and J. C. Marsaudon. "Modern tandem control systems." Nuclear Instruments and Methods in Physics Research A 352 (1994): 113-125.

Matheus, Christopher J. "The Internals of FORPS: A FORth-based Production System." Journal of Forth Application and Research 4, no. 1 (1986): 7-27.

McKay, John, C. E. L. Gingell, and John Barns. "Control of Tandem Accelerator Systems Using a Personal Computer." In Proc. XXth Symposium of North Eastern Accelerator Personnel, November 3-6, 1986. Teaneck, NJ: World Scientific Pub. Co., 1986, 245-258.

Mercaldo, Matt. "Fuzzilogy 101." The Computer Journal no. 55 (May-June 1992): 3-5.

Morgenstern, Leonard. "YAOOP: 'Yet Another' Object-Oriented Program in Forth." In 1990 FORML Conference Proceedings, by the Forth Interest Group. San Jose, CA: Forth Interest Group, 1990. 78-88.

Morizet-Mahoudeaux, Pierre. "On-Board and Real-Time Expert Control." IEEE Expert 11, no. 4 (August 1996): 71-81.

Motorola Corporation. "Applications of the MOC3011 Triac Driver." Application Note AN-780A. Phoenix, Arizona: Motorola, 1982.

Motorola Corporation. "MC6854 Advanced Data-Link Controller." Chap. in Motorola 8-Bit Microprocessor and Peripheral Data. Austin, Texas: Motorola, 1983.

Motorola Corporation. "Fuzzy Logic and Embedded Control." Application note. Phoenix, Arizona: Motorola, 1992.

Munakata, Toshinori, and Yashvant Jani. "Fuzzy Systems: An Overview." Communications of the ACM 37, no. 3 (March 1994): 69-76.

Murphy, Thomas. "AI Apprentice: Speaking in Tongues." AI Expert 8, no. 4 (April 1993): 13-16.

Naito, Norio, Akira Sakuma, Kei Shigeno, and Nobuyuki Mori. "A Real-Time Expert System for Nuclear Power Plant Failure Diagnosis and Operational Guide." Nuclear Technology 79 (December 1987): 284-296.

Nökel, K., and H. Lamberti. "Temporally Distributed Systems in a Diagnostic Application." Artificial Intelligence in Engineering 6, no. 4 (1991): 196-204.

Ostroff, Jonathan S. Temporal Logic for Real-Time Systems. Taunton, Somerset, England: Research Studies Press Ltd., 1989.

Park, Jack. "Toward the Development of a Real-Time Expert System." Journal of Forth Application and Research 4, no. 2 (1986): 133-154.

Pelc, Stephen. "PROCIC - A Process Computer for Computationally Intensive Control." in Proceedings of the 1992 Rochester Forth Conference, by the Institute for Applied Forth Research. Rochester, NY: Institute for Applied Forth Research, 1992, 69-70.

Perkins, W. A., and A. Austin. "Adding Temporal Reasoning to Expert-System-Building Environments." IEEE Expert 5, no. 1 (February 1990): 23-30.

Perreard, S., and E. Wildner. "Conditioning of high voltage radio frequency cavities by using fuzzy logic in connection with rule based programming," Nuclear Instruments and Methods in Physics Research A 352 (1994): 336-338.

Poehlman, W. F. S., R. J. Pollock, and R. A. McNaught. "Multi-Processor Control of the 100 KeV McMaster Mark IV Sputter Ion Source." IEEE Transactions on Nuclear Science 32, no. 5 (October 1985): 2083-2085.

Poehlman, W. F. S., and J. W. Stark. "Integrating Knowledge-Based Systems into Operations at the McMaster University FN Tandem Accelerator Laboratory." IEEE Transactions on Nuclear Science 36, no. 5 (October 1989): 1494-1498.

Poehlman, W. F. S., Wm. Garland, and J. Stark. "Design Principles Employed in Aid of Real-time Expert Control Systems Development." In Proc. Fourth Artifical Intelligence Symposium, Sept. 20-21, 1991, by the University of New Brunswick. Fredericton, New Brunswick: Univeristy of New Brunswick, 1991, 109-119.

Pollock, Robert J. "Computer Controlled Ion Injection Source Subsystem." M.Eng. dissertation, McMaster University, 1985.

Possnert, G., I. Carlsson, and G. Widman. "The EN tandem van de Graaff accelerator." In TSL Progress Report, July 1987 - December 1991, by The Svedberg Laboratory. Uppsala, Sweden: The Svedberg Laboratory, 1992, 47-48.

Pountain, Dick. Object-Oriented Forth. London: Academic Press, 1989.

Rather, Elizabeth. "EXPRESS: A Forth-based Process Control Software Product." In Proceedings of the 1993 Rochester Forth Conference, by the Institute for Applied Forth Research. Rochester, NY: Institute for Applied Forth Research, 1993. 87-91.

Redington, Dana. "A Forth Oriented Real-Time Expert System for Sleep Staging: a FORTES Polysomnographer." Journal of Forth Application and Research 4, no. 1 (1986): 47-56.

Reynolds, Dave. "MUSE: A Toolkit for Embedded, Real-time AI." In Blackboard Systems, ed. Robert S. Englemore and Anthony Morgan, 519-532. Reading, MA: Addison-Wesley, 1988.

Rodriguez, Bradford J. "PatternForth: A Pattern-Matching Language for Real-Time Control." M.S. dissertation, Bradley University, Peoria, Illinois, 1989.

Rodriguez, Bradford J. "A BNF Parser in Forth." ACM SIGForth Newsletter 2, no. 2 (December 1990): 13-18.

Rodriguez, Bradford J., and W. F. S. Poehlman. "A Survey of Object-Oriented Forths." ACM SIGPLAN Notices 31, no. 4 (April 1996): 39-42.

Rosen, Evan. "High Speed, Low Memory Consumption Structures." in Proceedings of the 1982 FORML Conference, by the Forth Interest Group. San Carlos, CA: Forth Interest Group, 1982. 191-196.

Sammut, Claude, and Donald Michie. "Controlling a Black-Box Simulation of a Spacecraft." AI Magazine 12, no. 1 (Spring 1991): 56-63.

Sanderson, Dean, and Adam Shackleford. "A Diagnostic Expert System in polyForth." In Proceedings of the 1986 FORML Conference, by the Forth Interest Group. San Jose, CA: Forth Interest Group, 1986. 281-289.

Sedgewick, Robert. Algorithms. Reading, MA: Addison-Wesley, 1983.

Schultz, David E., and Pamela A. Brown. "The Development of an Expert System to Tune a Beam Line." Nuclear Instruments and Methods in Physics Research A 293 (1990): 486-490.

Siddall, James N. Expert Systems for Engineers. New York: Marcel Dekker, 1990.

Sibigtroth, James M. "Creating Fuzzy Micros." Embedded Systems Programming 4, no. 12 (December 1991): 20-34.

Skapura, David M. "A Faster Embedded Inference Engine." AI Expert 4, no. 11 (November 1989): 42-49.

Sperry, Tyler. "Gaining a Fuzzy Perspective." Embedded Systems Programming 6, no. 1 (January 1993): 24-28.

Stark, J. W., and W. F. S. Poehlman. "Introduction to the Operation of the McMaster FN Tandem Using a Knowledge-Based Real-Time System." In Proc. 22nd Symposium of North Eastern Accelerator Personnel, October 24-27, 1988. Teaneck, NJ: World Scientific Pub. Co., 1988, 119-129.

Stark, J. W., W. F. S. Poehlman, T. Cousins, F. G. Strain, and P. C. Lind. "Development of a Knowledge-based Control System for a Model KN Van de Graaff Accelerator." In Proc. XXVth Symposium of North Eastern Accelerator Personnel, Oct. 24-28, 1991. Teaneck, NJ: World Scientific Pub. Co., 1992, 85-90.

Strole, Norman C. "The IBM Token-Ring Network -- A Functional Overview." IEEE Network 1, no. 1 (January 1987): 23-30.

Suzuki, Junzo, Naomichi Sueda, Yasuo Gotoh, and Akimoto Kamiya. "Plant Control Expert System Coping with Unforeseen Events -- Model-based Reasoning Using Fuzzy Qualitative Reasoning." In The Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Charleston, South Carolina, July 15-18, 1990, by the Association for Computing Machinery. New York: ACM, 1990, 431-439.

Tam, Patrick Wai Zee. "A Knowledge-Based Operator's Companion for the FN Tandem Accelerator." M.Sc. dissertation, McMaster University, 1989.

Tanenbaum, Andrew S. Computer Networks. Englewood Cliffs, NJ: Prentice-Hall, 1989.

Tanenbaum, Andrew S. Distributed Operating Systems. Englewood Cliffs, NJ: Prentice-Hall, 1995.

Tétreault, Mario, Bernard Marcos, and Jean Lapointe. "Temporal duration reasoning in qualitative simulation." Artificial Intelligence in Engineering 7 (1992): 185-197.

Townsend, Carl, and Dennis Feucht. Designing and Programming Personal Expert Systems. Blue Ridge Summit, PA: Tab Books, 1986.

Valenzano, A., C. Demartini, and L. Ciminiera. MAP and TOP Communications: Standards and Applications. Wokingham, England: Addison-Wesley, 1992.

van Liempd, Gidi, Hugo Velthuijsen, and Adriana Florescu. "Blondie-III." IEEE Expert 5, no. 4 (August 1990): 48-55.

Weehuizen, H. F., R. K. Fisch, E. J. S. Franklin, M. E. Hogan, I. H. Kohler, J. P. Rogers, and P. J. Theron. "A Distributed Control System for the NAC Cyclotrons." IEEE Transactions on Nuclear Science 39, no. 2 (April 1992): 205-209.

Wong, J. X. "Low Energy Steerer Control System for the FN Tandem Accelerator." M.Sc. dissertation, McMaster University, 1986.

Wong, J., and W. F. S. Poehlman. "Computer Automated Low Energy Steering Control for Dynamic Optimization of Ion Beam Intensity." In Proc. XXth Symposium of North Eastern Accelerator Personnel, November 3-6, 1986. Teaneck, NJ: World Scientific Pub. Co., 1986, 231-244.

Wright, M. Lattimer, Milton W. Green, Gudrun Fiegl, and Perry F. Cross. "An Expert System for Real-Time Control." IEEE Software 3, no. 2 (March 1986): 16-24.

Zanconato, Roberto. "BLOBS - An Object-Oriented Blackboard System Framework for Reasoning in Time." In Blackboard Systems, ed. Robert S. Englemore and Anthony Morgan, 335-345. Reading, MA: Addison-Wesley, 1988.

Zhang, Bing, and Edward Grant. "Experiments in Adaptive Rule-Based Control." In The Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Charleston, South Carolina, July 15-18, 1990, by the Association for Computing Machinery. New York: ACM, 1990, 563-568.

Zilog Corporation. "Z8530 SCC Serial Communications Controller." Chap. in Zilog 1981 Data Book. Cupertino, CA: Zilog, 1981.

Zsoter, Andras. "An Assembly Programmer's Approach to Object-Oriented Forth." Forth Dimensions 16, no. 6 (March-April 1995): 11-17.

Zsoter, Andras. "Does Late Binding Have To Be Slow?" Forth Dimensions 18, no. 1 (May-June 1996): 31-35.

APPENDIX A. INSTRUMENTING THE FN ACCELERATOR

A.1 Passive, Fail-Safe, Retrofit Interface Techniques

The McMaster Tandem Accelerator first went "on-line" in 1968. Its control system design dates from that era, and, unlike a modern accelerator, few of its instruments have been designed for attachment to computers. Since it would not be feasible to replace all of the control hardware for this project, some means to "retrofit" a computer interface to the existing electronics is needed. These retrofits must be:

a) *Optional*. It must always be possible to run the accelerator manually, in the accustomed fashion. The interfaces must be removable by computer command or the throw of a switch.

b) *Passive*. Even when the computer is controlling the accelerator, the human operator must be able to monitor its function. The interfaces must not interfere with the existing displays.

c) *Fail-Safe*. If the computer system should fail -- even if power to the interfaces is lost -- the interfaces must fail to a "safe" state, e.g., blocking the accelerator beam. Ideally they should fail to the "manual operation" state.

d) *Economical*. The interfaces should cost little, implying a minimum of new hardware, and the fewest number of changes to the existing hardware. Economies of scale should be exploited where possible.

Four circuits have been devised to meet these goals.

A.2 Isolated Meter Repeater

Sixty different analog quantities must be observed by the operator of the accelerator -- be it computer or human. These range from terminal megavolts to corona points position to vacuum pressure. The wide variety of sensors and metering circuits share no common design, and would have to be redesigned individually.

But there is one commonality: all sixty quantities are displayed for the operator on ordinary panel

meters. Thus a single circuit which could monitor the indication on a panel meter could interface all sixty measuring devices. Panel meters are frequently installed at high-voltage points within vacuum-tube circuits, so the "repeater" output must be electrically isolated from the meter.

Circuit Description

The Isolated Meter Repeater, Figure A-1, is a multi-range voltmeter/ammeter. The core of this circuit is an analog optoisolator, ISO1. The feedback of op amp U2A acts to make the ISO1B diode current equal to that of the ISO1A.[1] Thus a variable current can be measured with no electrical connection. The "primary" diode is biased into its approximately-linear region (Hewlett-Packard, 1985) by adding 1.25 mA, derived from an inexpensive precision voltage regulator D3. An equal current is subtracted in the "secondary" stage by the current mirror Q1-Q2.

A "current follower" op amp U1 provides drive for the optoisolator and buffers the input signal. This device may operates at a circuit potential of several hundred volts, so it must be provided with an isolated power source. Oscillator U3 generates 40 kHz AC, which is *capacitively* coupled to a voltage doubler. This scheme was thought to be less expensive than the more common transformer coupling. Suitable transformers are hard to find, whereas capacitors blocking 600 volts are readily available. The rectified signal is filtered and regulated to provide +5V for the TLC271 "micropower" op amp, and -2.5V for the bias.

Currents of plus or minus 1 mA can be measured. The bias current limits the negative input range, since Ibias must exceed -Iin. The isolated power source limits the positive range, since Ibias+Iin must not exceed the available current. This current is converted to a 0-5V output by U2B. R9 can be selected to provide a full-scale "movement" of 50 uA, 1 mA, or any current in between. Half-wave rectifier D5-D6 allows measurement of AC current and voltage; in this application R9 is scaled by 2.21 to convert half-wave average to RMS voltage (assuming a sine wave input).

Use of a "multiplier" resistor R1 converts the ammeter to a voltmeter. With a 50 uA movement,

---

1. C12 compensates for the propagation delay through the isolator; without it the circuit will oscillate.

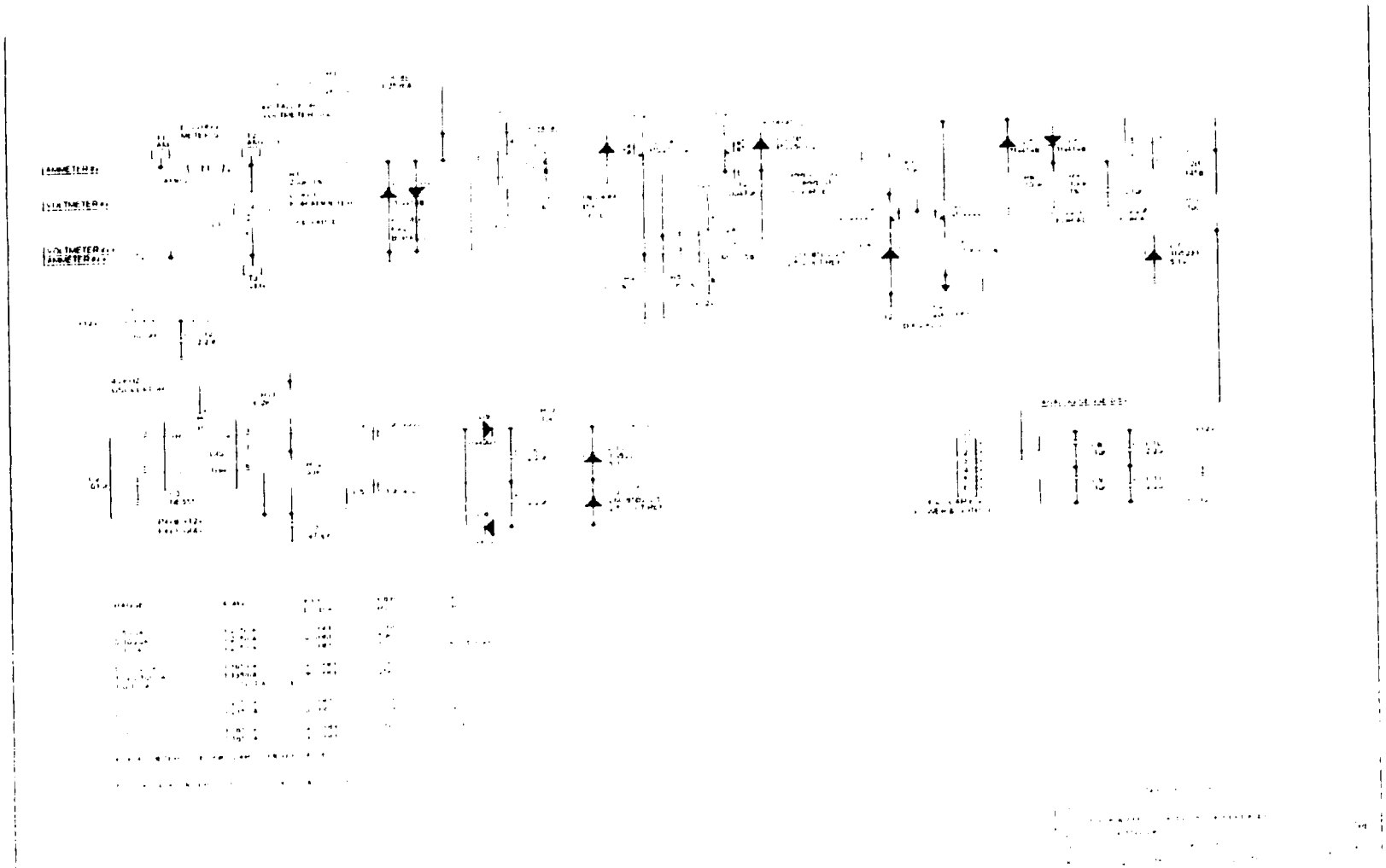


Figure A-1. Isolated Meter Repeater

the input impedance is 20,000 ohms per volt.

When inserted as an ammeter, the circuit should not interfere with the measured current flow. When powered, op amp U1 will be a "virtual short" (maintaining 0 volts across its input). When unpowered, D1-D2 provide a path for current to flow, ensuring a maximum voltage drop of 0.7 volts. These also protect the op amp input, without interfering with its operation.

The completed circuit is fabricated on a 2.25" x 2.25" printed circuit board, which can be mounted directly on the back of a standard panel meter.

Experience and Recommendations

Experience with the circuit has been generally good; it is reliable, responsive, and sufficiently linear. The board is versatile enough to be used unmodified for most of the AC and DC meters in the FN accelerator. It has proven easy to modify and customize in the field.

One customization deserves note. The FN Terminal Voltmeter requires a 2 volt DC meter with a high (megohms) input impedance. It was possible to cut and jumper the board to convert U1 to a voltage follower, with a 5 megohm input resistor. In retrospect, it would have been better to design the primary with a dual CMOS op amp (TLC272), allowing a stage for isolation and gain.

Experience has shown that the bias adjustment R2 needs a range of several hundred uA. (Even then, R7 or R3 must occasionally be replaced to match the bias currents). This presents no problem with a 1 mA movement, but when the full scale deflection is 50 uA, it is difficult to "zero" the output. A current gain stage in the primary would also solve this problem.

Optoisolators have a temperature-sensitive transfer characteristic. Using a matched dual isolator in a single package (NEC PS2501-2) is essential, but if one side of the package is warmed (even with a fingertip), the output begins to drift.[2] The zero adjustment (R2) should be made when the circuit has stabilized at operating temperature.

Some FN panel milliammeters show a significant drop when the power is removed from the

2. In private communication, Alan McIlwain of Atomic Energy of Canada Ltd. reports that adding a heat sink to the optoisolator improves this problem.

meter repeater. The 0.7 volt drop across D1-D2 is excessive in some applications. An earlier design used a relay to bypass the meter repeater when unpowered; either this more expensive alternative, or a more sophisticated electronic bypass, will be needed for a "mature" board design.

The greatest problems occurred with the capacitively isolated power supply. If the measured circuit and the computer system are truly isolated, there is no problem; but if they have any common ground (as is frequently the case), a "ground loop" can be created between the output and the input. The end result is that 40 kHz AC is coupled into the secondary amplifier. It was possible to correct this in most cases by judiciously inserting input resistors, but it is clear that capacitive coupling is inadequate, and it should be abandoned in favor of transformer coupling.

In a few cases, the meter repeater picked up 60 Hz AC from the FN electronics. The resultant ripple on the output appears as noise on the measured quantity. This was generally solved by putting bypass capacitors on the repeater input, and by increasing C1 (e.g., to 10 uF) to create a long time-constant RC filter. A redesign of the output stage would allow a more effective filter.

A.3 Solid State Variac

The principal control device used in the FN accelerator is a "Variac" autotransformer, adjustable from 0 to 120 VAC and providing a few amps of current. This device is not readily adapted to computer control. Although it is possible to turn a Variac shaft with a microprocessor-controlled stepper motor (Lind and Poehlman 1992), a non-mechanical solution is preferable. This involves replacing the Variac with an electronic device which can be controlled either manually or by computer.

Circuit Description

The "solid state Variac," Figure A-2, uses a Triac device to provide a variable AC output. By switching the Triac on partway through the AC half-cycle, the effective (RMS) voltage can be reduced from 120 VAC.[3]

The Triac Q2 must be "fired" (triggered) at some point from 0 to 180 degrees into each half cycle

3. The peak output voltage remains unaffected over half of the adjustment range.

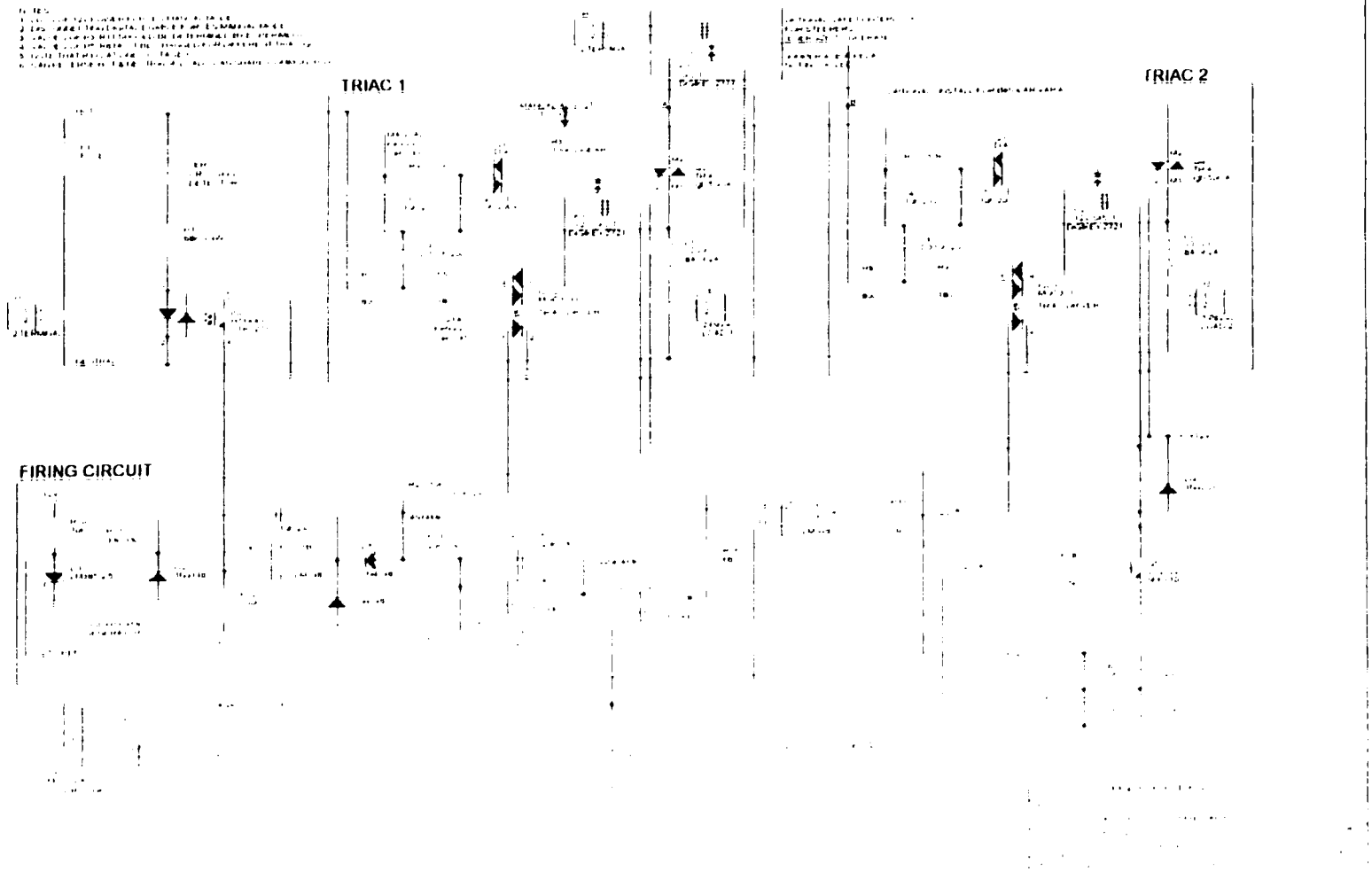


Figure A-2. Solid State Variac

of the AC line. If fired at 0 degrees (the start of the half cycle), the Triac is continuously on, producing full output. If fired at 180 degrees (the end of the half cycle), the Triac is continuously off, for zero output. This firing signal is generated as in the conventional lamp dimmer (ARRL, 1992), by a variable resistor and "Diac" trigger diode (R3 and Q1 in the figure). This provides a manual control.

The computer's control signal is a DC voltage from 0 to 5 volts. This is compared with a sawtooth wave synchronized with the AC half-cycle. U1B generates a positive-going ramp, it is reset to 0 volts when the AC wave crosses zero and ISO1 turns off. U1A produces a negative going ramp, reset to +5V at the zero crossing, as shown in Figure A-3. When this ramp falls lower than the input voltage, comparator U1C fires the Triac optocoupler ISO2, which in turn fires the Triac Q2 (Motorola 1982). If the control voltage is raised, the Triac fires earlier, yielding a larger AC output.

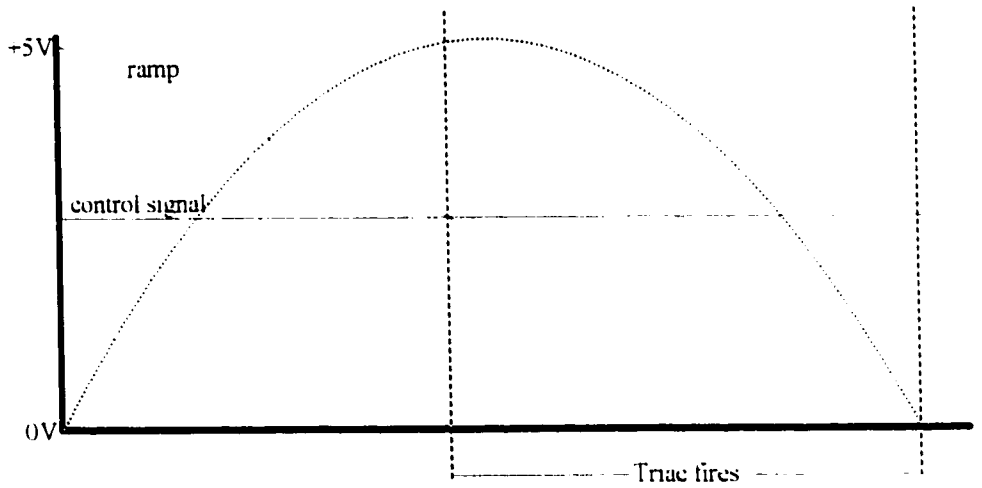


Figure A-3. Generation of Triac Firing Signal

When a Variac is used to control beam steerers, it can be connected so that when one steerer is adjusted to maximum voltage, the other is at minimum, and vice versa. In the solid state Variac, a second Triac is required. The same control signal is compared with the *positive*-going ramp to provide the firing signal for Triac Q4. When the control voltage is raised, the Triac fires later, yielding a smaller AC

output. Likewise, the manual control R3 operates in opposite directions for the two Triacs. When only a "unipolar" control is required, these components may be omitted.

Relays K1 and K2 are actuated by a digital output from the computer to transfer from the manual control R3 to the computer-controlled firing circuit. If the computer is disconnected or loses power, the relays select the manual control, which requires only the AC line for operating power. Relay K3 is an optional safety interlock; if installed, it must be energized for the steerers to operate. If de-energized, the steerers will be powered to full deflection (one switched off, the other at full voltage).

Experience and Recommendations

Aside from a few heater elements (e.g. in the cesium boiler), the most common use of the Variac is to adjust the output of a high-voltage power supply. A typical supply is an encapsulated module containing a transformer, a rectifier, and a filter capacitor. The output voltage of this supply, when lightly loaded, will follow the peak (and not the RMS average) of the AC input voltage. Early experimentation indicated that good control of the output could be achieved with a Triac circuit. Upon installation in the FN, however, two difficulties were encountered.

First, many of the high voltage supplies are normally operated with a low AC input, i.e., at the low end of the control range. The manual control circuit requires a minimum voltage (typically 35V) to fire the Diac; below this voltage the Triac will be cut off. Second, a hysteresis effect was frequently observed with the manual control circuit: the control had to be advanced excessively to fire the Triac, but then could be reduced below the firing level. Both of these problems were solved when they occurred by putting a resistance[4] in series with the load, such that the usual high-voltage settings corresponded to the middle of the Triac control range. This problem would not be expected with the computer firing circuit, which has exact control of phase angle, but the requirement for "fail-safe" manual control takes precedence.

Another difficulty was encountered with the "bipolar" Triacs used to control the steerers. The

4. A 40 watt incandescent lamp was found to be quite effective in many cases.

effective steerer voltage is the difference between two HV power supplies, controlled by two Triacs acting in opposition. The effective response curve of a Triac is nonlinear, especially when controlled by a variable resistance and Diac. The difference output of two opposed Triacs[5] exhibits a reversal near zero, i.e., a control reversal at the middle of the control range (the most common setting). This problem was not observed with the computer firing circuit. Time constraints precluded more experimentation with the manual firing circuit, but it should be possible to alter the response curve such that this control reversal no longer occurs.

A.4 Repeating Picoammeter

Many of the adjustments of the FN accelerator are made with reference to beam current, as measured at an inserted Faraday cup. The interface must perform two functions: first, convert an input current (in the range of tens of picoamperes to tens of microamperes) to a DC voltage that can be measured by the computer. Second, while doing so, permit the current to be measured with the existing panel meter. This could have been done with a meter repeater (section A.2), except that there is no way for the computer to read or change the range switch of the Keithley Model 414 picoammeter.

Circuit Description

Two electronic picoammeters cannot be simply connected in series to read the same current. Instead, the circuit of Figure A-4 operates by measuring the input current, and then generating an equal (but independent) output current. This output current can then be measured with the existing Model 414 meter.

U1 is a TLC271, a low-cost CMOS op amp, with a 1 pA input bias current. Relays K3 and K5 select feedback resistors of 100, 1, or .01 megohm, which respectively will convert 10 nA, 1 uA, or 100 uA to an output voltage of 1.0 V. Resistor K8 switches a 1/10 voltage divider into the feedback loop, multiplying the output of U1 by 10. In this case a current of 1 nA, 100 nA, or 10 uA will cause a "full scale" output of 1.0V from U1. This voltage will have the opposite polarity from the input current. Op

5. Two Triacs on an imperfect AC feed can also interact, since each can distort the AC wave.

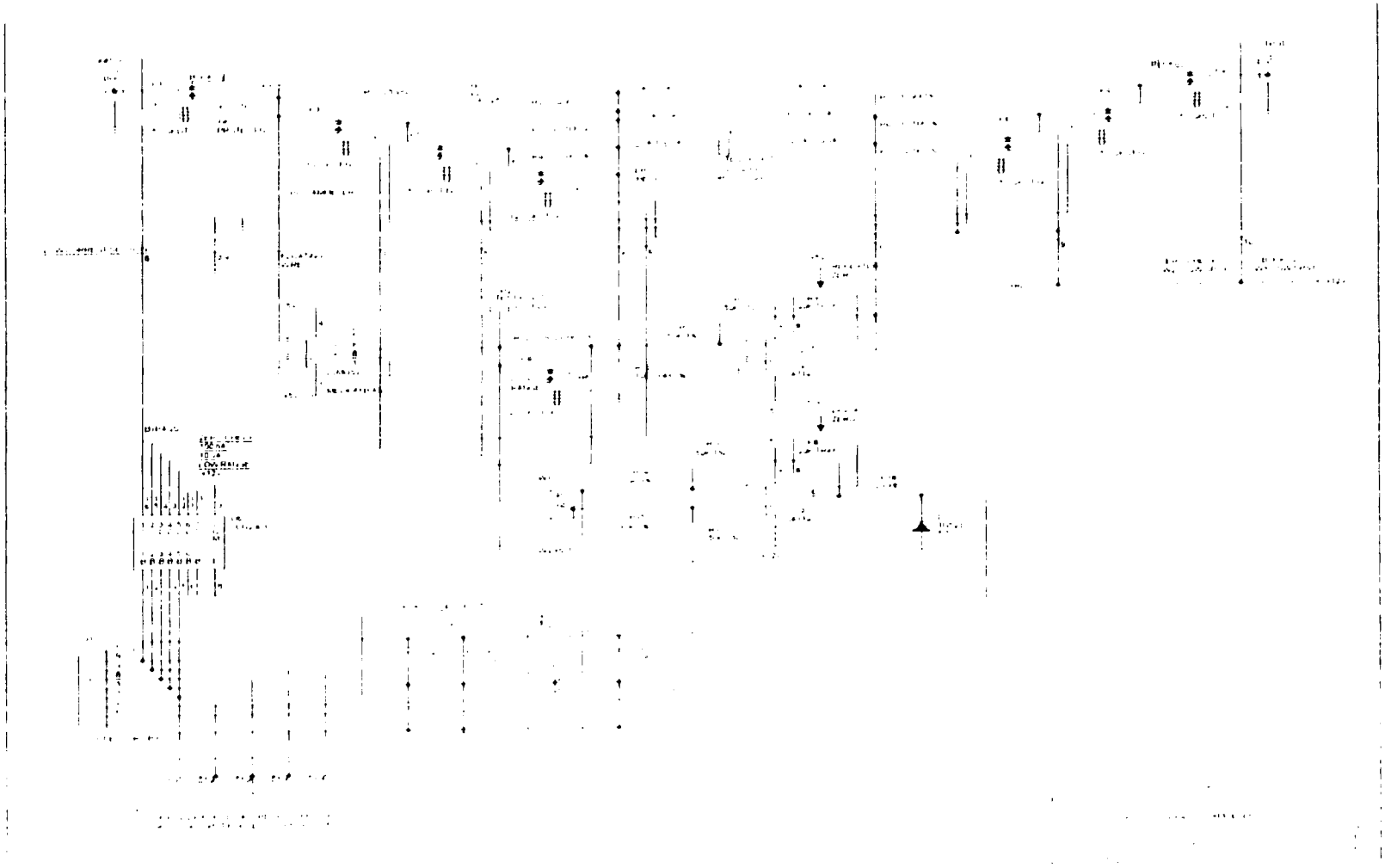


Figure A-4. Inline Picoammeter

amp U2 buffers this voltage and re-inverts the polarity. This voltage is then applied to a 100, 1, or .01 megohm resistance, selected by K4 and K6, to produce an output current equal to the input current.[6]

Meanwhile, op amp U3 amplifies the 1.0V DC to 5 volts DC, the full scale input of the A/D converter. The polarity can be made switch selectable (SW1) because each installation of this meter will only encounter one polarity of current (negative in the low energy section of the accelerator, positive in the high energy section).

In addition to reading this 5V analog signal, the computer has five TTL control outputs. Three select the range, controlling relays K2-K6 and K8 as described above. Another causes K7 to short the feedback resistor of U1, forcing an output of zero. This can be used by the computer to determine the zero offset of its analog input subsystem. The final control activates K1 and K2 to completely bypass the repeating picoammeter, connecting the output directly to the input. This is the "fail safe" default if the computer is disconnected or loses power. The zero check and the bypass can also be manually operated, by SW2 and SW3.

Experience and Recommendations

Unfortunately, accelerator operations were terminated before this unit could be put into service. Both the input (U1) and output (U2) circuits were tested in the laboratory, and both worked well.[7] There are two areas of potential concern.

First is current leakage, the curse of picoammeters. To minimize this, the low current section was isolated on a separate PC board, with only one low-current connection (U1 input) to the main electronics board. All other connections to the low-current board (relay control and U1 output) were far removed from the low-current traces. The effectiveness of these measures remains to be seen.

Second, it is likely that relay switching will induce transient currents into the input. Relays are unavoidable, since electronic switches cannot match the relays' 10^{10} ohm "off" resistance. The relays are

6. This takes advantage of the fact that the input resistance of the Model 414 picoammeter is effectively zero (much smaller than R5-R7).

7. Tests were conducted with a 10 megohm feedback resistor, as 100 megohm resistors were not on hand.

controlled with DC, so current should only be induced briefly when a relay is switched on or off. The computer can allow for this on the infrequent occasions when the meter range must be changed.

A.5 Beam Profile Monitor Tap

The other principal beam measurement uses a "beam profile monitor." This is a mechanism which moves a thin wire through the beam horizontally and vertically.[8] As the wire passes through the beam, a current proportional to the intersected beam is induced. On an oscilloscope this current appears as two "cross-sections" of the beam.

Circuit Description

The beam profile monitor (BPM) is controlled by two switch inputs, one to turn on the beam scanning process (by turning on a motor), and the other to switch in an optional x10 amplifier at the sensor head. The primary output is a voltage proportional to intersected current. A second output provides a synchronization pulse at the start of the scan. Scanning is continuous at about 17 scans per second.

The computer interface for this is quite simple, as shown in Figure A-5. The "probe" signal is simply buffered. The "index" signal is converted to a digital pulse by a comparator, using a circuit similar to that of the BPM control unit. These are noninvasive "taps" which do not affect the normal display operation.

The "motor" control output actuates a relay driver in U1, turning on K1, which switches the motor power on and disables the manual control. The "gain" control output actuates Q1, outputting 12 volts to turn on a relay in the BPM sensor head. Thus, both of the output controls are a "logical OR" with the manual controls: either unit can turn on the BPM; both must agree to turn it off. Since the x10 amplifier will affect the observed signal, the computer is provided with a "gain sense" input to detect when it is manually activated.

The project was terminated before this unit could be field tested.

---

8. A single cleverly bent wire, rotated by a motor, traverses the beam in both directions.

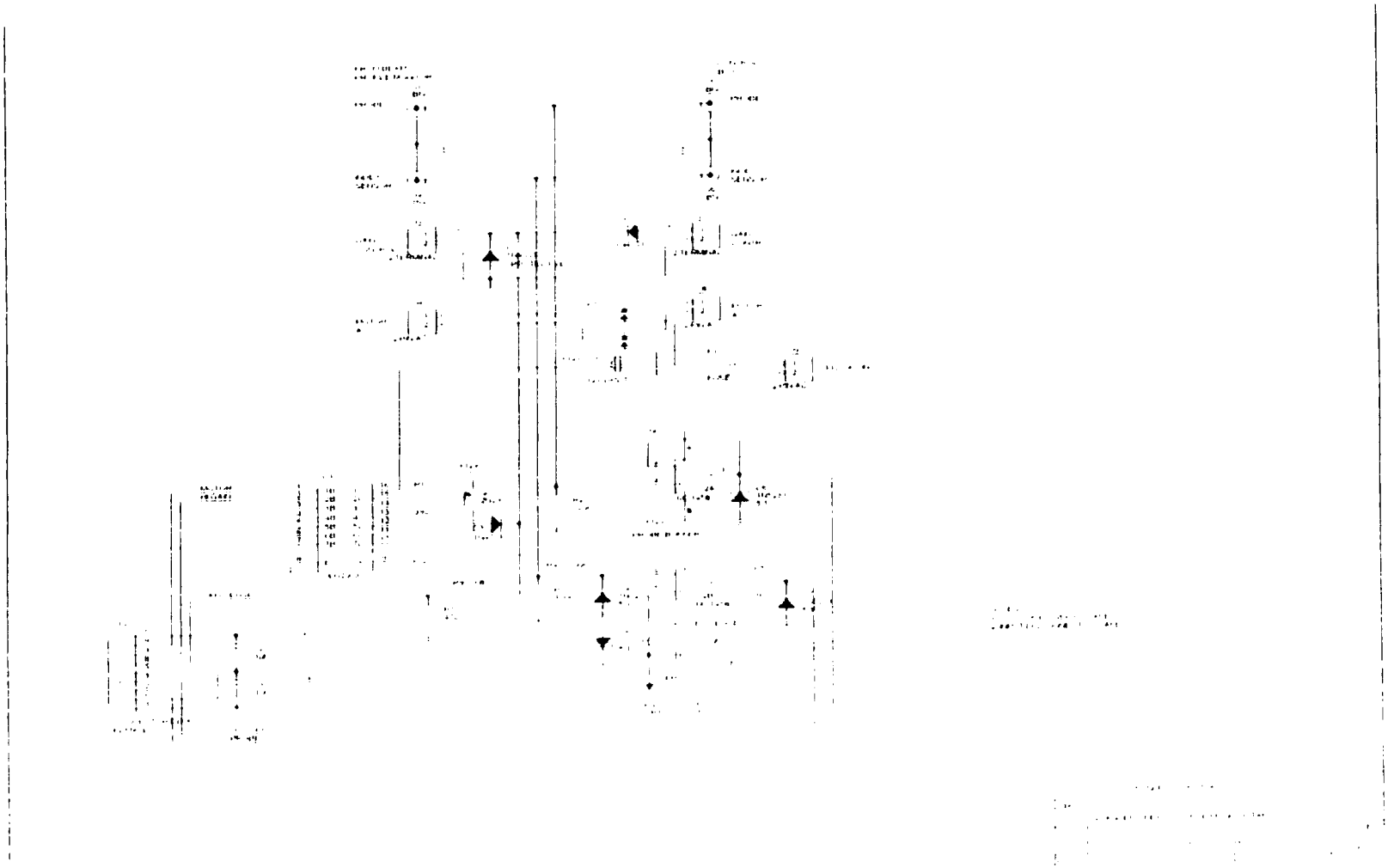


Figure A-5. Beam Profile Monitor Tap

A.6 Programmable Resistor

The FN accelerator employs a few "resistively programmed" power supplies. The output voltage (or current) of such a supply is set by an external resistance. Either a fixed or variable resistor may be used. This resistor is usually in the feedback loop of an amplifier, so it must be isolated from ground.

Circuit Description

Figure A-7 illustrates an optoisolated, voltage-controlled resistor. The key component in this circuit is an H11F1 optoisolated FET, which, like a conventional FET, acts as a variable resistance. The response curve of this device is decidedly nonlinear; however, experimentation has revealed that an H11F1 in parallel with a fixed resistor has a quite linear characteristic. Figure A-6 shows two examples: a 10K ohm resistor in parallel with the FET, and a 680 ohm parallel resistor. In each case the response "saturates" at 100 ohms resistance, with an input current of about 9 mA.

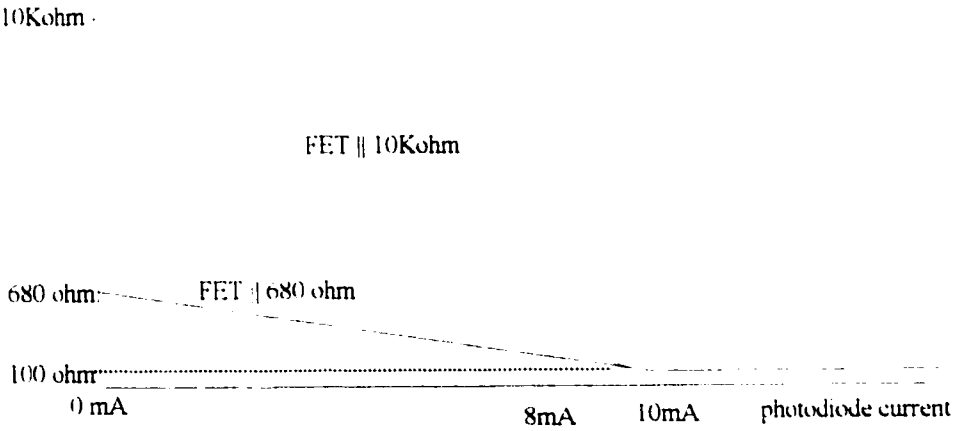


Figure A-6. Programmable Resistor Characteristic

U1 converts the 0 to 5V DC control signal from the computer, to a 0-10 mA current (the useful range) through optoisolator ISO1. Relay K1 transfers control from the existing manual potentiometer to the opto-FET. If the computer is disconnected or switched off, manual control is the default.

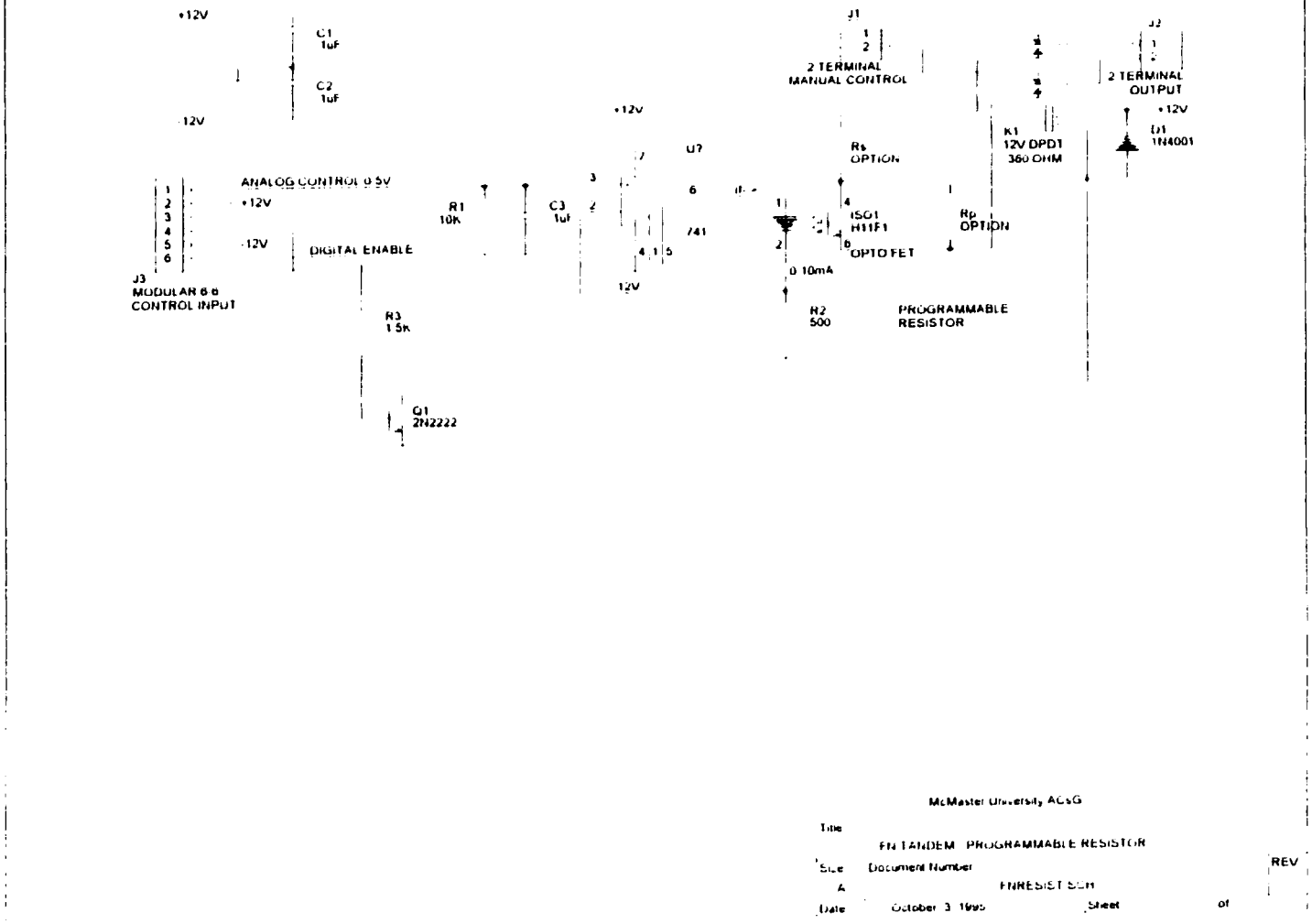


Figure A-7. Programmable Resistor

APPENDIX B. ASYNCHRONOUS TOKEN RING COMMUNICATIONS

B.1 Design Criteria for the Local Area Network

A fully distributed expert system implies some means for the individual experts to communicate -- in brief, a local area network. For this research, the network must meet the following goals:

a) *Inexpensive*. It defeats the purpose of using inexpensive microcontrollers, if the network interface is expensive or requires custom logic. Ideally, the network interface should require only the "standard" asynchronous serial port, and no external hardware.

b) *Fiber optic capable*. A particle accelerator is an "electrically hostile" environment. Also, it is desireable to place control microprocessors inside various high-voltage cages. A network which can be carried on fiber optic cable is needed.

c) *Open-ended*. The network should support a variable, and possibly large, number of stations.

d) *Peer-to-peer*. The reasoning model is a "society of experts," each of whom is free to consult with, or advise, any other. Thus peer-to-peer communication is essential.

e) *Broadcast*. Frequently, one station must advise many, or all, stations of a change in some fact. The network must allow broadcast messages from any station.

f) *Deterministic*. Deterministic protocols are preferred for real-time control, guaranteeing an upper bound on time for one station to access the net, and for a message to be sent. Maximimizing the utilization of this low cost (and presumably low speed) network is also desirable.

B.2 Physical Topology

Three physical topologies were considered for the local area network: bus, star, and ring.

The physical bus is widely used. It is open-ended and can be implemented on many embedded microcontrollers (Butler, 1991, 1992). Deterministic protocols exist which support peer-to-peer and broadcast messages. Its chief disadvantage is the difficulty of using fiber optic links. While fiber optics

have been implemented (Tanenbaum 1989; Valenzano, Demartini, and Ciminiera 1992), for bussed local area networks, these installations require specialized bus transceivers which increase the cost.

A star topology, on the other hand, uses exclusively point-to-point links and is therefore readily adapted to fiber optic cable. The remote stations can use common (and inexpensive) serial ports. The "hub" of the network is the bottleneck: it requires many serial ports, and this ultimately places a limit on expansion. The hub must also actively relay peer-to-peer and broadcast messages, and, without special hardware, this load increases with the number of serial ports.

Many advantages of physical rings have been stated by (Tanenbaum 1989):

> Among their many attractive features is the fact that a ring is not really a broadcast medium, but a collection of individual point-to-point links that happen to form a circle. Point-to-point links involve a well-understood and field-proven technology, and can run on twisted pair, coaxial cable, or fiber optics. Ring engineering is almost entirely digital, whereas [IEEE Standard] 802.3, for example, has a substantial analog component for collision detection. A ring is also fair and has a known upper bound on channel access.

A disadvantage frequently cited for physical rings is that, if one station on the ring fails, the entire ring fails. This, however, can be an advantage in a control system: for the proper operation of the accelerator, all stations must be functioning. If any station fails, safety requires that all stations enter their "safe" or shutdown mode.

The principal disadvantage of most ring networks is the requirement for specialized hardware. Inexpensive interfaces have long been available for rings such as the IBM Synchronous Data-Link Control (Intel, 1981; Motorola, 1983; Zilog, 1981). Unfortunately, SDLC, a "logical star," does not support peer-to-peer or broadcast messages.

The IBM Token Ring, standardized as IEEE 802.5 (IEEE, 1985; Strole, 1987), is more promising. This is a true peer-to-peer network in which ownership of the ring rotates among all of the stations on the ring. Both group and broadcast addresses are supported. The 802.5 protocol satisfies all of the functional requirements; unfortunately, it is a synchronous protocol requiring specialized interface hardware. The ideal would be a protocol like 802.5, using standard asynchronous serial ports.

B.3 The Token Ring Using Asynchronous Communications

Each computer in the Asynchronous Token Ring receives serial data from the "previous" station in the ring, and transmits to the "next" station in the ring. Thus each station requires only one serial port. At any time, one station is the "owner" (master) of the ring, entitled to put data on the ring, and responsible for removing this data from the ring. The other "listening" (slave) stations simply retransmit any data they receive. Serial data bytes are originated by the owner, handed around the ring by the slaves, and consumed when they return to the master. Each station in the ring introduces a delay of one byte period (rather than one bit period as in IEEE 802.5).

The ring owner transmits data in frames, using the format shown in Figure B-1. The beginning of each frame is uniquely identified by a START code (0FF hex). All listening stations interpret a START code, regardless of when received, to mean the beginning of a new frame. A frame can be aborted by simply sending a new frame.

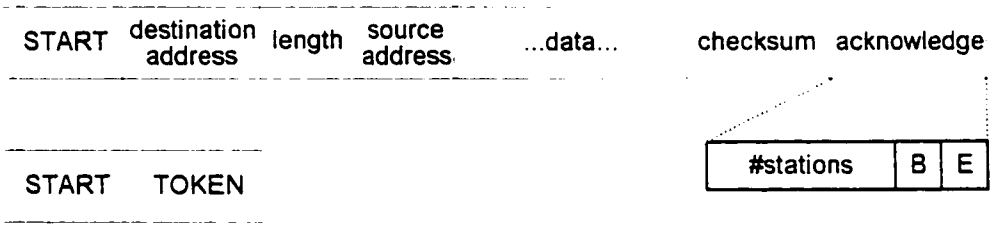


Figure B-1. Token Ring Frame Formats

The destination address byte may designate a specific recipient, or "broadcast" (00 hex). The station(s) designated must validate the frame, and if possible, receive it. Stations which do not recognize this address may return to passively "echoing" received bytes, while awaiting another START code. Nonaddressed stations need not buffer any of the frame: they can reject the frame immediately.

To simplify the logic of the software state machine, a message length byte is sent next. This is

the length of the *data* field, from 0 to 63 bytes. This is followed by the source address byte and the data field. During the transmission of the data field, reserved codes (such as START) are sent using "character stuffing" (Tanenbaum, 1989). The special code STUFF (0FD hex) is sent, followed by the data byte XOR 80h. This is transparent to the user and does not affect the transmitted length byte.

Following the data field, an FCS byte (Frame Check Sequence) is sent. This is the two's complement of the binary checksum over the destination, length, source, and data fields

The acknowledge byte is sent after the checksum so that it may be modified by listening stations as they pass the frame around the ring. Each station which recognizes the destination address, whether or not it copies the message, increments the 6-bit "#stations" field. When this byte completes its trip around the ring, the sender can discover missing stations, duplicate station addresses, or (in the case of broadcast messages) the total number of stations on the ring. The "B" bit is set by any recipient which was unable to copy the message due to lack of buffer space. The "E" bit is set by any recipient which detects a checksum error in the frame. Thus the sender can determine if the frame needs to be resent. In the case of broadcast messages, these bits indicate that at least one recipient failed to receive the frame

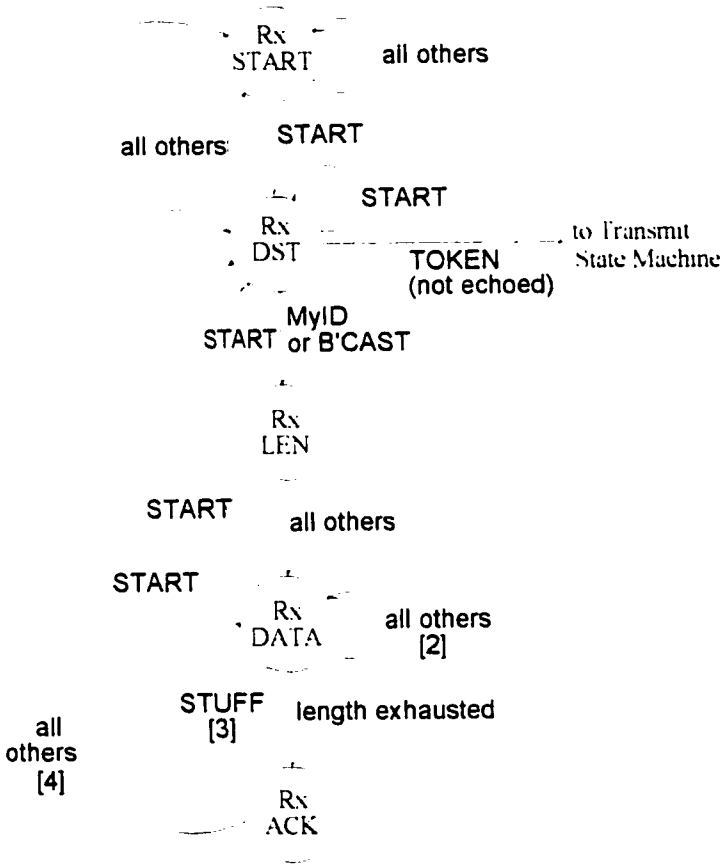
Since the ring is limited to 63 stations, the #stations field can only be incremented to 62, and the acknowledge byte can never be modified into any of the four reserved codes (0FC to 0FF hex)

When the ring owner is finished sending frames, it passes ownership to the next station in the ring with the two-byte sequence START TOKEN (0FF, 0FE hex). Using two bytes reduces the likelihood that a bit error will generate a spurious token. The station receiving the token may then assume ownership and begin transmitting frames, or, if it has no frames to transmit, will pass the token to its successor. When the ring is idle, the sequence START TOKEN continuously circulates, awaiting capture by the first station with data to send.

The Asynchronous Token Ring is managed by an interrupt-driven state machine in each processor. This software performs the functions of the IEEE Medium Access Control (MAC) sublayer Unlike the 802.5 system, the failure of any processor will halt the flow of data around the ring. This was

deemed an acceptable compromise in order to use low-cost microprocessor hardware for the stations.

Figure B-2 illustrates the state machine for a listening station. All received bytes are retransmitted (except the ACK byte which may be modified before being resent). No data is added to or removed from the ring. Receive data for this station [2] is stored if a buffer is available; otherwise it is discarded, and the B bit is set in the ACK byte. Character stuffing is handled within the RxDATA state

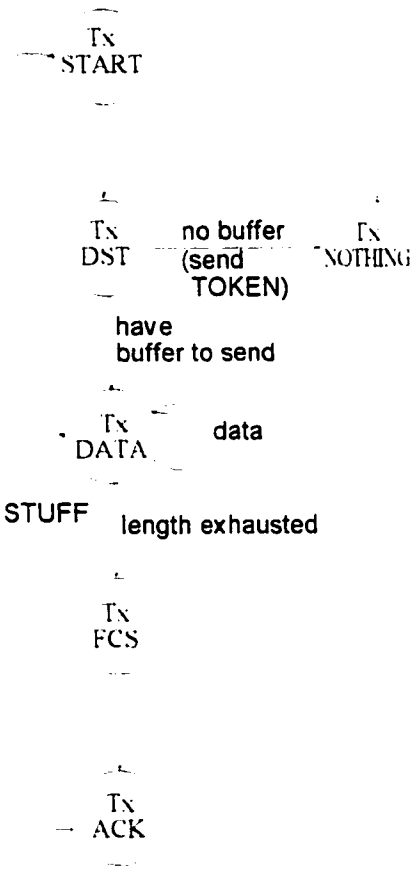


Notes:
1. All received characters are echoed (retransmitted).
2. Received data characters are stored if a buffer is available; otherwise they are discarded.
3. Character stuffing is handled within the RxDATA state, using an auxiliary state variable.
4. The ACK byte may be modified before it is echoed.

Figure B-2 Receive State Machine

[3], using an auxiliary state variable.

The state machine for a transmitting station (the ring owner) is much simpler, since it must merely put a frame as shown in Figure B-1 on the ring. As shown in Figure B-3, if no frame is waiting to be sent, the transmit state machine sends the token sequence, and then turns itself off. It will be restarted



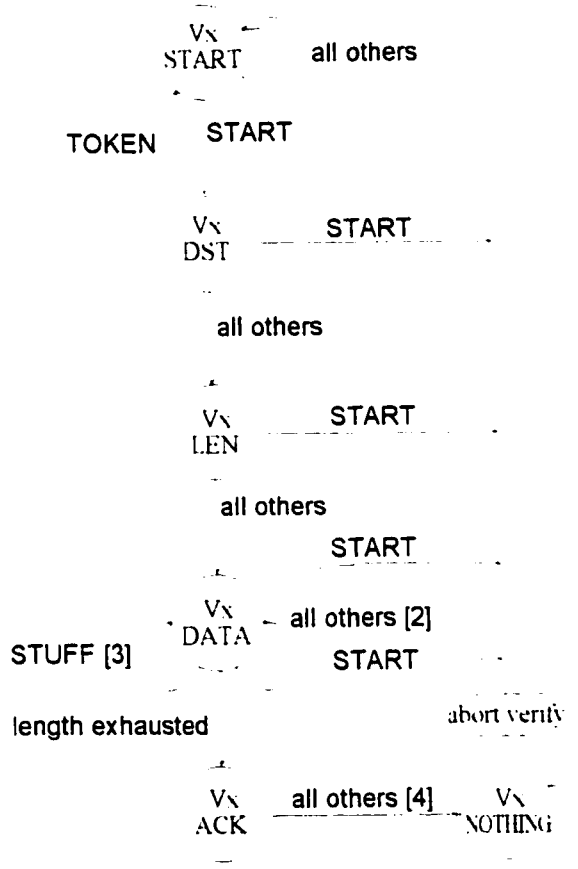
Notes:
1. If no buffer is awaiting transmission, the TOKEN is sent and the transmit state machine halts.
2. Character stuffing is handled within the TxDATA state, using an auxiliary state variable.

Figure B-3 Transmit State Machine

by the receive state machine when a token is received.

While the transmit state machine is placing data on the ring, a parallel state machine (Figure B-

4) removes the data from the ring and compares it to the transmitted data. This "verify" operation cannot be performed synchronously with the transmission, because of the variable and unknown delay for bytes to transit the ring. The verify state machine is essentially identical to the receive state machine, with the following significant differences. Received bytes are consumed instead of being echoed (thus removing the bytes that were placed on the ring by the transmit state machine). Received bytes are not stored.



Notes:
1. All received characters are consumed.
2. Received data characters are compared to the contents of the transmit buffer.
3. Character stuffing is handled within the VxDATA state, using an auxiliary state variable.
4. No valid ACK byte can be a START code.

Figure B–4 Verify State Machine

instead, they are compared with the transmit buffer. If a mismatch is detected, the buffer is flagged with a "verify" error. Receipt of a START code during the frame will abort the verify, and flag the buffer with an "abort" error. Finally, all bytes preceding the START code are consumed: as will be seen shortly, this is the mechanism by which the ring is purged of spurious data.

Only one frame may be on the ring at any time. Once a frame has been sent, the sender must wait for that frame to be verified before sending the next frame (or passing the token). This decreases ring utilization; in a ring with five stations, an idle time of five byte periods will occur between frames. The advantage is a great simplification of the verify logic. More importantly, an idle period is needed to recover from asynchronous framing errors.

B.4 Error Handling and Ring Supervision

Most bit errors will result in a garbled byte of data. Single-bit errors and many multiple-bit errors in the destination, length, source, or data field will be identified by the checksum. The receiving station will discard the frame, and report the error to the sending station through the E bit of the acknowledge byte. (In addition, all multiple-bit errors will be detected by the verify logic.)

Bit errors in the length field will change the apparent length of the frame. It is extremely unlikely that the result will be a valid frame, but the erroneous length may lead to other problems. If the length is decreased by the error, the tail of the frame will appear to be spurious data on the ring. If the length is increased, a complete frame will never be seen. Recovery from both of these conditions will be discussed below. A length increase could also cause a buffer overflow; to prevent this, the receive state machine always limits the length to the maximum buffer size (63 bytes of data).

Bit errors in the START byte will prevent the receiving stations from recognizing the frame. A bit error which creates a spurious START code will convert one valid frame into two invalid frames, both of which will be rejected.

Bit errors in the TOKEN byte can cause the loss of the token, discussed below. An erroneous TOKEN code in the middle of a frame will be ignored (treated as bad data). Only a multiple-bit error in

the destination address field can create a spurious token. Should this happen, two stations may begin putting frames on the ring. It is conceivable that each station will exactly consume the other's frames (causing both stations to see verify errors). But eventually, one station will pass the TOKEN before the other. TOKENs are simply consumed by the verify state machine, since any station running the verify state machine already considers itself the ring owner. Thus the spurious TOKEN will be destroyed, and normal operation will resume.

Bit errors in the acknowledge byte are an unresolved problem. If a single-bit error clears the E or B bits, the transmitting station may incorrectly conclude that the frame was successfully received. (In the IEEE 802.5 protocol, this problem was addressed by sending the flag bits twice.) This will be a subject for future development.

Unlike IEEE 802.5 (and other bit synchronous systems), a garbled bit in an asynchronous ring can cause the apparent *loss* of a byte, or the addition of a *spurious* byte. Also, the corruption of a start bit can cause a loss of byte synchronization (continuous framing errors); this is handled by requiring each frame to "clear" the ring before another frame may be placed on the ring.

A spurious byte may occur before a frame, within a frame, or after a frame. Spurious bytes preceding a frame will be removed by the verify state machine, while it waits for a valid START code. Spurious bytes following a frame will appear to be spurious bytes preceding the next frame, and will likewise be removed. Note that even spurious bytes preceding the START TOKEN sequence will eventually appear as bytes preceding a frame. Thus by requiring the verify state machine to discard invalid START bytes, the ring is regularly purged.

A spurious byte inserted within a frame will corrupt the remainder of the frame and increase its length by one, appearing to the listening stations as corrupted data followed by a spurious byte. The frame will be flagged as erroneous, and the "appended" byte will be discarded as just described.

"Lost" data bytes are a difficult problem. If a data byte is lost around the ring, the verify state machine never sees a completed frame. Since it must remove the entire frame from the ring before

sending the next frame or passing the token, the ring will halt. This problem is addressed by a timeout counter. If the ring owner sees no receive (verify) data for 165 msec,[1] it declares that frame to be "timed out." The frame is flagged as unsuccessfully sent due to a timeout error, and the owner proceeds to send the next message or pass the token. Receiving stations awaiting completion of the frame will be reset by the START code of the new message or token.

It is also possible to lose the token, especially during idle periods when the token is circulating continuously on the ring. When this happens, the station sending the token sees that the token has been passed, and considers itself no longer the ring owner. But no station receives the token to become the new owner. If the token was garbled, the result is a ring of slaves endlessly circulating a nonsense byte. To handle this problem, one station is designated the "supervisor" of the ring. It monitors the ring constantly for the presence of a token. If 385 msec[2] elapses without a token being seen, it declares a "supervisor timeout" and generates a new token to be placed on the ring. The "non supervisor" stations also detect the loss of token; however, their action is merely to set a "slave timeout" flag. This flag may be used by higher level software to detect network failure.

Station number 01 is designated the supervisor. This is appropriate for this application, where the ring consists of several embedded controllers plus one PC as an operator's station. A second PC may be installed as a "monitor" station, to passively observe and tabulate ring traffic.

B.5 Broadcast Protocols

The handling of broadcast messages was a particular goal in the design of the Asynchronous Token Ring. A major advantage of physical ring protocols like IEEE 802.5 is the inherent ability to acknowledge a broadcast message, since every station may modify the acknowledge byte.

---

1. Three ticks of the 18.2 Hz system clock. If the first clock tick occurs immediately after the verify engine starts, the second tick could be seen in as little as 55 msec. At 9600 baud, each station adds a minimum delay of 1 msec. Thus a network of 55 stations could cause a false timeout, if the counter was set to two clock ticks.

2. Seven ticks of the 18.2 Hz clock. This timeout delay is a compromise. If too long, the network can go "down" for an excessively long period. If too short, heavy traffic on the network (each station sending packets before passing the token) can cause spurious timeouts. Further research may yield a more satisfactory solution to this problem.

IEEE 802.5 allows each station to flag "address recognized" and "frame copied" (successfully received). For this application, it is more useful to indicate "frame *not* copied." In the case of a broadcast message, any station may set this bit; when the sender sees this bit set, it knows that at least one station failed to copy the message, and a retransmission is required. The acknowledgement differentiates between messages which must be resent due to data errors (the E bit), and due to the lack of a receive buffer (the B bit).

An innovation here is the use of an address-recognized *count*, rather than a single bit. For messages to a single destination, this count will be returned to the sender as either 0 (address not recognized), or 1 (address recognized). For broadcast messages, the count is the number of stations recognizing the broadcast address (regardless of whether they successfully copied the message). If the sender knows the number of stations on the ring, it can detect whether any station failed to recognize the message.

The Asynchronous Token Ring essentially provides an "acknowledged datagram" service (Tanenbaum, 1989). This is a "connectionless" service which does not guarantee message sequence, but does indicate successful receipt. Two kinds of message delivery can be offered to the application. "At most once" transmits the message only once. No destination will receive a duplicate of the message, but some destinations may fail to receive it at all. "At least once" delivery retransmits the message until it is succesfully received by all addressed destinations (the E and B bits return clear, and the message verifies successfully). Every destination receives the message, but some may receive duplicate copies. Rejection of duplicate messages is handled at a higher protocol layer.

B.6 Message Interpretation

It became necessary to write the received-message processing software, before the content of received messages was finalized. To do this, an "open ended" message format was devised. Each data field is a statement in the tokenized ITL[3] language. When the token ring state machine receives a valid

3. The Inferencing Token Language is described in Chapter 3.

message, it is passed to the ITL interpreter. Some ITL tokens are defined specifically for message processing (see Appendix D); others -- such as ASK and TELL -- represent network-related actions which must be taken by the processor. Since new tokens can be added to the ITL language at any time, this provides a very flexible message processing mechanism.

This approach allowed a simple method to reject duplicate messages. The SEQUENCE token and a sequence number are placed by the sender at the beginning of a message. This token has the following action: if the sequence number matches the last sequence number received *from this sender*, terminate ITL processing, ignoring the rest of the message. The sender increments the sequence number for each new message to the same destination. The guaranteed delivery of the "at least once" service ensures that sequence numbers will not be skipped.

This also allows sequencing of broadcast messages. Each sender maintains a distinct "outgoing" sequence number for broadcasts. Each recipient of the broadcast message, upon encountering the SEQUENCE token, compares the sequence number to that of the last *broadcast* message from that sender. Thus each station maintains two "incoming" sequence numbers for each possible sender: one for point-to-point messages, and one for broadcast.

B.7 Future Work

Several refinements of the protocol are possible. The implementation of the acknowledge byte is weak; it should be sent redundantly (as in 802.5) or with an error-check code (such as a parity bit). With the current acknowledge format, only 64 of the 252 valid station addresses can be used. Some of the remaining 188 could be used for group addresses. Also, the maximum message length could be increased from 63 to 251, although in this case a better frame check sequence would be advisable.

At present, one station (the supervisor) is crucial to ring operation. It would be better if any station could assume the role of ring supervisor, as in IEEE 802.5.

The current network uses RS-232 links, star-wired to a passive hub -- that is, serial cables from each station are brought to a central switchbox. This allows any station to be manually bypassed. An

electronic switch that automatically bypasses any inactive station would let the ring recover from a "dead" CPU.

The potential of "active" messages which are executed by an interpreter has yet to be explored. Use of the ITL interpreter opens the possibility of distributed execution of ITL language -- processors sending logic expressions and even entire subprograms to other processors over the network, where they are executed to produce similar "return" messages.

APPENDIX C. TOWARD A DISTRIBUTED, OBJECT-BASED FORTH

C.1 Objectives

As the complexity of the accelerator expert system increased, it quickly became clear that an object-based[1] approach would be useful. For example, all rules may be asserted with a new truth value, but the actions involved to assert this value may vary depending on the particular rule: a simple fact may just involve a store, while a temporal fact may require updating of historical information. The obvious way to manage this is to define classes of rules with associated methods.

An object model is also well suited to distributed processing. The concept of "send a message to an object" translates directly to a network message with parameters, whereas a remote procedure call involves the further (and subtle) problems of blocking and parameter return. A message passing protocol may be implemented with nonblocking calls, greatly preferred for this application.

This distributed object implementation should meet four objectives:

a) *Late binding.* Processor A may not know the class of the rule on Processor B, to which a message is being sent. Thus, late binding is implicitly required.

b) *Fast.* Since methods such as *assert* and *invoke* are used frequently in the real-time control loop, it is important that their execution be fast. This precludes many of the late-binding techniques currently in use (Rodriguez and Poehlman, 1996).

c) *Abstract.* Many early-binding systems achieve speed by precompiling object or method references as addresses in the CPU data space. In a distributed system of mixed CPUs, this technique is not feasible. All objects and methods must be identified by abstract labels or selectors.

d) *Not encapsulated.* For the immediate application, there is no pressing need for encapsulation or

1. (Tanenbaum, 1995) makes the distinction between object-based and object-oriented systems. Briefly, object-based systems use objects and methods, but with no attempt at encapsulation.

information hiding, or for any particular syntax.

C.2 Implementation

All objects in a given CPU are referenced by "object numbers" starting from zero. Within each CPU is an "object index," which points to the data structure for each object (Figure C-1).

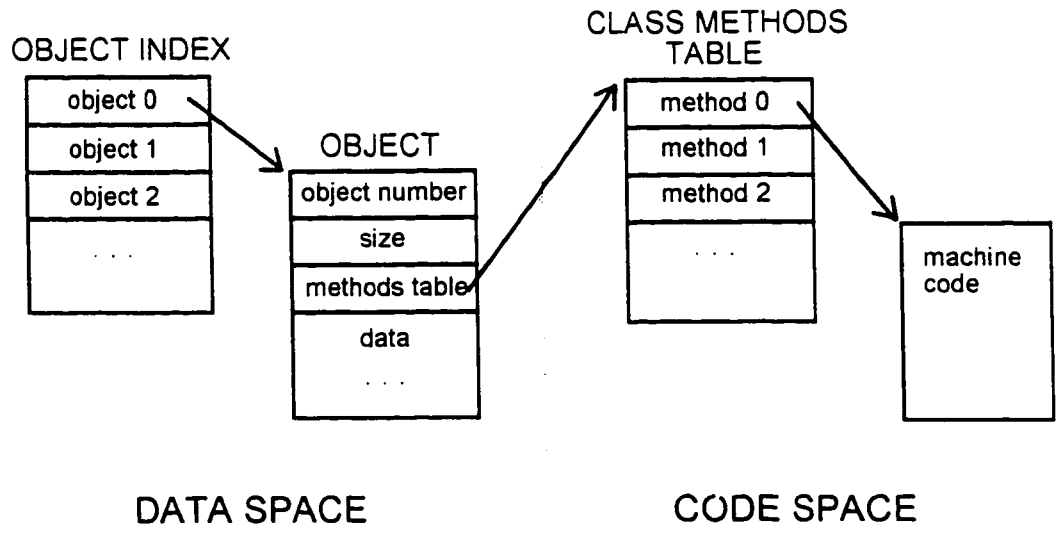


Figure C-1. Object Data Structures

The object data structure contains a header, followed by the instance data for the object. The header includes the object number,[2] size of this instance, and a pointer to the methods table for this class of object.

Like objects, all methods are referenced by method numbers starting from zero, which index into a table. In this case, the table contains the address of Forth words (typically machine code) which perform the actions for each method. Method tables have been previously employed (Morgenstern, 1990; Zsoter, 1995). Method numbers are shared among all classes: e.g., *assert* is method #3 in every class, and classes which do not support *assert* are required to have an *undefined* method in table entry 3. This means that

2. References to "self" need to determine the object *number*, given the object's *address*.

for most classes, the methods table is sparsely populated -- a tolerable tradeoff for execution speed -- and that method numbers must be carefully assigned (to be discussed shortly).

Object data, which can change, must be located in the Data (RAM) space of the embedded processor. Methods are fixed at compile time, and thus located in Code (ROM) space. Once compiled and burned into ROM, the methods of a class are fixed -- but an object may be changed to a different class.

Executing a given method for a given object thus involves an index and fetch into the object table, an offset and fetch from the object, an index and fetch into the methods table, and an indirect jump. This requires only 9 machine instructions on the 8086, and 12 machine instructions on the 68HC16, an acceptable overhead for a late-binding procedure call.[3]

C.3 Syntax

Class data structures (instance variables) are defined using an extension of the common Forth structure declaration word set (Bradley, 1984; Pountain, 1987; Bartel and Bartel, 1989). The syntax for a declaration is

```
INHERIT classname
       size ITEM fieldname
       size ITEM fieldname
         . . .
CONSTANT sizename
```

The data structure[4] of the parent class is inherited, and additional instance variables may be defined with ITEM. If this is a totally new class, it inherits from superclass COMMON. After the last ITEM is defined, the total size of the instance data is available on the stack; this is defined as a normal Forth CONSTANT so that it may be used later in the class definition.[5]

Next, the method routines can be defined. Each method is a Forth word -- high-level or machine

---

3. (Zsoter, 1996) describes the use of CPU registers to hold the address of the current object, and the address of its methods table. With this technique, the late-binding call overhead can vanish completely on some processors, e.g., the 80386.

4. Strictly speaking, only the data allocation is inherited here. Instance variables are visible to all classes, and so are available to the child class.

5. A CONSTANT is not necessary, but is much safer in the event of a stack imbalance during the definition of the methods.

code -- which will receive the address of the object on the top of stack. Additional parameters passed to the method, if any, will be on the stack under the object address. The method consumes the object address and parameters, and leaves any return parameters on the stack.

Each method is given a name with the statement

```
n METHOD methodname
```

For example, the first four methods of a "fact" object are

```
0 METHOD :build                  \ initialize new rule object
1 METHOD :scrub                  \ reset rule to "unknown"
2 METHOD :invoke     ( -- d )    \ get rule's truth value
3 METHOD :assert     ( d -- )    \ set new truth value
```

These indices refer to locations in a manually constructed methods table, containing the addresses of the corresponding Forth words:

```
CREATE tablename    ' word ,   ' word , ...
```

or, in this example,

```
CREATE FactMethods
    ' FactBuild ,    ' FactScrub ,
    ' FactInvoke ,   ' FactAssert ,
```

The class is then defined with

```
sizename tablename CLASS classname
```

Objects may be instantiated dynamically at run-time, or statically at compile-time. The latter is preferred due to its faster and deterministic run-time speed. Invoking "classname" will cause an object to be created, given a name, and added to the object index:

```
classname objectname
```

When "objectname" is executed, it will then return the object number. Executing a "methodname" will then apply a method to that object. By convention, method (message) names begin with a colon (:), so to the programmer, the message passing syntax is

```
(optional parameters)  object :message
```

C.4 Further Work

No effort has been expended to make the syntax "pretty." A starkly functional package was adequate to develop the object-based expert system, and allows experimentation with the "internals" of the object implementation. An improved syntax would make the code easier to write and to understand.

The greatest current deficiency is the need to manually assign and track method numbers, and manually construct the methods table. This could be automated if a suitable assignment algorithm were employed. (Hamilton, 1996) describes an object system remarkably similar to this one, and suggests the use of a graph-coloring algorithm to efficiently assign method numbers (selectors).

There has as yet been no need for encapsulation or information hiding. All methods, data structures, and selectors are visible at all times. Should encapsulation become desirable, any of the techniques employed by previous object-oriented Forths -- vocabularies, dictionary relinking, or private symbol tables -- should be directly applicable.

At present, only single inheritance of instance data is explicitly supported. (Method inheritance is implicitly supported, since all methods are global and the methods table is manually constructed.) As rule classes have been defined to perform specialized functions (PID control, history tracking), the need for multiple inheritance has become evident.

Although this package builds the foundation for a true distributed object-oriented system, such a system has not yet been implemented. This will probably involve the creation of a "DoMethod" network message, plus messages to pass parameters. Methods currently may return parameters as well as accept messages; either some network mechanism for returned data must be devised (analogous to Remote Procedure Call), or the object-programming model should be revised to eliminate the need for return values (e.g. by returning object messages instead).

APPENDIX D. THE INFERENCING TOKEN LANGUAGE

Network Inferencing Functions

Hex	Token name	Function
10	<Exit>	Terminate token processing (end of ITL statement)
11	<Text>	Put following text in the Terminal Input Buffer (TIB). Followed immediately by a counted string: a byte count n, then n characters.
12	<DoText>	Interpret the text string in TIB as a Forth command, then clear the TIB
13	<Type>	Type the following text to the console. Followed immediately by a counted string: a byte count n, then n characters.
14	<Sequence>	Terminate token processing (exit the token thread) if this sequence number matches the last received from this sender. Followed immediately by a one-byte sequence number.
15	<Noop>	Do nothing.
16	<ReSequence>	Reset all sequence numbers. Normally sent as a broadcast, in an unsequenced message, to reset all the CPUs in the network.
17	<Setclock>	Set the real-time clock to the given 32-bit value. Followed immediately by a four-byte value in "big-endian" order: MSB, 2ndSB, 3rdSB, LSB.
18	<Tell>	Assert the given value and expiration time for a rule. Followed immediately by rule#, time, and value, all 16-bit values in "big-endian" order.
19	<Ask>	Request the current value and expiration time of a rule. Followed immediately by rule#, a 16-bit value in "big-endian"order. The reply is a <Tell> message.
1A	<Invoke>	Invoke the given rule. Followed immedately by a 16-bit rule number
1B	<Scrub>	Reset all rules to "unknown."

Rule-Writing Language

Hex	Token name	Function
20	<#literal>	Put a literal value on the stack. Followed immediately by a two-byte value in "big-endian" order.

Rule-Writing Language

Hex	Token name	Function
21	<'literal>	Execute the Forth word at the given address. Followed by a two-byte address in "big-endian" order
22	<UnnestF>	Terminate token processing (exit the token thread) IF the fact on the top of stack is unknown or false. Regardless of outcome, the fact is left undisturbed on the stack.
23	<Now>	Put the current 16-bit time on the stack.
24	<Forever>	Put the maximum future time on the stack.
25	<Hence>	Put the current 16-bit time plus a literal value on the stack. Followed immediately by a two-byte in "big-endian" order.
26	<Fetch>	Fetch the Forth variable specified by the following address, and stack it with an expiration date of "forever." Followed by two-byte "big-endian" address
30	AND	Logicaly AND two temporal quantities on the stack. (A temporal quantity has a value and an expiration time.)
31	OR	Logically OR two temporal quantities on the stack.
32	NOT	Logically invert a temporal quantity on the stack.
33	NULL	Put the temporal quantity "false, unknown" on the stack. (A value of zero, and an expired time.)
40	+	Add two temporal integers on the stack.
41	-	Subtract two temporal integers on the stack.
42	MIN	Return the minimum of two temporal integers.
43	MAX	Return the maximum of two temporal integers.
44	>	Return a temporal true, if temporal integer 1 > temporal integer 2.
45	<	Return a temporal true, if temporal integer 1 < temporal integer 2.
46	=	Return a temporal true, if temporal integer 1 = temporal integer 2.
47	U<	Like < but uses unsigned integer comparison rather than signed.
48	U>	Like > but uses unsigned integer comparison rather than signed.
49	0=	Return a temporal true, if the value of the temporal integer is zero.
4A	0<	Return a temporal true, if the value of the temporal integer is negative.

APPENDIX E. ACCELERATOR CONTROL PROGRAM

```
\ ============================================================
\ CPU #0C fact definitions
\ ============================================================
hex 0C target

\ INPUT "RULES" ------------------------------------------
\ All of the analog inputs are software filtered.  These
\ rules are evaluated 18 times/second, by making them
\ dependent on the 18HZ fact.

HEsteer1x RULE:
    DOFORTH adc2  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
HEsteer1x 18HZ :dependent
HEsteer1x StdCoeffs
hex 82C decimal 300 HEsteer1x ScaleFactors

HEsteer1y RULE:
    DOFORTH adc3  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
HEsteer1y 18HZ :dependent
HEsteer1y StdCoeffs
hex 7E7 decimal 330 HEsteer1y ScaleFactors

HEsteer2x RULE:
    DOFORTH adc4  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
HEsteer2x 18HZ :dependent
HEsteer2x StdCoeffs
hex 753 decimal 435 HEsteer2x ScaleFactors

HEsteer2y RULE:
    DOFORTH adc5  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
HEsteer2y 18HZ :dependent
HEsteer2y StdCoeffs
hex 78C decimal 470 HEsteer2y ScaleFactors

CoronaPos RULE:
```

```
    DOFORTH adc6  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
CoronaPos 18HZ :dependent
CoronaPos StdCoeffs
hex   0 decimal 550 CoronaPos ScaleFactors

CoronaLoad RULE:
    DOFORTH adc7  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
CoronaLoad 18HZ :dependent
CoronaLoad StdCoeffs
hex  1D decimal 214 CoronaLoad ScaleFactors

NISvacuum RULE:
    DOFORTH adc8  DOFORTH 2ndFilter
    18 HENCE
;RULE
NISvacuum 18HZ :dependent
NISvacuum StdCoeffs

LEvacuum RULE:
    DOFORTH adc9  DOFORTH 2ndFilter
    18 HENCE
;RULE
LEvacuum 18HZ :dependent
LEvacuum StdCoeffs

HEvacuum RULE:
    DOFORTH adc10  DOFORTH 2ndFilter
    18 HENCE
;RULE
HEvacuum 18HZ :dependent
HEvacuum StdCoeffs

Ionpumps RULE:
    DOFORTH adc11  DOFORTH 2ndFilter
    18 HENCE
;RULE
Ionpumps 18HZ :dependent
Ionpumps StdCoeffs

\ OUTPUT RULES ------------------------------------------

\ OPERATING RULES ---------------------------------------
decimal
variable MaxExtension   220 MaxExtension !     ( 22" )
variable MinExtension    40 MinExtension !     (  4" )
variable DesiredCorona   30 DesiredCorona !    ( 30 uA)
```

```
variable ThreshCorona
variable OuterThr      5 OuterThr !   5 ThreshCorona !
variable InnerThr      2 InnerThr !   ( for hysteresis )
: +extend    -1 RLY6 0 RLY7   InnerThr @ ThreshCorona ! ;
: +retract   0 RLY6 -1 RLY7   InnerThr @ ThreshCorona ! ;
: 0points    0 RLY6  0 RLY7   OuterThr @ ThreshCorona ! ;

CoronaLoad-low RULE:
     INVOKE CoronaLoad
     DesiredCorona FETCH    ThreshCorona FETCH  -
     <
;RULE
CoronaLoad-low 1HZ :dependent
0 CoronaLoad-low !export

CoronaLoad-high RULE:
     INVOKE CoronaLoad
     DesiredCorona FETCH    ThreshCorona FETCH  +
     >
;RULE
CoronaLoad-high 1HZ :dependent
0 CoronaLoad-high !export

ExtendPoints RULE:
     CoronaLoad-low
     TermMV-high  AND
     INVOKE CoronaPos  MaxExtension FETCH <  AND
     CONCLUDES DOFORTH +extend
;RULE

RetractPoints RULE:
     CoronaLoad-high
     TermMV-low  AND
     INVOKE CoronaPos  MinExtension FETCH >  AND
     CONCLUDES DOFORTH +retract
;RULE

HoldPoints RULE:
     ExtendPoints
     RetractPoints OR  NOT
     CONCLUDES DOFORTH 0points
;RULE

\ Rules to turn off extend/retract motors incorporate
\ hysteresis by using a different test value.
\ Note also, < > for position test gives small hysteresis.
\
\ -ExtendPoints RULE:
\      INVOKE CoronaLoad    DesiredCorona FETCH    >
\      INVOKE CoronaPos  MaxExtension FETCH >  OR
```

```
\     DesiredMV 0=  OR
\     CONCLUDES DOFORTH -extend
\ ;RULE
\ -ExtendPoints 1HZ :dependent
\
\ -RetractPoints RULE:
\     INVOKE CoronaLoad   DesiredCorona FETCH   <
\     DesiredMV 0= NOT  AND
\     INVOKE CoronaPos  MinExtension FETCH <  OR
\     CONCLUDES DOFORTH -retract
\ ;RULE
\ -RetractPoints 1HZ :dependent

\ ============================================================
\ CPU #0D fact definitions
\ ============================================================
hex 0D target

\ INPUT "RULES" ----------------------------------------
\ All of the analog inputs are software filtered.  These
\ rules are evaluated 18 times/second, by making them
\ dependent on the 18HZ fact.

HEchgV RULE:
    DOFORTH adc2  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
HEchgV 18HZ :dependent
HEchgV StdCoeffs
hex -0BD decimal 628 HEchgV ScaleFactors

HEchgA RULE:
    DOFORTH adc3  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
HEchgA 18HZ :dependent
HEchgA StdCoeffs
hex   4 decimal 250 HEchgA ScaleFactors

LEchgV RULE:
    DOFORTH adc4  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
LEchgV 18HZ :dependent
LEchgV StdCoeffs
hex  5D decimal 700 LEchgV ScaleFactors

LEchgA RULE:
    DOFORTH adc5  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
```

```
;RULE
LEchgA 18HZ :dependent
LEchgA StdCoeffs
hex -0B3 decimal 250 LEchgA ScaleFactors

HEcolA RULE:
    DOFORTH adc6  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
HEcolA 18HZ :dependent
HEcolA StdCoeffs
hex  72 decimal 190 HEcolA ScaleFactors

LEcolA RULE:
    DOFORTH adc7  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
LEcolA 18HZ :dependent
LEcolA StdCoeffs
hex 0D5 decimal 270 LEcolA ScaleFactors

TermMV RULE:
    DOFORTH adc8  DOFORTH 2ndFilter
    DOFORTH ApplyScale  18 HENCE
;RULE
TermMV 18HZ :dependent
TermMV StdCoeffs
hex  12 decimal 2100 TermMV ScaleFactors

HEquad1 RULE:
    DOFORTH adc10  DOFORTH 2ndFilter
    18 HENCE
;RULE
HEquad1 18HZ :dependent
HEquad1 StdCoeffs

HEquad2 RULE:
    DOFORTH adc11  DOFORTH 2ndFilter
    18 HENCE
;RULE
HEquad2 18HZ :dependent
HEquad2 StdCoeffs

\ OUTPUT RULES ------------------------------------------
\ The analog outputs are set via closed-loop PID control.
\ These rules are evaluated 18 times/second, by making them
\ dependent on the 18HZ fact.

HEchgVOut RULE:
    HEchgV  DOFORTH DROP     \ sensor value
```

```
    DOFORTH ApplyPID  DOFORTH dac1
    DOFORTH OLDFACT
;RULE
hex 100 SET HEchgVOut .Igain
   1000 SET HEchgVOut .Pgain
      0 SET HEchgVOut .Dgain
      0 SET HEchgVOut .sumerror
      0 SET HEchgVOut .error

LEchgVOut RULE:
    LEchgV  DOFORTH DROP     \ sensor value
    DOFORTH ApplyPID  DOFORTH dac2
    DOFORTH OLDFACT
;RULE
hex 100 SET LEchgVOut .Igain
   1000 SET LEchgVOut .Pgain
      0 SET LEchgVOut .Dgain
      0 SET LEchgVOut .sumerror
      0 SET LEchgVOut .error

: resetpid  0 HEchgVOut objadr .sumerror !
            0 LEchgVOut objadr .sumerror ! ;

\ OPERATING RULES ----------------------------------------
\ terminal voltage control
\ these rules are evaluated every 1 second
decimal
VARIABLE Setpoint*10    300 Setpoint*10 !
VARIABLE Increment       5 Increment !
VARIABLE Tweak           1 Tweak !       ( fine adjustment )
VARIABLE Threshold      20 Threshold !   ( 0.20 MV )
VARIABLE TweakThresh     1 TweakThresh ! ( 0.01 MV )
VARIABLE RateTrip       10 RateTrip !    ( 0.10 MV/sec max )
VARIABLE MVTrip        500 MVTrip !      ( 5.00 MV max )
VARIABLE SparkTrip     -50 SparkTrip !   ( 0.50 MV drop )

: le  18 hence lechgvout :assert ;
: he  18 hence hechgvout :assert ;
: mv  forever desiredmv :assert ;   0 DesiredMV !export

\ emergency shutdown
: Panic
    0 mv  0 le  0 he    \ all setpoints to zero  ***TEMP***
    ;

: -chg  ( increment -- )
    Setpoint*10 @       ( current value)
    SWAP -              ( decrease, )
    300 MAX             (  but not less than 30)
    DUP Setpoint*10 !
```

```
    DUP 10 / 18 HENCE  LEchgVOut :assert
    5 + 10 / 18 HENCE  HEchgVout :assert ;    ( staggered )

: --chargeKV  Increment @ -chg ;
: -chargeKV   Tweak @ -chg ;

: +chg  ( increment -- )
    Setpoint*10 @        ( current value)
    SWAP +               ( increase, )
    1500 MIN             (  but not less than 150)
    DUP Setpoint*10 !
    DUP 10 / 18 HENCE  LEchgVOut :assert
    5 + 10 / 18 HENCE  HEchgVout :assert ;

: ++chargeKV  Increment @ +chg ;
: +chargeKV   Tweak @ +chg ;

VARIABLE PrevMV              \ previous TermMV reading
: DeltaMVEval
    TermMV :invoke swap          \ current TermMV reading
    DUP PrevMV @ -
    SWAP PrevMV !
    swap ;                    ( -- value time )


DeltaMV RULE:                \ change in Terminal MV
    DOFORTH DeltaMVEval
;RULE
DeltaMV 1HZ :dependent

TermMV-low RULE:          \ true if Desired>Term by at least .20 MV
    DesiredMV  INVOKE TermMV  -
    Threshold FETCH  >
;RULE
TermMV-low 1HZ :dependent
0 TermMV-low !export

TermMV-high RULE:         \ true if Desired<Term by at least .20 MV
    INVOKE TermMV  DesiredMV -
    Threshold FETCH  >
;RULE
TermMV-high 1HZ :dependent
0 TermMV-high !export

TermMV-bitlow RULE:       \ true if Desired>Term by at least .01 MV
    DesiredMV  INVOKE TermMV  -
    TweakThresh FETCH  >
;RULE
TermMV-bitlow 1HZ :dependent
```

```
TermMV-bithigh RULE:      \ true if Desired<Term by at least .01 MV
    INVOKE TermMV  DesiredMV -
    TweakThresh FETCH  >
;RULE
TermMV-bithigh 1HZ :dependent

TermMV-trip RULE:
    INVOKE TermMV  MVTrip FETCH  >
    CONCLUDES DOFORTH Panic
;RULE
TermMV-trip 1HZ :dependent

Spark RULE:
    DeltaMV  SparkTrip FETCH  <
    CONCLUDES DOFORTH Panic
;RULE

ChargeTooFast RULE:
    TermMV-low
    INVOKE DeltaMV  RateTrip FETCH  >  AND
    CONCLUDES DOFORTH --chargeKV
;RULE

MoreTweak RULE:
    INVOKE TermMV-bitlow
    TermMV-low NOT  AND
    CONCLUDES DOFORTH +chargekv
;RULE

LessTweak RULE:
    INVOKE TermMV-bithigh
    TermMV-high NOT  AND
    CONCLUDES DOFORTH -chargekv
;RULE

MoreCharge RULE:
    CoronaLoad-high NOT
    TermMV-low   AND
    CONCLUDES DOFORTH ++chargeKV
;RULE

LessCharge RULE:
    CoronaLoad-low NOT
    TermMV-high   AND
    CONCLUDES DOFORTH --chargeKV
;RULE
```

APPENDIX F. TOWERS OF HANOI BENCHMARK

F.1 Towers of Hanoi in CLIPS

```
; Towers of Hanoi in CLIPS

(defrule move-disk
   ?old-list  <- (goals disk ?number ?from ?to $?rest)
=>
   (retract  ?old-list)
;   (printout t "move disk from " ?from " to " ?to crlf)
   (assert (goals ?rest)))

(defrule move-single-disk-tower
   ?old-list  <- (goals tower 1 ?from ?to $?rest)
=>
   (retract  ?old-list)
   (assert (goals disk 1 ?from ?to ?rest)))

(defrule move-tower
   ?old-list  <- (goals tower ?number&~1 ?from ?to $?rest)
=>
   (retract  ?old-list)
   (assert (goals tower =(- ?number 1) ?from =(- 7 ?from ?to)
            tower 1 ?from ?to
            tower =(- ?number 1) =(- 7 ?from ?to) ?to  ?rest)))

(defrule completed
   ?old-list <- (goals done)
=>
   (retract ?old-list))
```

F.2 Towers of Hanoi in TEXMEX 3

```
( ================ TOWERS OF HANOI IN TEXMEX3 ================ )
(         ADAPTED FOR DISTRIBUTED INFERENCING   5 DEC 95
( translated from Towers of Hanoi in FORPS,
( by Christopher J. Matheus,
( in "The Internals of FORPS: A FORth-based Production System"
( Journal of Forth Application and Research, Vol. 4, No. 1

1 constant GoalKeeper          \ keeper of the goals
2 constant DiskExpert          \ knows how to move disks
3 constant TowerExpert         \ knows how to move towers


 1 CONSTANT HOME          2 CONSTANT GOAL        ( tower numbers
 0 CONSTANT TOWER         $80 CONSTANT DISK      ( goals)
VARIABLE SOURCE    VARIABLE TARGET     VARIABLE SPARE
VARIABLE #DISKS    VARIABLE DISK?      VARIABLE STARTED

: SPARE!  ( - )    SOURCE @  TARGET @  OR 7 XOR SPARE ! ;

( Constants, variables, and GOALSTACK words, from FORPS.)
( Modified to change cell size from 4 to 2 bytes.)
( Modified again to pack goal into a single cell.)
(    f f f f t t t t d n n n n n n n n
(    -from-- --to---   ---#disks----  d=1 if disk, 0 if tower)


10 CONSTANT MAX-#DISKS
CREATE GOALSTACK  MAX-#DISKS 1- DUP + 2 +  CELLS ALLOT
VARIABLE GS.PTR   GOALSTACK GS.PTR !

: >GSTACK ( n a - a)    DUP ROT SWAP ! 2 + ;
: GSTACK> ( a - a n)    2 - DUP @ ;
: >GOAL ( goal -- )
    GS.PTR @ !  2 GS.PTR +! ;
: GOAL> ( -- goal )
    -2 GS.PTR +!  GS.PTR @ @ ;
: @GOAL   GS.PTR @ 2- @ ;
variable curgoal          \ to signal when problem is solved
: .goals
    cr gs.ptr @ goalstack do i @ 5 u.r 2 +loop ;

: PACKGOAL ( action #disks from to - goal time )
    $100 * SWAP $1000 * + + +   \ pack into one cell
    FOREVER ;
: UNPACKGOAL ( goal -- )
    DUP $80 AND DISK? !
```

```
    DUP $7F AND #DISKS !
    FLIP $FF AND  $10 /MOD  SOURCE !  TARGET !  SPARE! ;

: maingoal  ( #disks -- )
    #disks !
    goalstack gs.ptr !
    0 >goal
    tower #disks @ home goal packgoal drop >goal ;

\ : ADDGOAL ( action #disks from to - )
\     GS.PTR @ >GSTACK >GSTACK >GSTACK >GSTACK GS.PTR '
\ : REMOVEGOAL ( - )   GS.PTR @  GSTACK> DROP  GSTACK> #DISKS '
\     GSTACK> SOURCE !  GSTACK> TARGET !  SPARE!  GS.PTR '  ;

: .PEG ( n)   DUP 1 = IF ." A" DROP ELSE 2 = IF ." B" ELSE
        ." C" THEN THEN ;

\ Knowledge Base ============================================
( The logical expression IS-GOAL must be converted to a rule
( for TEXMEX to deduce its consequences.
( The "truth value" returned by IS-GOAL is the current goal,
( rather than a boolean flag.
( IS-#-OF-DISKS is similarly modified.

\ Rules owned by Goalkeeper.
FACT ADDGOAL      \ These three words have propound actions
  ADDGOAL EVALUATOR-IS (import)
  Goalkeeper ADDGOAL ]Rule .Owner !
FACT REMOVEGOAL   \ which manage the goal stack.
  REMOVEGOAL EVALUATOR-IS (import)
  Goalkeeper REMOVEGOAL ]Rule .Owner !
FACT NEWGOAL      \ The TELL message causes the action to happen.
  NEWGOAL EVALUATOR-IS (import)
  Goalkeeper NEWGOAL ]Rule .Owner !
FACT CURRENT-GOAL    \ returns a cell value, the current goal.
  CURRENT-GOAL EVALUATOR-IS (import)
  Goalkeeper CURRENT-GOAL ]Rule .Owner !

\ Rules owned by TowerExpert.
FACT MOVE-TOWER
  MOVE-TOWER EVALUATOR-IS (import)
  TowerExpert MOVE-TOWER ]Rule .Owner !
FACT MOVE-SINGLE-DISK-TOWER
  MOVE-SINGLE-DISK-TOWER EVALUATOR-IS (import)
  TowerExpert MOVE-SINGLE-DISK-TOWER ]Rule .Owner !

\ Rules owned by DiskExpert.
FACT MOVE-SINGLE-DISK
  MOVE-SINGLE-DISK EVALUATOR-IS (import)
  DiskExpert MOVE-SINGLE-DISK ]Rule .Owner !
```

```
\ (import) evaluators will be overridden below for each cpu.

( The following is a direct translation of the FORPS rules.)

: setgoals   ( goal time -- )
  DROP UNPACKGOAL ;

: is-tower?  ( -- flag time )
  DISK? @ 0=  FOREVER ;

: 1-disk?    ( -- flag time )
  #DISKS @ 1 =  FOREVER ;

: many-disks? ( -- flag time )
  #DISKS @ 1 >  FOREVER ;

TowerExpert ME = #IF

    : move-tower-action
        NULL REMOVEGOAL Goalkeeper TellOnce
        TOWER  #DISKS @ 1-  SPARE @  TARGET @ PACKGOAL
            ADDGOAL Goalkeeper TellOnce
        DISK   #DISKS @     SOURCE @ TARGET @ PACKGOAL
            ADDGOAL Goalkeeper TellOnce
        TOWER  #DISKS @ 1-  SOURCE @  SPARE @ PACKGOAL
            ADDGOAL Goalkeeper TellOnce
        -1 FOREVER NEWGOAL Goalkeeper TellOnce
        ;

    MOVE-TOWER RULE:    CURRENT-GOAL  DOFORTH setgoals
                        DOFORTH is-tower?
                        DOFORTH many-disks?
                        AND
    CONCLUDES DOFORTH move-tower-action
    ;RULE

    : move-1tower-action
        NULL REMOVEGOAL Goalkeeper TellOnce
        DISK  1  SOURCE @  TARGET @  PACKGOAL
            ADDGOAL Goalkeeper TellOnce
        -1 FOREVER NEWGOAL Goalkeeper TellOnce
            ;
    MOVE-SINGLE-DISK-TOWER RULE: CURRENT-GOAL  DOFORTH setgoals
                        DOFORTH is-tower?
                        DOFORTH 1-disk?
                        AND
    CONCLUDES DOFORTH move-1tower-action
    ;RULE
```

```
#THEN

DiskExpert ME = #IF

    : move-disk-action
        NULL REMOVEGOAL Goalkeeper TellOnce
        \     TARGET @ SOURCE @
        \     CR ." *** Move disk on peg " .PEG ."  to peg " .PEG
        -1 FOREVER NEWGOAL Goalkeeper TellOnce
            ;

    MOVE-SINGLE-DISK RULE:  CURRENT-GOAL  DOFORTH setgoals
                            DOFORTH is-tower?  NOT
    CONCLUDES DOFORTH move-disk-action
    ;RULE

#THEN

Goalkeeper ME = #IF

    : fetch-goal ( -- goal time)  GS.PTR @ 2- @ FOREVER
        over curgoal ! ;  \ to signal when problem is solved
    CURRENT-GOAL RULE: NEWGOAL  DOFORTH 2drop   DOFORTH fetch-goal
            ;RULE        \ ^--for propagation
    0 CURRENT-GOAL }Rule .Export !     ( changes are broadcast)

    : push-goal ( goal time -- )  DROP >GOAL ;
    ADDGOAL EVALUATOR-IS NULL
    ADDGOAL proponent-IS push-goal

    : pop-goal  ( fact -- )   2DROP GOAL> DROP ;
    REMOVEGOAL EVALUATOR-IS NULL
    REMOVEGOAL proponent-IS pop-goal

    NEWGOAL RULE:  -1 LITERAL NOW  ;RULE     ( for AND)

#THEN

\ Exercise routine ==============================================

( Read the 18.2 Hz BIOS clock)
CODE BIOSCLK     ( - d)
    MOV AH, # 0   INT # $1A   PUSH DX   PUSH CX   NEXT
END-CODE

( The high level word is modified somewhat from FORPS.)
( The system is initiated with the command   n DISKS  )
VARIABLE #CYCLES   1 #CYCLES !

ALSO ITL ALSO FORTH
```

```
: DISKS ( n - )

  FWD-CHAIN ON  BWD-CHAIN OFF
  BIOSCLK DROP  ( n time - )
  #CYCLES @ 0 DO

    OVER maingoal                \ set goal = n disks
    @goal curgoal !              \ zeroed when problem solved
    -1 forever newgoal assert    \ start inferencing
    ( will propagate changes until goal=0.)

    BEGIN
        key? if abort then       \ does ?incoming & ?outgoing
    curgoal @ 0= UNTIL

  LOOP
  BIOSCLK DROP SWAP -  SPACE U. ." ticks"
  DROP  ;

( same, using backward chaining )

: BDISKS ( n - )   FWD-CHAIN OFF

  BIOSCLK DROP  ( n time - )
\  #CYCLES @ 0 DO

    OVER #DISKS !

    GOALSTACK GS.PTR !
    0           >GOAL
    TOWER  #DISKS @  HOME  GOAL  PACKGOAL DROP >GOAL

    BEGIN
        MOVE-TOWER ]Rule EVALUATE
        MOVE-SINGLE-DISK-TOWER ]Rule EVALUATE
        MOVE-SINGLE-DISK ]Rule EVALUATE
    GS.PTR @  2 - @
    0= UNTIL      ( until bottom of rule-stack reached)

\  LOOP
  BIOSCLK DROP SWAP -  SPACE U. ." ticks"
  DROP  ;

PREVIOUS PREVIOUS
bwd-chain off fwd-chain on
```

APPENDIX G. GLOSSARY OF ABBREVIATIONS

ACK Acknowledge.

ATR Asynchronous Token Ring, the local-area network developed during this research.

BPM Beam Profile Monitor.

CPU Central Processing Unit (of a computer).

DSP Digital Signal Processing.

FCS Frame Check Sequence (in this case, a checksum).

FN Model designator of the McMaster University Tandem particle accelerator.

HV High Voltage.

LIPS Logical Inferences Per Second.

ITL Inferencing Token Language.

PID Proportional-Integral-Derivative, a method of linear feedback control.

TEXMEX Threaded EXecution Micro EXpert, the expert system developed during this research.

68HC16 A 16-bit embedded microprocessor made by Motorola, Inc.