# A Real Time Embedded Controller for Smart Structures

by

Christian P. Ahrens

Thesis Submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Richard O. Claus, Chairman

Ronald J. Pieper

Adel Sarrafzadeh

August, 1989
Blacksburg, Virginia

# A Real Time Embedded Controller for Smart Structures

by

Christian P. Ahrens

Dr. Richard O. Claus, Chairman

Electrical Engineering

(ABSTRACT)

This thesis presents a simple Real Time Embedded Control System (RTECS) in an application of Intelligent Structure Monitoring. Research in smart structures, especially the area of vibration suppression, has warranted the investigation of advanced computing environments. Real time PC computing power has limited development of high order control algorithms. The system discussed allows for implementation of I/O intensive algorithms and provides capability for advanced system development.

In an application of Modal Domain Sensing for Vibration Control, RTECS is compared to a PC AT based system for overall functionality and speed. Using a model development procedure, the system is optimized for efficient operation and speed. This includes minimizing the computational overhead associated with I/O. A comparison shows an order of magnitude increase in system speed with larger speed increases discussed. The results provide an avenue for high order control system design. This leads to more accurate device modeling and a higher level of system control.

RTECS employs a novel RISC microcontroller capable of 15 MIPs continuous performance and burst rates of 40 MIPs. Advanced CMOS circuits are integrated on a single printed circuit board measuring 100 mm by 160 mm and require only 1 Watt of power. An operating system written in Forth provides the flexibility for high speed operation in short development cycles.

# Acknowledgements

I would like to express my gratitude to Dr. R.O. Claus, my graduate advisor, for his guidance and words of encouragement. I am also grateful to Dr. Ronald Pieper and Dr. Adel Sarrafzadeh for their help and for serving on my graduate committee.

I would also like to thank Dr. James Phelan for his constant words of advice, support, and for providing me the opportunity to pursue this work. Also, a special thanks is due to

for his help and motivation. Without his efforts, this thesis would not have been possible. And finally thanks go to my family and dear friend for their enduring support and encouragement.

# Table of Contents

# List of Illustrations

# 1.0 Introduction

Research of Smart Structures for the military and aerospace industries continues to grow as new sensor techniques are developed and the potential for active embedded sensor arrays increase. The research employs optical fiber sensors embedded in materials to monitor stress and fatigue. These sensors offer a means of material and structural evaluation from the manufacturing stage up to and including the active operational stage. As the sensor technology evolves, control systems using these sensors will require more computing power to meet the increased integration of multiple sensors and increased system I/O requirements. The computing requirements of smart structures are determined from the following criteria [1]:

- Number of Sensors x Sensor Bandwidth
- Number of Actuators
- Control Law Calculations
- Reliability
- Fault Recovery
- Low Power
- Low Weight
- Radiation Hardened
- Temperature Gradient Protection
- Application Flexibility

The level of control of the system will determine the exact computing requirements of a smart structure. Structures exhibiting a high level of control would be capable of reacting to their environment and ideally learn in the process. Presently, such work is in its developmental stages, although it is rapidly progressing with system applications discussed by Thursby and Grossman [2] and hardware advances by Maher et. al. and others [3,4,5]. Such systems normally require considerable computing power and memory. These high level control systems exhibit the characteristics of slow speed, slow I/O, fault tolerance, reliability, and multitasking [1]. Software technology applied to these systems would come under the category of Artificial Intelligence, Expert Systems or Neural Networks.

Systems exhibiting medium levels of control will lack the flexibility of the systems above, yet offer larger interfacing capabilities and less complexity in the software design. Characteristics of such systems include medium speed, medium I/O, reliability, and multitasking in PC type environments.

Systems exhibiting a low level of control require high speed, high I/O bandwidth, dedicated tasking, and minimum flexibility. These systems, called embedded systems, would include systolic arrays, parallel architectures, and implement new algorithms.

The smart structure system application, discussed herein, implements a Modal Domain (MD) sensor for Vibration Control [6]. A PC environment exhibiting medium I/O and speed provide a level of control that provided limited system performance [6]. This type of system places it with systems requiring a medium level of control. To increase system performance, maintain flexibility, introduce multitasking capability,the following system is developed.

The Real Time Embedded Control System (RTECS) implements a Reduced Instruction Set Computer (RISC) processor and it is designed to take advantage of the recently developed VLSI devices. It is capable of providing high I/O bandwidth, high throughput rate (MIPS),

low power and is reconfigurable. It is small and simple, but most importantly it is designed to add the element of flexibility not associated with low level controllers. This aspect is attributed to the programming environment. RTECS combines flexibility and speed with its multi-tasking capabilities to yield the speed of low level controllers with the development ease of medium level controllers.

This research investigates the potential of using RTECS in place of the IBM PC AT in the application of MD sensing for Vibration Control. A controlling program equivalent to the Basic program written by Cox et.al. on the IBM PC AT will be rewritten and executed on RTECS. It will show a speed increase, the ability to incorporate other functions and the potential for advanced development.

This thesis is organized as follows:

- The system design and operation with hardware considerations will be discussed in chapter 2.
- Digital control systems, considerations and a Model Development Procedure (MDP) is discussed in chapter 3.
- The application of RTECS is discussed in chapter 4 including the theory of MD sensing for vibration control, the system design using the MDP from chapter 3, system operation, and speed improvements.

# 2.0 System Description

The Real Time Embedded Control System can be described by hardware and software. It is the integration of the two in an environment that provides development ease and high speed performance. This chapter discusses the system considerations including the software language and development cycle times. This is followed with the operating system software, the host system software and then the hardware description.

## 2.1 Software Language Considerations-Forth

The motivation for the design of RTEC for intelligent structure applications is provided in chapter 1. Characteristics including high I/O bandwidth, high throughput (MIPS), radiation hardened, fault tolerant, reconfigurable, and low power in a small package are necessary to a smart structure application. But one characteristic, in addition to the hardware requirements, is the characteristic of flexibility for fast development cycles. The fact remains that embedded systems are designed by people to be used by people to perform tasks defined by people. The task of controlling an embedded actuator within an antenna support arm, or a moving a robot arm on the space shuttle require advanced algorithm development. But the embedded system must be simple, compact and ultimately as efficient as possible since its environment allows limited memory space, yet demands high speed response. From the programmers perspective, the issues of reliability and reconfiguration are directly related to the ability to develop the system effectively. Efficient, organized development will produce the simplest and most effective target system. Thus, a real time embedded system is not simply powerful hardware performing many high speed calculations, it is a system of software and hardware integrated gracefully and effectively to provide real time solutions in short development cycles.

Provided with this motivation, there is a question of an appropriate software language. Operating system development is dependent on the flexibility and ease of use of the system language. High level languages (HLLs) like Fortran, Basic, Pascal, or PL/M could offer the system developer an ideal environment. The advantages would be that the operating system is written in a language commonly known to all engineers, and therefore easy to understand, transfer, debug, and use. Development cycle time is therefore short. The drawbacks, however, are more numerous.

Memory space requirements of a HLL operating system are very large. Variable definitions, mathematical operations, and subroutine calls can all require a large amount of compiled code to execute. This has three implications. One, the actual hardware requirements of an HLL system are large, necessitating very high density design with exceptionally fast response time. The second is that the size of the operating system may be overwhelmingly large compared to the tasks it must perform. The execution performance of the controlling processor can thus become loaded down with the demands of the controlled system. Lastly, the efficiency of the system is directly dependent on the efficiency of the HLL compiler in its ability to produce compact, optimized code. Clearly, in the design of embedded real time control systems, the HLLs are not appropriate.

The other end of the programming spectrum is machine level programming. Machine language (ML) requires no compiling since it is used by the processor directly. Code written in ML is very compact and efficient and is capable of the shortest run time. The major drawback to ML is programming. Programming is very time consuming and error prone. Development cycle time is thus unacceptably long.

Another programming option available is assembly language programming. Assembly language (AL) programming uses mnemonics such as ADD, SUB, JMP to describe processor operation. Special programs called assemblers convert the AL to machine code for execution. The process uses more system overhead than ML programming and therefore assembled code is less efficient than ML, but it still is very fast. However, from the programmers perspective it is still tedious and error prone.

The alternative is a nonconventional language called Forth. It is a low level language like AL, but with HLL capability. It is comparable to Lisp and C, but it is really a language apart. It provides the programmer with a highly expressive language and an implicit model of program execution. Its execution speed is comparable to AL and its memory requirements are

modest. For these reasons and the fact that a new Forth microcontroller is available, RTECS is developed in Forth.

### 2.2 Development Cycle Time

Efficient operation of system software is as important as the time it takes to develop it. During application development, the RTECS operating system works in an interpreted mode or a compiled mode. Conventional systems operate in a compiled mode, where a program written in its entirety and complied before execution. An interpreted program however is translated on a line by line basis. An interpreted program is less efficient than a compiled program, but has the advantage that changes are easy to make and development time is reduced. A compiled program, on the other hand requires compiling changes before execution, but can run faster by comparison. RTECS allows for both compiled code execution and interpreted execution. Thus the development cycle for a target system is reduced and the target code can then be compiled and downloaded for fast execution.

### 2.3 Basic System Operation

The key to RTECS operation is the use of the parallel stacks. The stacks contain address and data information and are suitably named the data and return stack. For example, prior to any mathematical manipulations, the arguments of the expression are pushed onto the data stack. Mathematical expressions appear in Reverse Polish Notation (RPN) so that the last element placed on the stack is the first element off. So that " 3 4 + " will push 3 and then 4 on the stack, compute the sum and return 7 to the top. The character "+" is a previously defined word encountered in the system dictionary.

Another character ":" is a previously defined word that is used to define new words. Using previously defined words, new words can be defined to meet the specific needs of the

application. This is the real power of the system, since a program is simply a compilation of new words defined by the programmer. Each program threads its way through component words equivalent to the subroutine call of an HLL. Liberal use of subroutine calls is held to be good programming practice. Information for these calls is stored on the return stack. Every word evoked by the calling word will push a return address on the return stack. This is a process used by HLLs called a *stack frame*. The *stack frame* partitions memory, stores the state of the processor, and stores any necessary variables prior jumping to the subroutine. This is enormously inefficient. Nested subroutines only compound the effect. The RTEC saves memory and executes quickly, fulfilling the intelligent structure processing requirements. Specific features of the operating system are provided in the next section.

### 2.4 Operating System Software

The software environment of the RTEC is a full Forth operating system. All the facilities normally associated with a Forth programming environment are available plus a host of others. The Operating System Software was provided by Doug Jaeger [7].

The operating system resides in EPROM, which means that whenever the RTEC is powered on, the Forth Operating system is engaged. This is called the Kernel. Kernel features include:

- Double Precision
- Local variables
- Decompilation
- Macro Substitution
- Multitasking
- Single Step Tracing
- Variable Data Structures
- I/O file interface
- Memory Management

Within the Kernel is the monitor program. This is the feature that allows the RTEC to communicate with a host terminal or computer. Its features specifically include:

- Dump Registers
- Dump Memory
- Set Memory
- Load Target Image
- Set Break Points

An interesting element of the system design is the use of Shadow EPROM during system operation. Shadow EPROM is a term used to describe the concept of copying the contents of EPOM to the static RAM of the system during boot up. Transferring operation from EPROM to RAM effectively speeds the system operation considerably, since execution from RAM is much faster than from EPROM due to the speed limitations of dense EPROMs. This feature exemplifies embedded system design.

Lastly there is the Target Compiler. This is a program that operates on a host computer under a Forth operating system on the host (Mach 2 for the Macintosh, FPC for the IBM compatibles). The target compiler provides for generation of image files which can be downloaded to the RTEC. Also it can create Intel Hex files for burning EPROMs with permanent system software. Compiler features include:

- Dump Tables
- Decompiler
- Dump ROM
- Instruction List
- Compiler Directives

With these features, a typical application is developed in the following manner. Using the host, the application program is developed in Forth. Once operational functionality has been verified, the Target compiler is employed under the Host's Forth Operating System and the application is compiled to the target, in this case, RTECS. To test the application, RTECS is powered and a serial communications package provides the host/target interface. The Kernel is operable and provides a download command to accept the compiled code. Once downloaded, the application can be tested. Full debugging capabilities provide debugging at

the at level as the Host's debugger. Verification of successful operation is followed by the last step of the application development, to compile in a format suitable for burning EPROMs. This last step provides the embedded target with stand-alone operation.

One of the ways to test various processing systems is to run programs that perform a series of functions and compare run times. These tests are called benchmarks and are standardized to make fair comparisons. Five benchmarks for four machines are compared against RTECS in the table of Figure 2-1.

### 2.5 Hardware Description- Central Processing Unit

The following hardware description gives a brief discussion of the operation and capabilities of the devices used in RTECS. Block and functional diagrams are used in this chapter to describe general functionality. Schematics of the system are provided in Appendix A. This section provides the digital aspects of the design while section 2.6 and 2.7 discuss the analog aspects.

The central processing unit of RTECS contains all the processing devices of the system on a single 100 mm by 160 mm 6 layer PC board, see figure 2-2. The heart of the system is the RTX-2000 microcontroller. Serial interface to a host system is provided through two RS-232 serial terminal DB-9 connectors. Parallel communication to the host is possible through a Centronics type parallel port and a DB-25 connector. System bus interfacing is available with a VME type, 96-pin DIN 4162 connector. The signal designation is such to allow board to board connections through a VME J1 backplane bus. A 2-input terminal strip with RF filters, decoupling capacitors, and voltage limiting diode provide connection for stand-alone 5 Volt power. Manual Reset and Interrupt switches control the system. The system memory is provided by any size EPROM (64 Kbyte typical) and 128 Kbyte or 256 Kbyte Static Ram. A functional block diagram of the system is shown in figure 2-3.

| Processor | RTECS | 68020 | 68000 | 80386 | 80286 |
|---|---|---|---|---|---|
| CPU Speed (MHz) | 5.0 | 15.6 | 7.8 | 20 | 8 |
| Float Pt Processor | - | 68881 | 68881 | 80387 | 80287 |
| Integer Multiply | .19 | 2.90 | 5.50 | .90 | 2.70 |
| Sort Algorithms | 3.91 | 44.20 | 154.20 | 26.89 | 84.39 |
| Fibonacci | 12.96 | 83.70 | 264.00 | 42.48 | 120.96 |
| Sieve (16-bit) | 3.48 | 40.20 | 170.20 | 23.18 | 73.65 |
| Matrix + * T | 1.68 | 10.20 | 67.10 | 3.06 | 11.69 |

**Benchmark Definitions**

| | |
|---|---|
| Integer Multiply: | 16-bit x 16-bit integer multiply with 32 bit result. Performed 1,000,000 times |
| Sort Algorithms: | Sorts 8000 bytes with Quick, Shell, Heap sorts |
| Fibonacci: | Finds the first 24 Fibonacci numbers 100 times. |
| Sieve: | Finds first 1899 prime numbers 100 times |
| Matrix + * T: | Adds, Multiplies, Transposes 60x60 matrices |

Figure 2-1. Comparison of Benchmarks

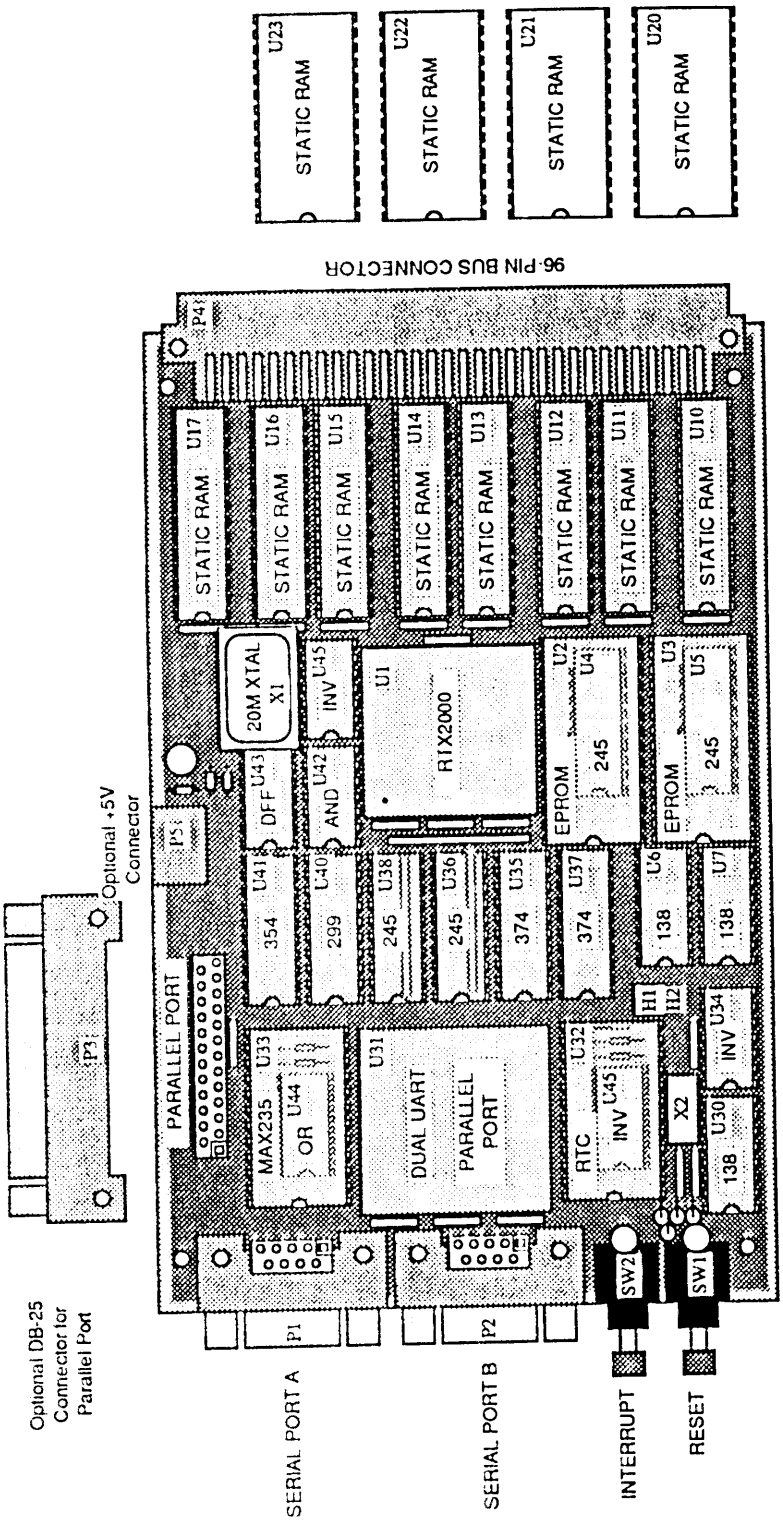
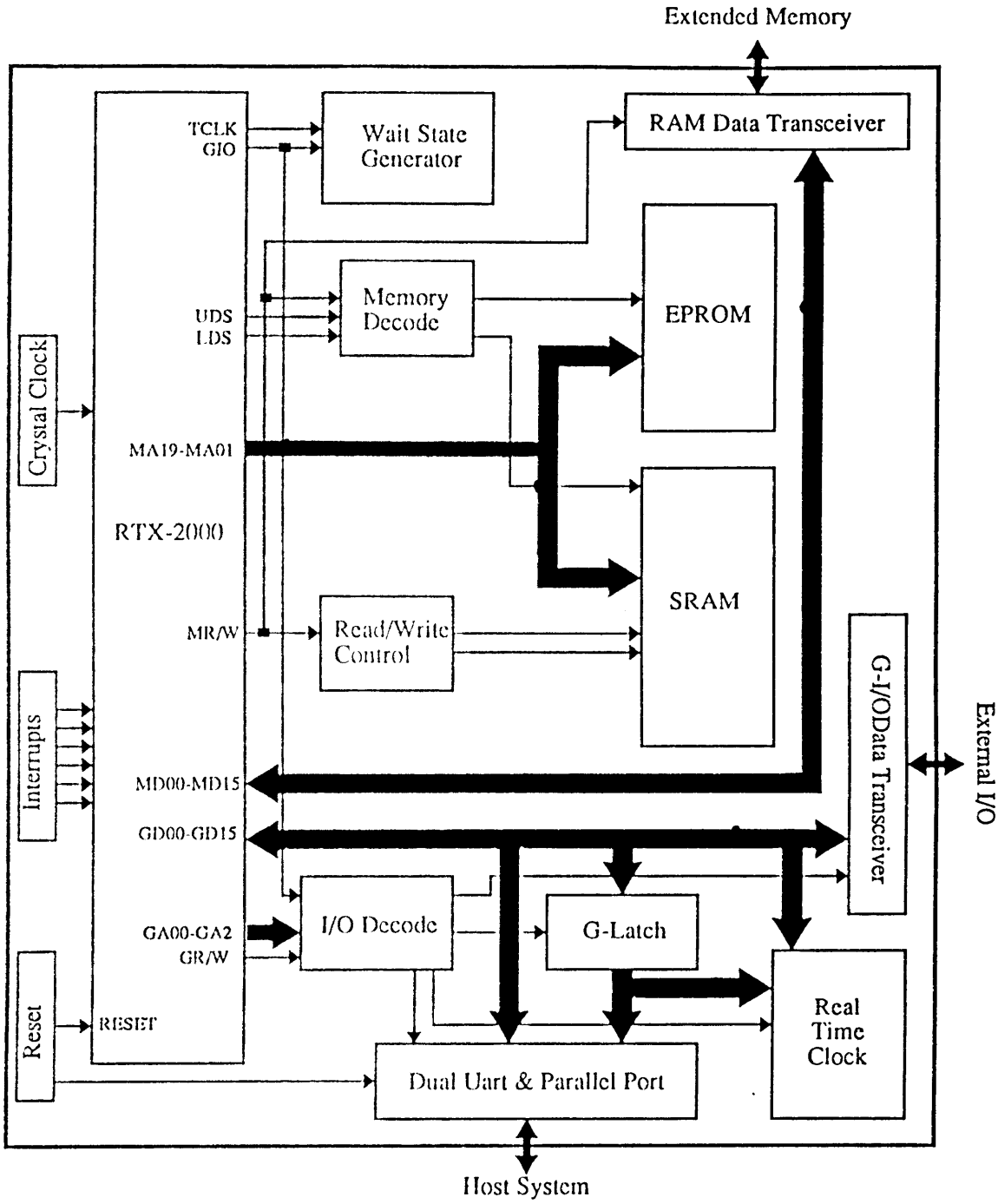



Figure 2-2. RTECS Board Layout


Extended Memory
RAM Data Transceiver
Wait State Generator
TCLK
GIO
Memory Decode
EPROM
UDS
LDS
Crystal Clock
MA19-MA01
RTX-2000
SRAM
MR/W
Read/Write Control
G-I/OData Transceiver
External I/O
Interrupts
MD00-MD15
GD00-GD15
GA00-GA2
GR/W
I/O Decode
G-Latch
Reset
RESET
Real Time Clock
Dual Uart & Parallel Port
Host System

### 2.5.1 System Processor

RTEC allows for direct execution of Forth with an interpreter/compiler on a microcontroller made by Harris Semiconductor. The Harris RTX-2000 is a 10 MHz, 16-bit microcontroller with an on-chip 16 x 16 multiplier, on-chip data and return stacks, on-chip interrupt controller and 3 timer/counters, see figure 2-4 [12]. It is capable of addressing up to 1 Mbyte of memory. Its instruction set is comprised of optimized Forth words. This instruction set is optimized since execution of up to four Forth words per cycle is possible. The instruction cycle sequence in figure 2-5 [12] shows the multi-instruction sequence executed during a single cycle. This is possible with an internal structure that is highly parallel organized around 4 separate data buses allowing simultaneous I/O and processing, see the table in figure 2-6 [12].

The arithmetic logic unit (ALU) is 16-bits wide and capable of most arithmetic and logical operations, see figure 2-7 [12]. An important speed feature is that the TOP and NEXT registers can undergo single bit shift operations and logical or arithmetic operations in the same cycle.

The dual parallel stacks as specified in section 2.3, figure 2-8 [12], are available and operational at the same time. The parameter stack is used to store data and pass subroutine parameters thus minimizing the overhead of subroutine calls. The return stack stores the return address of the calling routines with a 21 bit wide register for loop variables and memory page settings. It also allows for subroutine calls with a minimum of overhead.

### 2.5.2 Memory Interface

The memory interface, Appendix A, is the decode logic that allows the processor to access and read or write to the memory devices. Control is provided using 2 (3 x 8 ) 74F138 decoders. These devices are enabled by the Memory Read/Write (<<MR/W*), (<<BOOT), and

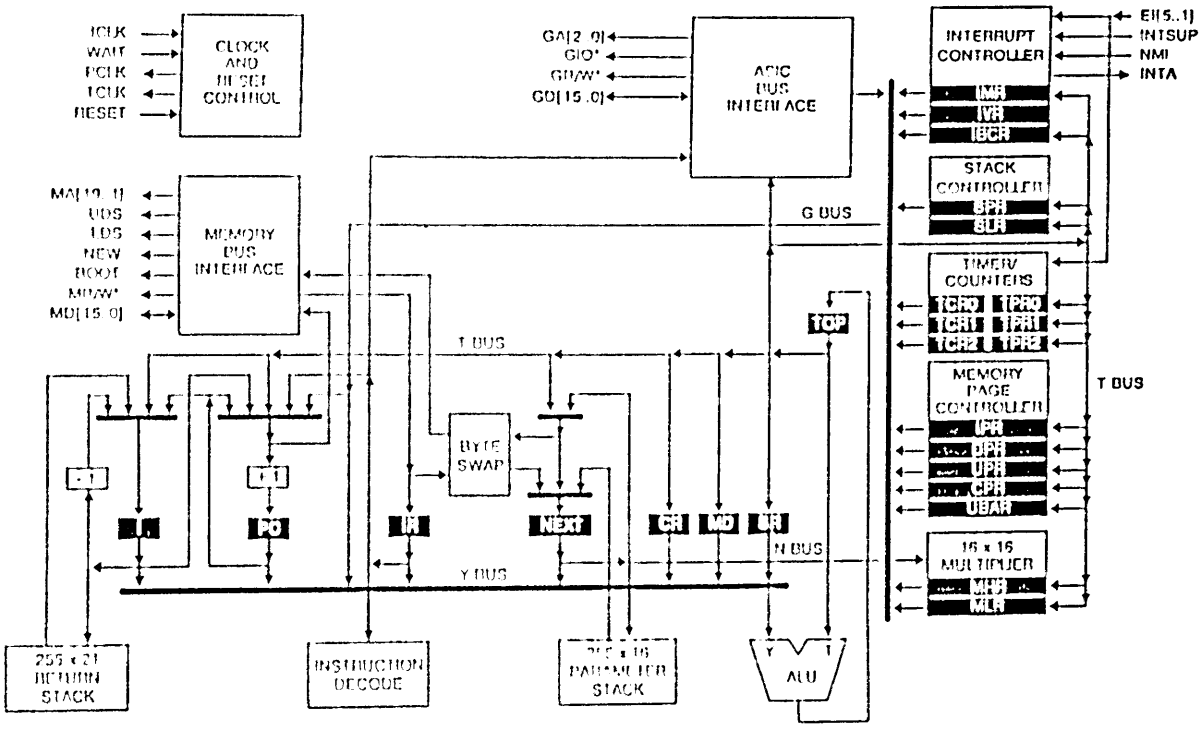



Figure 2-4. RTX-2000 Functional Block Diagram [12]

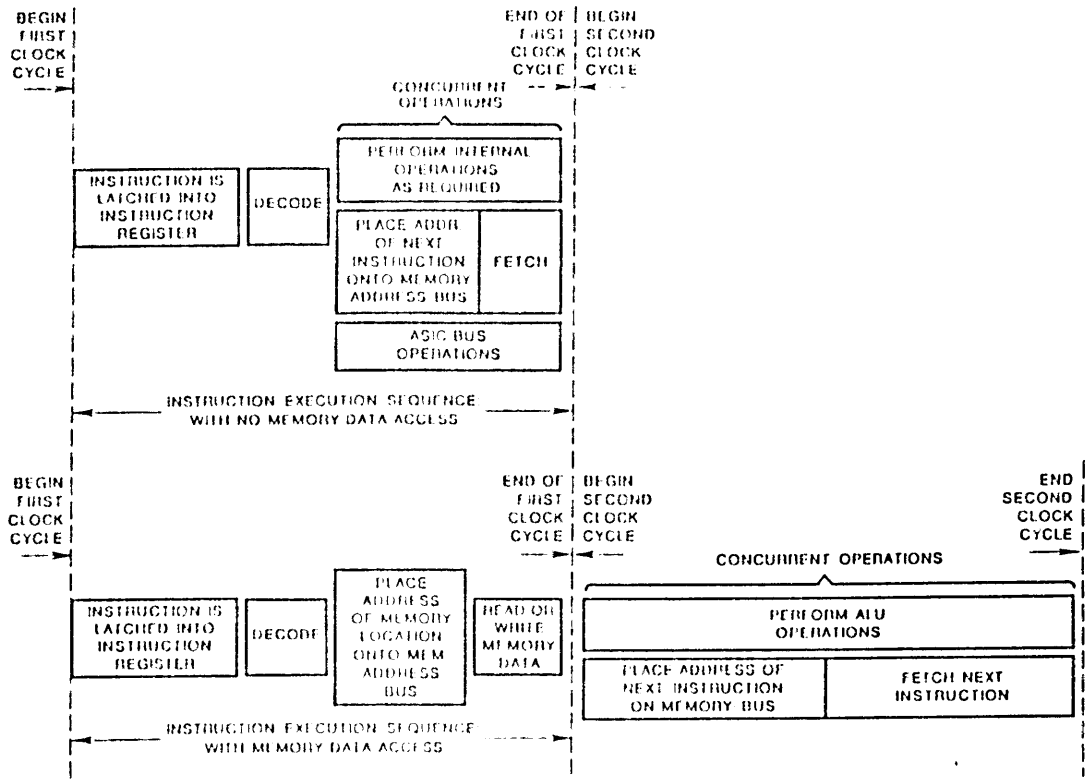



Figure 2-5. Instruction Cycle Sequence.[12]

| BUS NAME | CLASS | DESCRIPTION |
|---|---|---|
| Memory Data Bus | Primary | External; Carries data to and from Main Memory |
| ASIC Bus | Primary | External, Bidirectional bus formed when the T-Bus and G-bus are internally multiplexed; supports off-chip ASIC devices |
| Return Bus | Primary | Internal; Carries data to and from the Return Stack |
| Parameter Bus | Primary | Internal; Carries data to and from the Parameter Stack |
| T-Bus | Secondary | Internal; (TOP-Bus) Output from the TOP register; Connects to all internal registers and the ALU; Supplies ASIC Bus output |
| Y-Bus | Secondary | Internal; Second input to the ALU (the other is the TOP register) |
| G-Bus | Secondary | Internal; Input-only bus through which the core inputs data from internal registers and external I/O devices, including ASIC Bus input |

Figure 2-6. RTX-200 Data Buses.[12]

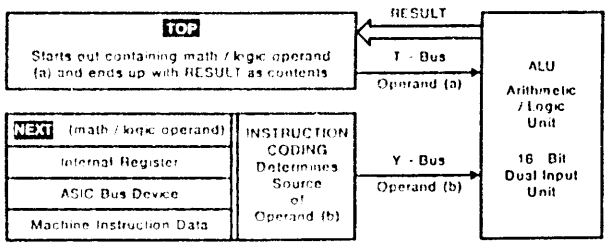



Figure 2-7. RTX- 2000 Arithmetic Logic Unit.[12]

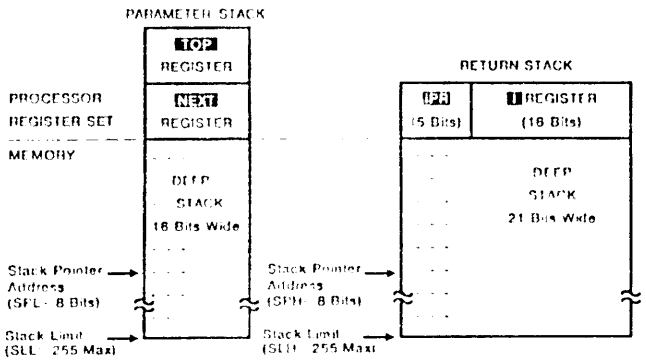



Figure 2-8. Structure of Parameter and Return Stacks.[12]

with the BOOT pin. The OE and WE extensions represent the output enable and write enable control of the memory devices respectively. These signals are provided separately from the memory chip select to allow for maximum speed according to the manufacturers specifications of commercially available Static Rams (SRAM's). This decoding scheme allows for the use of 35 ns static rams in the system and still guarantee full speed operation over temperature.

The flexibility for SRAM's packages of 32K x 8 as well as 64K x 4 is available where the only adjustment is in the placement of a jumper (H-1). Also, extended memory interfacing is possible since data buffers U5 and U6 are used to buffer <<MD00-<<MD15.

### 2.5.3 ASIC Port Interface (G-PORT)

The Applications Specific Integrated Circuit (ASIC) Port or G-port provides high speed parallel Input and Output with the RTX-2000. The G-port peripherals are selected and enabled using another (3 x 8) 74F138 decoder which is controlled by G-port I/O select (GIO*) and the Processor Clock (<<PCLK), see Appendix A. For off board peripheral interface like the A/D and D/A the CPU includes buffered G-port data lines <<BGD00-<<BGD15.

### 2.5.4 G-Latch

A specially designed latched interface is provided to communicate with the RTC, the DUARTPP, and the Wait State Generator. The G-latch is designed for two purposes. First, more than eight devices are desired to be accessed through the G-port since the RTX-2000 only provides 3 G-address bits. This offers a means to extend G-port devices using a linear selection scheme. Every bit of the latch is an additional address for a G-port device. The second purpose

for the latch is that it provides convenient 2 cycle access to slow devices that normally could not respond in 1 cycle.

G-port access of the devices listed above occurs when the first of two G-port write cycles stores a device enable word in the latches U35 and U36, see Appendix A. The second cycle actually selects the peripheral and then provides the read or write information. When the device access ends, a G-port write to the G-latch disables the device.

### 2.5.5 Real Time Clock (RTC)

The SGK Thompson MK48T02B real time clock is a clock circuit and a 2K x 8 SRAM backed up by a lithium battery with guaranteed operation of 11 years. This source of continuous memory and timing is useful in controlled power down conditions since system variables can be stored and retrieved. On the other hand, periodically stored state variables can be recovered when an uncontrolled power down occurs where system operation can continue from the last stored state. This is particularly attractive in stand alone embedded control applications.

### 2.5.6 Dual Universal Asynchronous Receive and Transmit and Parallel Port (DUARTPP)

To develop an application and access the Forth interpreter, RTEC interfaces with a host computer with one of three ports, RS-232 Serial Port-A, Port-B or the Centronics type parallel port. Important RTEC system objectives were met (small package and low power criteria) when all these functions were combined on a single integrated circuit, the Silicon Logic LD1108. It performs serial to parallel conversion transparently to the RTX-2000 providing an interrupt when a character is available. It also provides a parallel to serial output to a serial line driver. Its parallel port is accessed with the format providing Centronics type formatted output.

The serial interface requires signal levels that adhere to the EIA RS-232C standard. This is achieved with a charge pumped serial line driver. This single chip device, the MAX235, Appendix A, drives 5 RS-232 signals from TTL/CMOS inputs and receives 5 RS-232 signals providing TTL/CMOS outputs. The unit operates off a single 5V supply and does not require any external pump capacitors, a convincing advantage over other possible serial interface devices available for RTECS.

### 2.5.7 Wait State Generator

In the event that an external device can not respond in a timely fashion to a G-port or memory access, a wait signal is available to stop the processor clock until the responding device is ready. This signal is normally generated by the external device, but the RTEC has the ability to provide user programmable wait states as well. Wait states from 0 to 7 processor clock (<<PCLK) cycles are available. This is unique since the programmer can control access to slow devices and then change the wait count when device access is complete. This is another speed and flexibility criteria built into RTECS.

## 2.6 Analog Interface Considerations

The advantage of high speed digital computing for Smart Structures is to process analog information quickly and provide feedback for system control. The analog interface is then just as critical in its design considerations. The next sections describe the requirements of the digital to analog interface, including a discussion of the components used in the design of the A/D board. The next section 2.7 discusses the design.

### 2.6.1 Digital to Analog Converter (DAC)

Necessary to the implementation of the RTEC is the digital to analog interface. The digital to analog converter (DAC) accepts " n" binary digits and provides an analog voltage output as

$$V_0 = Vref ( B_1 2^{-1} + B_2 2^{-2} + ... + B_n 2^{-n})$$

where $B_1$ is the most significant bit (MSB) and $B_n$ is the least significant bit (LSB). A simple block diagram of a DAC system includes an accurate voltage reference, a switching network of resistors or capacitors (the DAC), and an output buffer, see figure 2-9. The DAC is itself composed of a network of precision weighted resistors or capacitors and a network of analog switches controlled by the input code. The binary input code determines the weight or load applied to a voltage reference resulting in a scaled version of the input voltage at the output.

The characteristics of a D/A converter most important to RTECS are resolution and speed. Resolution is a measure of the smallest incremental output change or a change in output for a LSB change in input. For example, an 8-bit DAC has 256 distinct output values which correspond to the number of different binary input values. A change in the LSB represents an output change of 0.4%.

Speed is a measure of conversion time and ultimately limits the system bandwidth. Speed is obtained with a pipelined internal switching structure. It is always the case that speed is a trade-off for resolution. So the system designer must critically evaluate system parameters to determine if the speed of a converter is fast enough for the resolution provided. Although speed and resolution are the first parameters considered, they certainly are not the only parameters. A list of parameters for DACs that applies to ADCs as well follows:

- Offset Error- The output of the DAC when a zero code is applied.
- Gain Error- A departure in the actual output from the design output.
- Linearity- A deviation from the straight line drawn through the endpoints of a DAC transfer function
- Differential Nonlinearity- The difference in actual voltage step size from the ideal 1 LSB step size.

To point out the advanced characteristics that advanced technology provides for RTECS consider the TRW TDC1012. The TDC1012 is a 12 bit monolithic DAC that uses a weighted array of current switches to obtain very high speed conversion times, less than 30 ns, see figure 2-10. The least significant six data inputs are used to directly control the six least significant binary weighted switches. The most significant six bits are decoded into 63 control lines each enabling a precision uniformly weighted current switch. These, along with the six binary weighted current switches form a current array capable of producing $(2^{12} -1)$ or 4095 distinct outputs. At any time, the current flowing in either output, (OUT+) or (OUT-), depends on the number of current switches on at that time. The exact current produced at the output is dependent on the stability of the voltage reference to the DAC. Since the reference is used, in this case, as a constant current source to the current switch array, the stability of the DAC is dependent on the stability of the reference. To provide the guaranteed accuracy of the device, $\pm$ 1/2 LSB, the reference must be stable to 1/2 $(1/2^n)$ or $1/(2^{n+1})$. For a 12-bit DAC, that is a stability of 0.0122% or 72 dB.

The other criteria that affects DAC performance is the stability of the load. Since this device is designed to drive 25 to 50 ohm loads, a DC servo- motor of that input impedance could be driven directly, but its input characteristics could vary as a function of the motor load or the motor operating temperature. This may appear to degrade DAC performance. If such is the

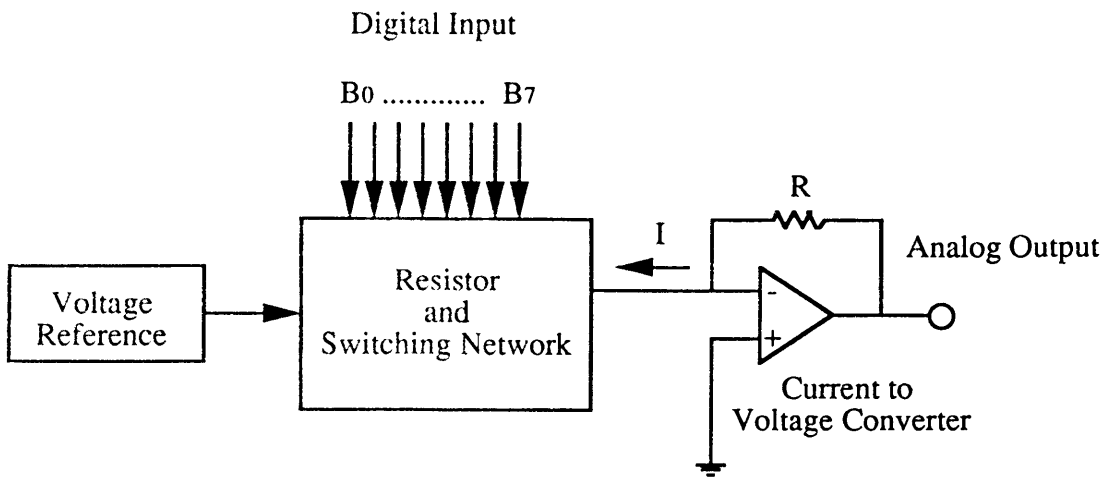



Figure 2-9. Digital to Analog Conversion System

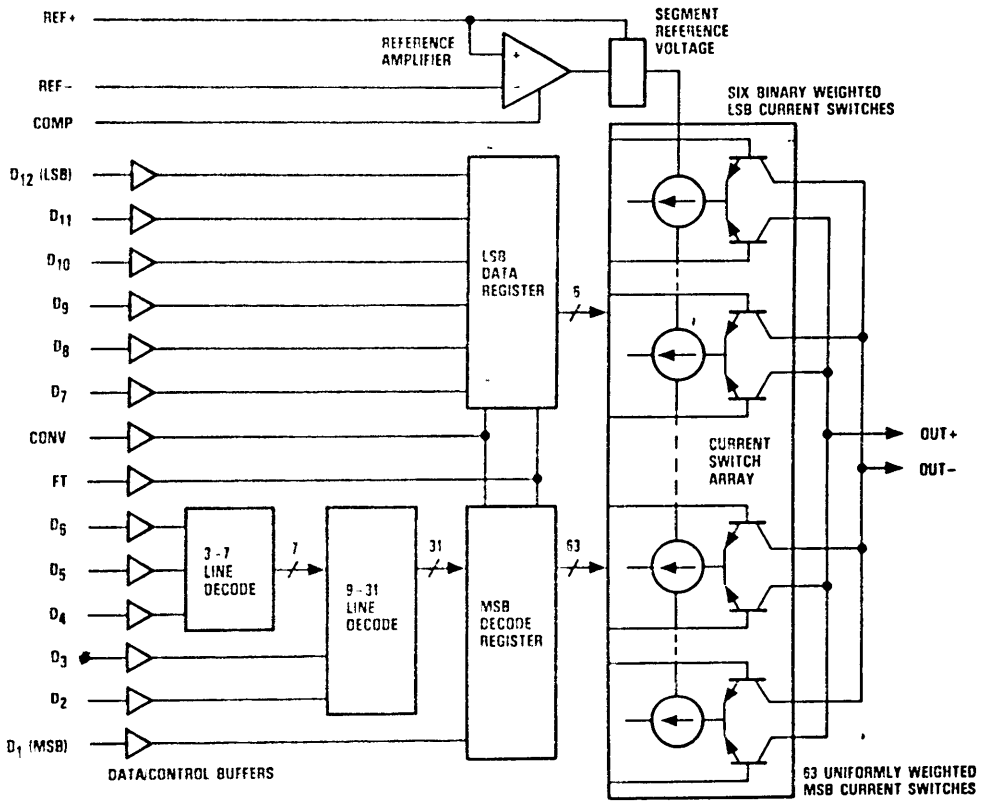



Figure 2-10. TDC1020 Functional Block Diagram.[13]

case, then a buffer, that has very stable output current characteristic over dynamic loads, must be provided at the DAC output.

### 2.6.2 Analog to Digital Converter (ADC)

As fundamental to the RTEC as a DAC is the ADC. The ADC is capable of mapping or quantizing a continuous signal into one of $2^n$ discrete ranges where " n" is the number of bits of the converter. Each range is assigned a representative binary output equivalent or code. The transfer characteristic of the DAC contains a linear division of the full scale analog input versus an increasing binary code. Figure 2-11 shows the transfer function of a 3-bit ADC[14]. The transition points or the threshold voltages ideally cause a change in adjacent binary output code. The difference in two adjacent threshold voltages is the minimum resolution of the ADC. The nominal value represented by any one binary output is the average of these voltages. Therefore the inherent accuracy of the ADC is limited by $\pm$ 1/2 LSB, which is the quantization error.

Other errors internal to the ADC that could affect the signal sampled by the RTEC are offset, scale factor, linearity and differential nonlinearity, see figure 2-12 [14]. It is essential in RTECS design that these error sources are not applied at the input of the processing system. Therefore, RTECS is designed with components exhibiting excellent characteristics over specified operating conditions.

The operation of the basic ADC is shown by using the successive approximation method, see figure 2-12. Notice that a DAC is inherent in the ADC design. Ultimately the conversion speed of the DAC will limit the ADC system speed. Thus, an ADC with the same resolution of a DAC will have longer conversion cycles. By figure 2-13, the conversion time for the ADC is

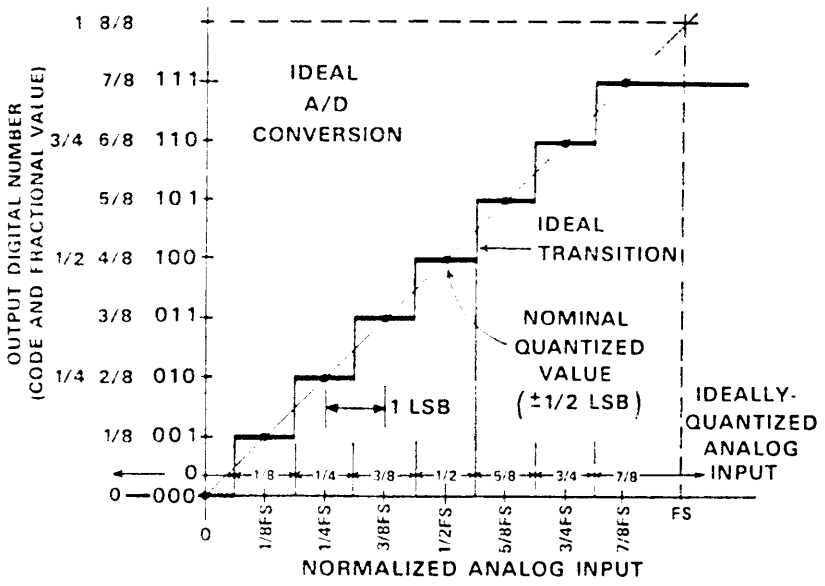



Figure 2-11. Conversion Relationship of an ideal 3-bit straight binary A/D.[14]

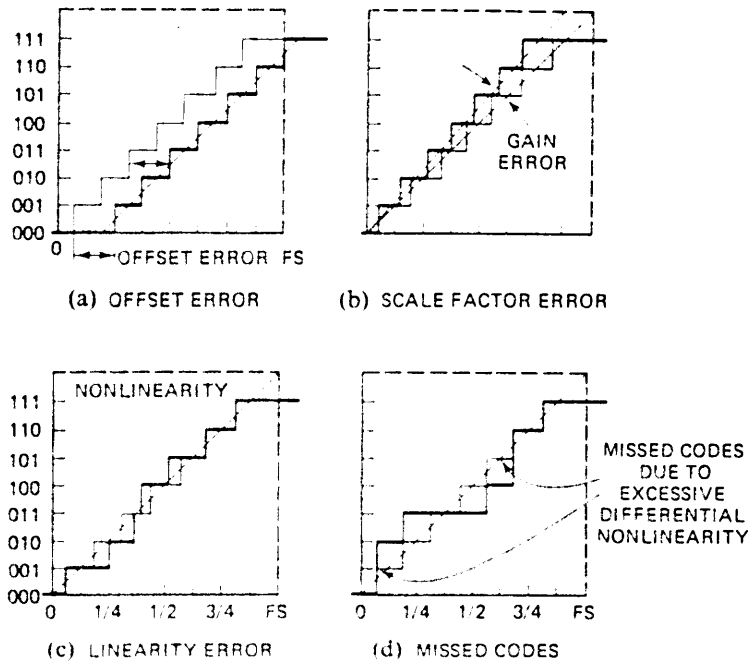



Figure 2-12. Errors of a 3 -bit A/D.[14]

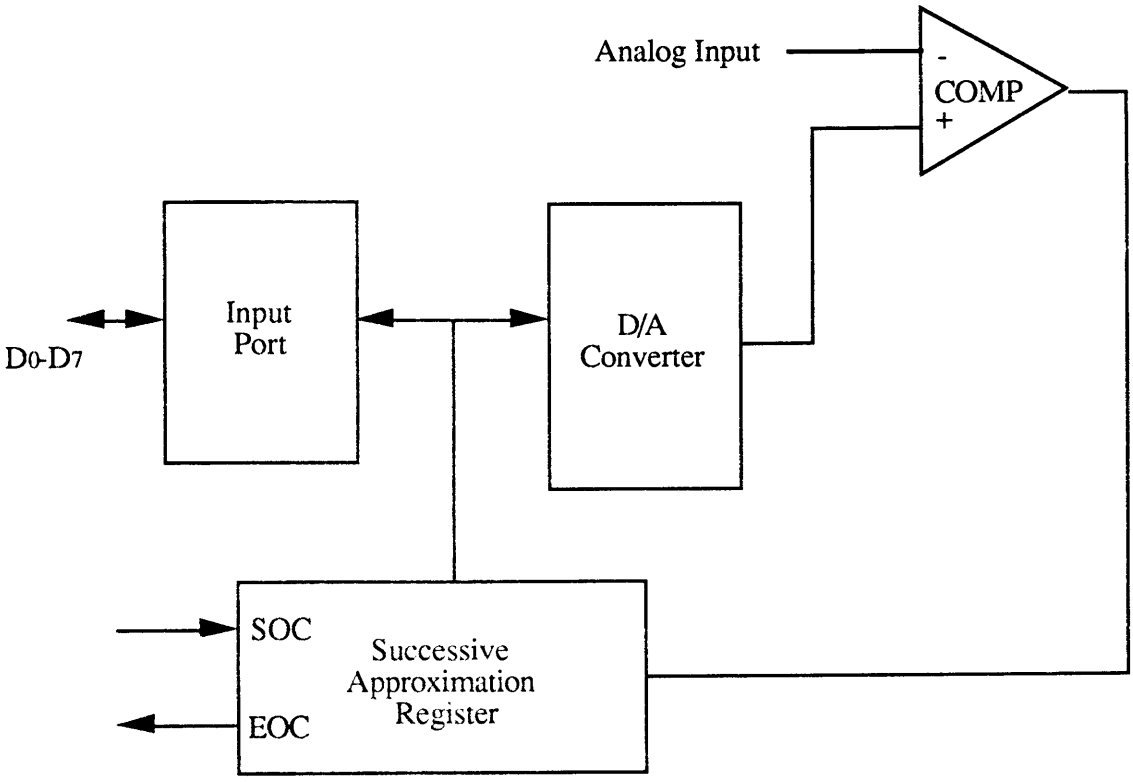



Figure 2-13. Hardware Approach to Successive Approximation Method.

$$T_{conv,ADC} = n*T_{conv,DAC} + n*T_{comp}$$

where $T_{comp}$ is the time required for the comparator to settle and the successive approximation register (SAR) to change output. To determine the ADC output, the SAR executes the following algorithm.

1. The MSB is set.
2. If the comparator output is 1, the MSB is reset, if not, it remains the same.
3. If all bits have not been tested, the next significant bit is set and step 2 is repeated, if all bits have been tested, conversion is complete.

Technological innovation in the manufacturing area provides RTECS with the advantage of much more advanced ADCs. In its design is the Crystal CSZ5412 12-bit ADC. The design implements a brute force conversion process not capable in years past due to the large number of analog and digital precesses required in one package. The technique used in its operation is called flash conversion where the analog input is compared to $2^n$-1 graduated voltage levels. The outputs from the $2^n$-1 comparators are processed and encoded into the proper format. An important note is that the accuracy and number of comparators double with each additional bit of resolution. Therefore practical flash converters are limited to 8 or 10 bits of resolution.

The two step technique employed by the CSZ5412 uses a track and hold (T/H) amplifier, a 6-bit flash ADC, a 6-bit DAC, and a differential amplifier, see figure 2-14 [15]. When a conversion is initiated, the T/H holds the analog input while the 6-bit flash ADC converts the 6 MSBs of the output. These outputs are decoded and latched. The 6-bit ADC output is fed back to the 6-bit DAC, where a signal that represents the analog input less the quantization error of the first six bits. The signal generated by the 6-bit DAC is subtracted from

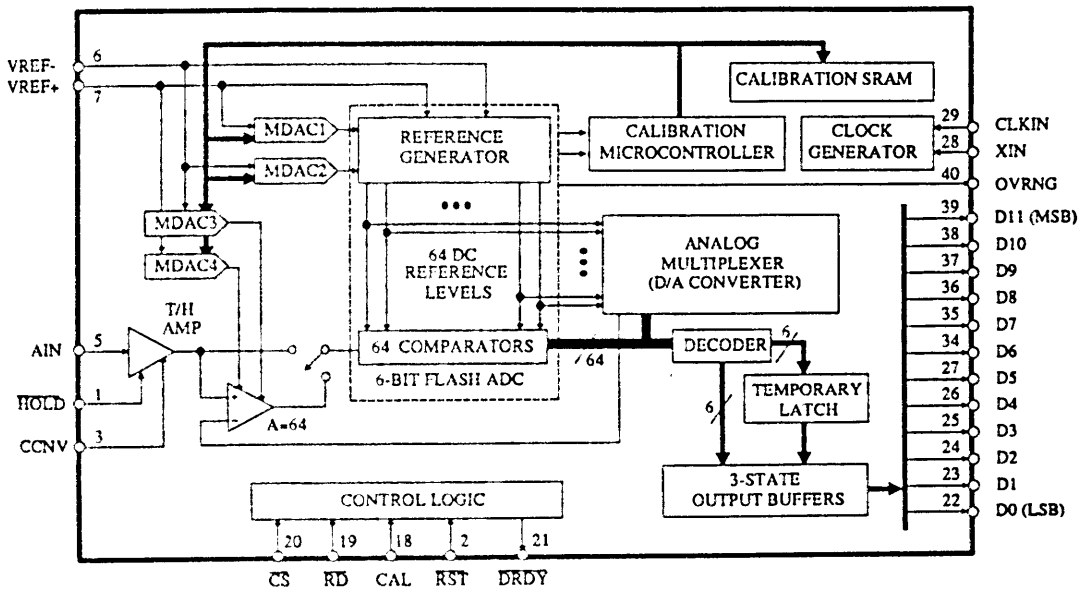



Figure 2-14.A Block Diagram of CS5412.[15]

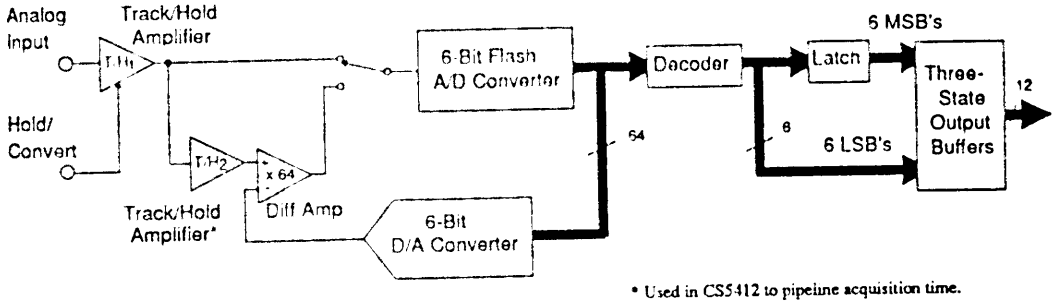



Figure 2-14.B Block diagram of the 2 step Flash Converter.[15]

the input and multiplied by $2^6$ at the differential amplifier before being applied again to the 6-bit ADC. The second conversion then yields the six LSBs.

The converter also hosts its own auto-calibration and auto-zeroing circuitry controlled by its own internal microcontroller. An entire of PC board of digital control and microprocessor and analog conversion circuitry placed in a single monolithic device. Capturing the advantages of modern manufacturing technology and utilizing them to the fullest potential is the goal and achievement of this system.

### 2.6.3 Track and Hold (T/H)

The T/H circuit, sometimes called the sample and hold (S/H), is capable of tracking an analog signal and, upon command, holding the signal input for use by an ADC. It provides an accurate and stable input while the ADC performs its conversion. On the other hand, the T/H can be used at the output of a DAC to minimize the glitches produced when the output changes by sampling the output after the settling time of the DAC has elapsed.

A simple S/H circuit is shown in figure 2-15. It provides buffered operation by isolating the control switch and hold capacitor from the input and output with a unity gain op-amp. This provides both a high input impedance to the analog input so that it will not load the signal it is tracking, as well as maintenance of the hold capacitor voltage in the hold mode. Operation is simply described as follows. The output voltage follows the input when the switch is closed, this is the track mode. Upon a change in the applied switch control voltage, the hold mode is engaged and the charged capacitor maintains the input signal seen as the switch was opened. Since the impedance seen by the capacitor is not infinite, the hold voltage level will decay or droop. The next stage must use the hold signal before the droop is greater than 1/2 the LSB of the system.

As is the case with the CSZ5412, many ADCs provide the T/H on chip, however, for RTECS design, more are required as the system uses multiple channels and simultaneous channel sampling is required. This is discussed in the next section.

### 2.6.4 Analog Multiplexers

RTECS is a multi-channel system, where digitally controlled analog multiplexers (MUXs) are essential. Employing switches and buffered inputs and outputs, these devices are available off the shelf in 4-bit to 16-bit packages. As seen in the CS31412, figure 2-16, on-chip S/Hs and a digitally controlled analog MUX are available in a single package[15]. The CS31412 can be operated as a 4 to 1 single ended input MUX or dual 2 to 1 differential input MUXs. It uses 4 track and hold amplifiers with on chip hold capacitors, an analog MUX and two output buffers. It is capable of true 12-bit performance which means that signal degradation is guaranteed to less than $1/2(2^{-12}) \times 100\%$.

One added advantage to the capability of multi-channel operation is the capability for offset and gain correction in the ADC should it occur. This is achieved by calibrating the system prior to data sampling. Applying a full scale reference and zero voltage reference via the MUX to the ADC, the multi-channel system can store the digital equivalent of these signals N(R) and N(0). When sampled data is input N(M), a processor based correction factor can be applied using

$$Vm = \frac{N(M) - N(0)}{N(R) - N(0)}$$

thus compensating for any linear errors introduced by the ADC.

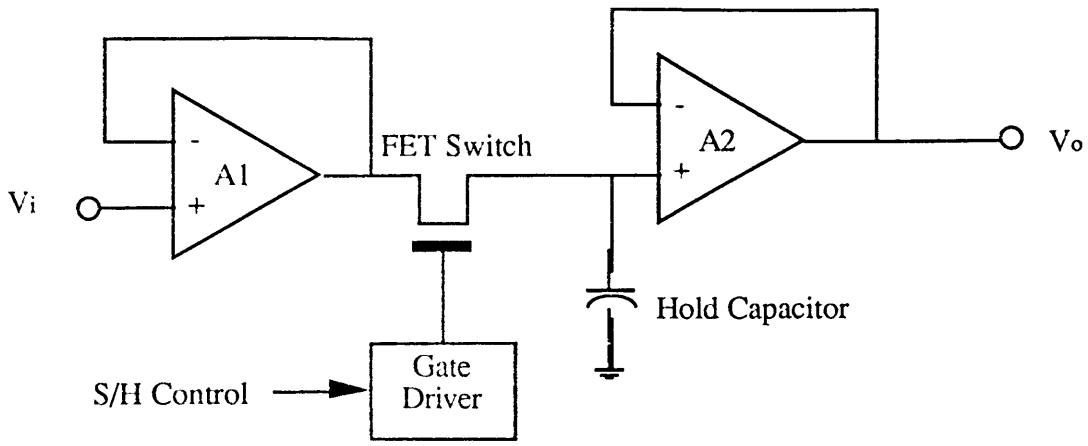



Figure 2-15. A Practical Sample and Hold.

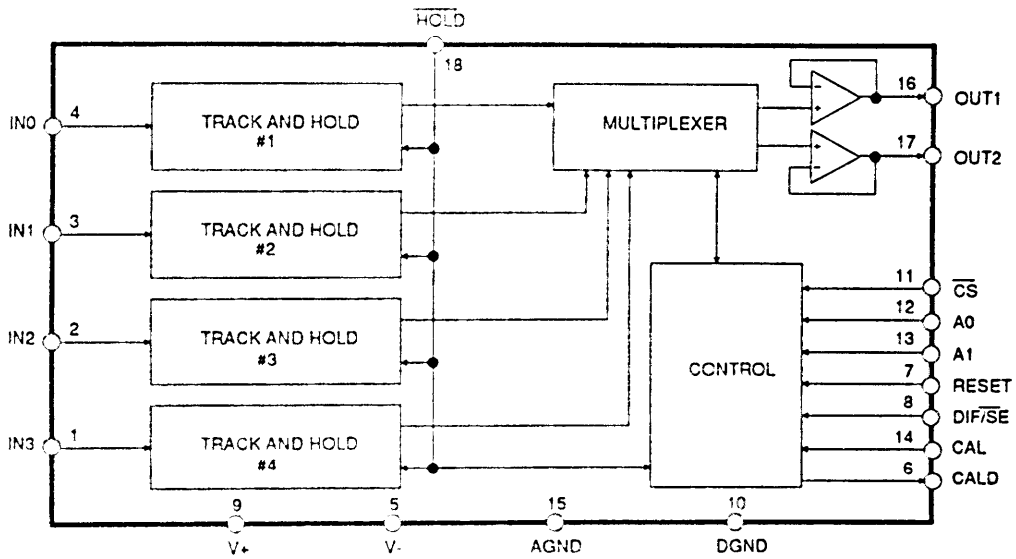



Figure 2-16. CS31412 Block Diagram.[15]

## 2.7 Analog Interface Design

The Digital to Analog interface was designed specifically for this project. The goal was to increase speed and implement more computationally intensive algorithms, and provide a digital to analog interface that would not limit system throughput. To accommodate this need was challenging in that the system resolution must be maintained and speed increased. As mentioned in section 2.6.2, speed and resolution are inversely related. The devices of the previous sections were used to fulfill the interface requirements of this project, see block diagram, figure 2-17, and Appendix A for detailed schematics.

In consideration of speed, the CS5412 A/D converter can be operated in continuous convert mode, its fastest mode. This means that the A/D converter is sampling at 1 MHz and providing digital output every microsecond. Even though this is much higher than the present requirements of this project, it is certainly not a limiting factor to advanced algorithm development. However, it is prudent to consider the timing relationship of the A/D read operation and the last sample time so that correct data is input to the processor. The data provided can be selectively input to accommodate the cycle time of the algorithm. Data selection is provided by the 4 channel sampling multiplexer. The CS31412 4 channel sampling multiplexer requires an external latch to hold control information. This latch called the A/D and D/A (ADDA) latch also contains the sample hold and channel selection control bits.

An important implication of this design is that the timing of the A/D and D/A interface is in the responsibility of the programmer. This has the advantage that the program limits are not going to be bound in hardware, rather they are only limited by the computational requirements of the algorithm. However, careful attention must be made to sampling times.

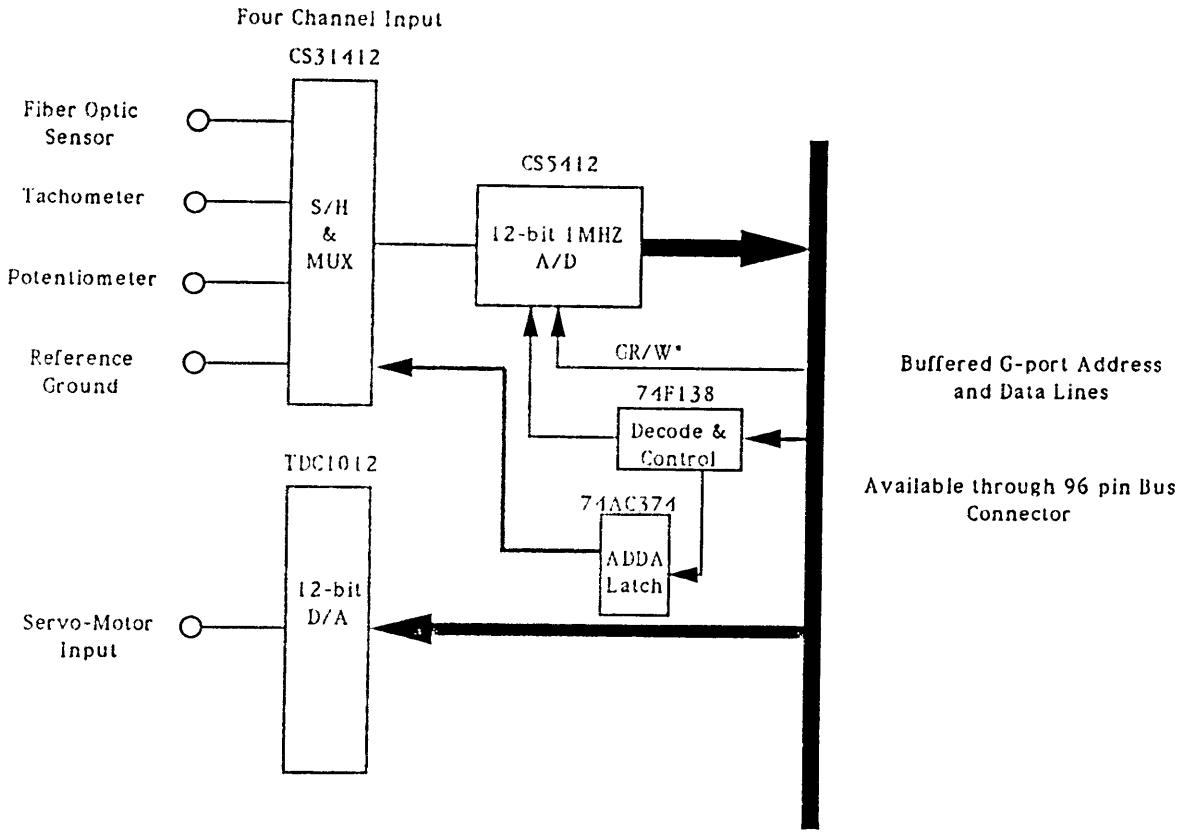



Figure 2-17. Block Diagram of the A/D and D/A interface Design.

# 3.0 Control System Design for Smart Structures

It is necessary to provide a model development procedure for digital control systems in order to maximize the utility and efficiency of the proposed Real Time Embedded Control System (RTECS). It will also provide the foundation for the specific application discussion of chapter 4 and it will provide the source for advanced system models that can take advantage of the enhanced capabilities of RTECS. The procedure begins with a brief discussion of the continuous state variable system with its relationship to the discrete state variable system. It is followed by a step wise procedure for transforming continuous time functions to discrete time functions and concludes with program development for implementation on RTECS.

### 3.1 Digitizing Continuous Time Control Systems

The control system is one through which the system excitations produce some desired response. The block diagram of figure 3-1 shows a continuous time control system with an actuator that provides excitations to the normally passive plant. The plant will respond in a predictable manner as a function of its parameters. When it fails to respond in a predictable or desirable way, application of some type of control is required. The control system can be classified as open loop or closed loop. The open loop system responds independently of its output. A closed loop system is one which the excitation depends on its response by the application of feedback. Figure 3-1 is a system with feedback applied. The effect of feedback is dependent on the generation of the error signal, E. This signal represents the difference between the input and the controlled output. It provides the information necessary to respond to random disturbances, to reduce sensitivity to inaccuracies in the system model, and to compensate for parameter variations.

Traditionally, the generation of the error signal is provided with differential analog elements such as mixers, multipliers, and difference circuits. Advances in the manufacturing of digital processors has increased the capabilities and reduced the cost of digital processing systems capable of providing error signals in addition a host of other capabilities. Figure 3-2 provides a digitized system block diagram that partitions the analog continuous time domain and the digital discrete time domain. Here the error signal is generated in the digital processor. The processor interface to the continuous time system is provided by analog to digital converters, discussed formally in section 2.6. The effect on the continuous time system model is

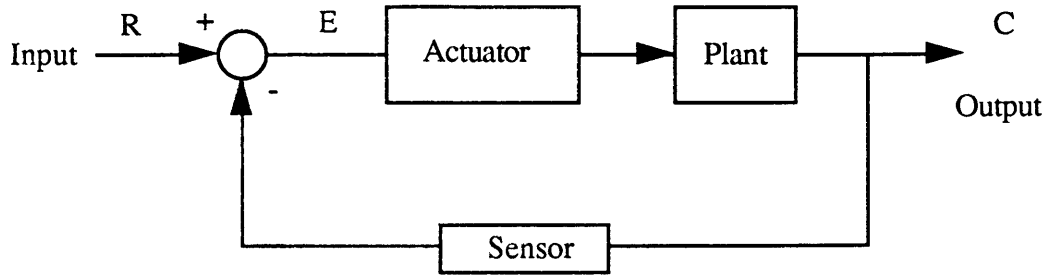



Figure 3-1. General Control System with Feedback

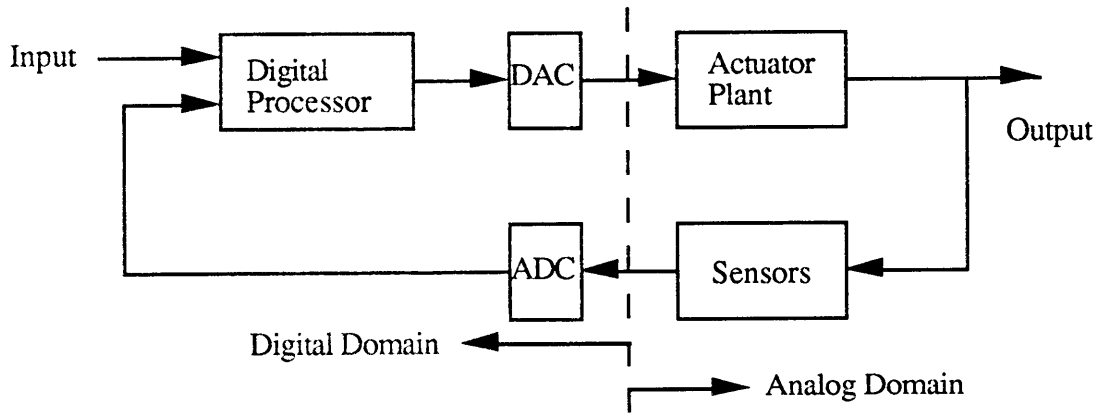



Figure 3-2. General Control System with Digital Interface

discussed in the next sections. The derivation of discrete models for execution on RTECS begins with the derivation of the continuous time state variables.

### 3.2 Continuous time state variable system

Since most control systems involve a continuous time plant, it is logical to begin the model development procedure with a brief development of a continuous time system. The state variable description of a system is of a form that can easily transition to discrete systems, therefore it is used in this procedure. Consider a linear, continuous time state variable model of the form

$$\frac{dx}{dt} = A\,x(t) + B\,u(t) \tag{3-1}$$

$$\frac{dy}{dt} = C\,x(t) + D\,u(t) \tag{3-2}$$

where $x(t)$ - is the state vector

$y(t)$ - is the output vector

$u(t)$ - is the input vector

A,B,C,D - specify system characteristics

The method of establishing the state model of a system is to first develop a set of simultaneous differential equations. The natural state variables derived from a set of scalar differential equations

$$y^{n}(t) + a_{n-1}y^{(n-1)}(t) + \ldots a_1 y^{(1)}(t) + a_0 y(t) = u(t) \tag{3-3}$$

are

$$x_1 = y, x_2 = y^{(1)}, x_3 = y^{(2)} \ \ldots \ x_n = y^{(n-1)}$$

so that the state model for Equation (3-3) is

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 \\ -a_0 & -a_1 & -a_2 & \ldots & -a_{n-1} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 1 \end{bmatrix} u(t)$$

$$y(t) = [\, 1\ 0\ 0\ 0\ \ldots .\ 0\,]\ x(t)$$

To obtain the general solution to equation (3-1) consider the scalar equation

$$\dot{x} = a\,x\ ,\ \ x(0) \qquad (3\text{-}4)$$

with solution

$$x(t) = e^{at}\, x(0) \qquad (3\text{-}5)$$

Extend this equation to vector equations with an exponential function $e^{A(t)}$ ( exp (At)) where A is an (n x n) matrix [8]. If equation (3-1) is pre-multiplied by $e^{-A(t)}$

$$e^{-A(t)}\ [\, \dot{x}(t)\ - A\ x(t)] = e^{-A(t)}\ B\ u(t)$$

the left side of the equation becomes an exact derivative. Therefore integrating from '$t_0$' to 't' yields

$$\int_{t_0}^{t} d/dt \{ e^{-At} x(t)\} dt) = \int_{t_0}^{t} e^{-At} B\, u(t)\, dt$$

$$x(t) = e^{A(t-t_0)} x(t_0) + \int_{t_0}^{t} e^{A(t-t)} B\, u(t)\, dt \qquad (3\text{-}6)$$

the general solution to equation (3-1). The output **y**(t) is simply a combination of solution **x**(t) and **u**(t). The matrix $e^{A(t-t_0)}$ used in equation (3-6) is called the state transition matrix with properties listed in reference [8].

### 3.3 Discrete Time State Variable System

To obtain a discrete time system from the continuous time system, the inputs are sampled and clamped. The effect on a linear time invariant system with general solution Equation 3-6 will be to consider x(t) only at sampled times. Evaluating x at times (kT + T) with $t_0$ = kT yields

$$x(kT+T) = e^{AT} x(kT) + \int_{kT}^{kT+T} e^{A(kT+T-t)} B\, u(t)\ dt \qquad (3\text{-}7)$$

letting t = kT+T-t and realizing the input to the system u(t) is clamped so that u(t) = u(kT) for $kT \leq t \leq kT + T$ then Equation (3-7) becomes

$$x(kT+T) = \{e^{AT}\}x(kT) + \{\int_{0}^{T} e^{At} B\, dt\}u(kT) \qquad (3\text{-}8)$$

$$x(kT+T) = F(T)\,x(kT) + G(T)\,u(kT) \tag{3-9}$$

By considering the outputs of the state model at the sample times yields

$$y(kT) = C\,x(kT) + D\,u(kT) \tag{3-10}$$

Thus (3-9) and (3-10) define a sampled equivalent discrete time state model.

To define equations (3-9) and (3-10) some important assumptions were made and must be justified. First, the input and output have an ideal sample and hold applied, see figure 3-2. Figure 3-2.A shows the analog to digital converter (ADC) providing constant inputs to the computer during discrete time intervals kT. At the output, the digital to analog converter (DAC) provides clamped signal levels to the system during the same interval. This discrete level output performs the function defined as a zero order hold (ZOH), figure 3-3.B. The important point to remember is that in a discretized continuous system, the sampling process only provides signal information at the discrete sample intervals. Since this information can actually be representative of any number of continuous time signals, the initial conditions and system signal bandwidth must be known.

To discuss discrete time systems in a convenient format, the z-transform is used. The z-transform of the sequence {x(kT)} is

$$Z\{\,x(kT)\,\} = X(z) = \sum_{n=0}^{\infty} x(nT)\,z^{-n} \tag{3-11}$$

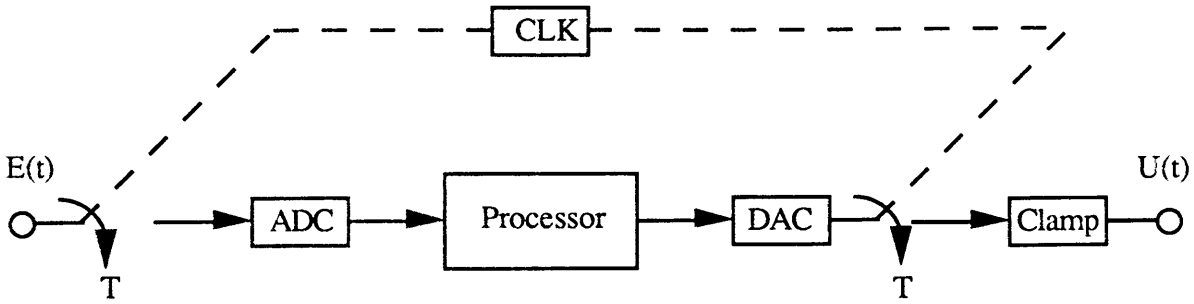



Figure 3-2.A S/H Hardware Perspective

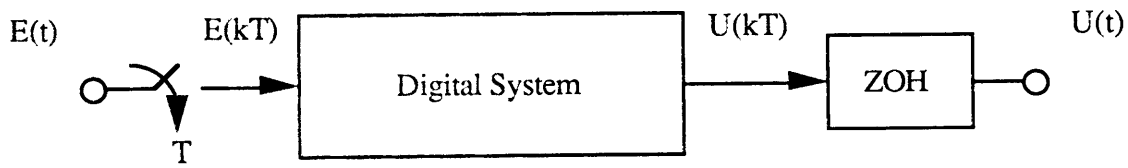



Figure 3-2.B S/H Functional Perspective

Recall that the counterpart of the z-transform variable in the continuous time domain is the the Laplace Transform S. The relationship between the two is

$$z = e^{sT} \qquad \text{where T is the sample period} \tag{3-12}$$

Basic properties and transform relationships of the z-transform and the s-transform are provided Kuo and Hugh [11]. These transform relationships are extremely useful for transformation of continuous time systems to discrete systems . Thus the background for continuous time and discrete time systems has been provided and a model development procedure is discussed next.

### 3.4 The Model Development Procedure

From the background development of the the continuous time/discrete time relationship, it is useful to have a procedure that gives a convenient stage by stage development of an expression suitable for execution on RTECS. The format of the transfer function of filters, controllers, and estimators is often an expression of the form

$$G(s) = \frac{C(s)}{R(s)} = \frac{(s-z_1)(s-z_2)....(s-z_m)}{(s-p_1)(s-p_2)....(s-p_n)} \tag{3-13}$$

The three stage procedure will convert an expression in the form of Equation (3-13) to a difference equation for high speed execution.

Stage 1. : Design the desired function in the frequency domain and provide in the form of Equation (3-13).

Stage 2. : Convert the parameters from the frequency domain to the sampled data domain.

Stage 3. : Convert the parameters from the sampled data domain to the time domain. This stage includes 5 steps described below.

Step 1. Rewrite the transfer function with inputs on the right hand side and the outputs on the left hand side.
Step 2. Multiply through to eliminate the denominators.
Step 3. Divide all terms by the highest order of z that appears in the expression.
Step 4. Replace each $z^{-n}$ with a time delay equal to n.
Step 5. Solve for the undelayed output term.

The best way to introduce the procedure is by way of brief example. A common application that is certainly a candidate for the compensator design in chapter 4 is a low pass filter. Stage 1 is defines the function. The expression for a low pass filter in Laplace domain is

$$H(s) = \frac{2\pi f}{s + 2\pi f} \quad \text{where f is the cutoff frequency} \qquad (3\text{-}14)$$

Letting f = 100 Hz the expression becomes

$$H(s) = \frac{628.3}{s + 628.3} \qquad (3\text{-}15)$$

Stage 2 involves transforming the continuous time expression in the frequency domain to a discrete time domain expression. Using the transform tables [9], the following substitution

| S-Plane Expression | Z-Plane Expression |
|---|---|
| $\frac{a}{s+a}$ | $\frac{z(1-e^{-aT})}{(z-e^{-aT})}$ |

is straightforward. One obvious decision required by the system designer is the value of the sample period T. The limits on the sampling rate are determined by two criteria, hardware speed and system bandwidth. The Nyquist criteria suggests a minimum value of two times the system bandwidth, but as a guideline, a value of at least ten times the system bandwidth is acceptable provided hardware is fast enough. Let f= 1000 Hz then T=0.001 sec. Substituting in this value and evaluating the expression in z completes stage 2.

$$H(z) = \frac{0.4665z}{z - 0.5335} \tag{3-16}$$

For complex functions the expression can be broken into partial fractions before conversion.

Stage 3 converts the z-domain expression to a time domain difference equation. Using the property that dividing by z has the effect of one delay, the objective is to rewrite the expression replacing each $z^{-n}$ with a corresponding delay of 'n'. The five steps are listed below.

Step 1. $H(z) = Y(z)/U(z) = 0.4665z/(z-0.5335)$

Step 2. $Y(z) \times (z-0.5335) = U(z) \times 0.4665z$

Step 3. $Y(z) - 0.5335 \times Y(z)/z = 0.4665 \times U(z)$

Step 4. $Y(k) - 0.5335 \times Y(k-1) = 0.4665 \times U(k)$

Step 5. $Y(k) = 0.5335 \times Y(k-1) + 0.4665 \times U(k)$

The expression in step 5 when coded on the RTEC will act as a low pass filter. The listing of such a program is provided in Appendix B. With a 25 Hz input function the output is

attenuated by 3% and it lags the input by a delay of 1 to 2 msec, corresponding to phase angles of 10° to 20°. An actual low pass filter should theoretically provide 3% attenuation and 14° of lag, which correlates well with this result.

### 3.5 Program Development for RTECS

RTECS is capable of executing programs at high speeds when the controlling program has been developed with the RTECS system in mind. The model development procedure concludes with a discussion of considerations for speed improvement for programs running on RTECS. Embedded system speed will be directly affected by the use of these programming techniques. The following are necessary to draw the most out of any algorithm developed for RTECS.

1. Utilization of the parallel stacks.
2. Utilization of scaled integer operations.
3. Limitation of the dynamic range of the program.
4. Utilization of the user defined variable space.
5. Utilization of the optimized instruction set.

The first and most important technique to incorporate in a controls program for RTECS is the use of the parallel stacks. Variable storage and retrieval can often be eliminated with use of the data stack to temporarily store data. This limits the overhead associated with the calculation of the address and the memory fetch cycle. Also, the parameter stack is available for the same function. However, the depth of the stack is limited to the amount of data that a programmer can effectively use and recall in the program development.This is especially true for algorithms incorporating nested loops. Any items left on the stack unintentionally, can

accumulate in a nested program and could crash the system. So prudent use of the stack is a most effective tool for control system development.

The second technique is the utilization of scaled integer operations. RTECS is an integer machine and thus floating point operations, although possible, are performed in software and require much computing time. Simple scaling routines like those found in the LPF example program in Appendix B, provide an effective and fast method for effectively moving decimal points.

The third technique is an important consideration usually overlooked by the inexperienced programmer. The dynamic range of a control system algorithm is determined by the accuracy or number of significant digits introduced, and the range of input and output values of values of an external interface. When the range is maintained throughout the algorithm, then unnecessary scaling can be avoided. This is very important as it affects the use of single or double precision calculations. If the system can be limited to 16-bits (4.5 decimal places) precision, then single precision can be used. Single precision calculations carry less than half the overhead of double precision calculations and substantial speed improvements.

The fourth technique employs a special feature of RTECS, the " user space." The user space is available on the RTX-2000 to access variables referenced often, and the access time is only two cycles. This is compared to the five cycle memory variable access. Extensive use of this capability will dramatically increase execution speed.

The final technique is truly a function exclusive to the RTX-2000. By organizing the program code in such a manner that optimized instructions are grouped, then an optimizing compiler can recognize and combine the instructions into one opcode. Examples of optimized instructions are memory access or G-port access instructions followed by ALU operations. The compiler can combine these instructions into one instruction and execute it in one cycle.

This finalizes the model development procedure. The techniques discussed are used in chapter 4, an application in structural control.

# 4.0 A System Application Model

This chapter incorporates RTECS in an application of Structural Control. The Project will be described from the model development stage through programming stage. The model development procedure is used to develop the application for RTECS. System considerations and a speed comparison are provided.

## 4.1 Introduction to Optical Sensing for Vibration Control

The issues in embedded control system applications discussed in chapter 1, are addressed with RTECS in the following System Application Model. The model application to be simulated with RTECS is the Smart Structure application by Thomas and Cox et. al. [6] "A Modal Domain Sensor for Vibration Control."

For this project, a fixed free beam is excited with a servo-motor actuator and monitored with a tachometer, for radial speed, a potentiometer for position, and an optical fiber sensor for distributed strain monitoring, figure 4-1. The optical fiber is used as modal domain (MD) sensor of vibration of a flexible beam. The modal domain sensor is used to suppress vibrations in the beam using a controller design technique developed by Celano and Lindner [10]. This project was completed in June 1989.

The conclusions of the project cited hardware limitations as the cause of limited experimental results. Upon discussion with the system developers, the overwhelming conclusion to system limitations was accurate model definition of the actuator, the flexible beam and the optical fiber sensor. This, in essence, contributed directly to the compensator design. A second order compensator design had been used. It was effective in compensating for the low order modes of vibration, but the analysis shows that higher order effects were contributing significantly. This suggests that a higher order compensator model would be more effective. Thus the motivation is provided for a processing device that can accommodate higher order compensator design.

The processing unit used by Thomas and Cox et.al. consisted of an IBM PC AT that used an Intel 80286 microprocessor, the Intel 80287 math coprocessor operating at 16 MHz clock. The

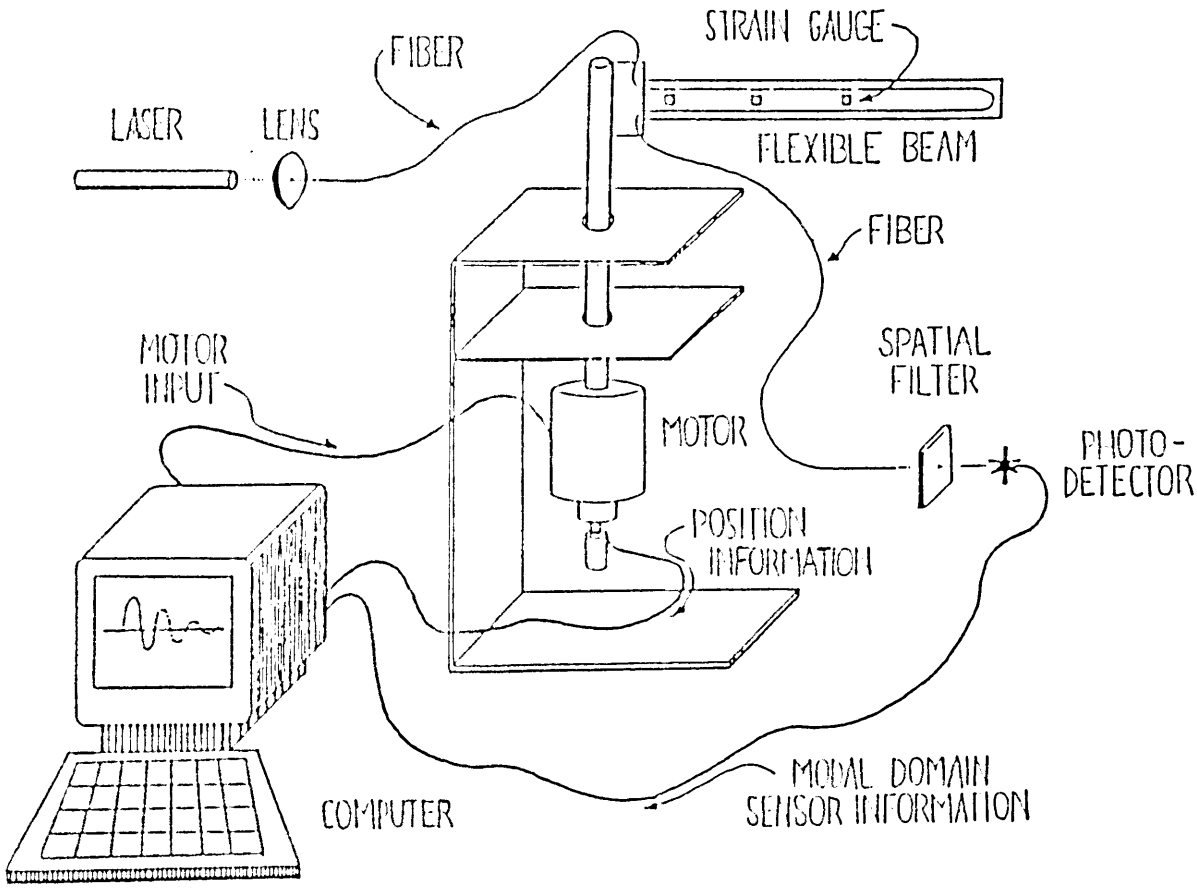



Figure 4-1. Vibration Controller Experimental Set-up.

system was also supplied with an A/D and D/A interface plug-in board capable of sampling four input channels with programmable gain applied. The plug-in board's maximum sampling rate was 400 Hz.

The programming environment consisted of Microsoft's Turbo Basic. This software package creates compiled Basic programs that can run exceptionally fast on the target machine, the 80286. An equivalent system is developed for RTECS. The model development procedure is used to derive the code for RTECS. The operation of RTECS will then be modeled on a Macintosh II running under MACH 2, a Forth development system. And then the code will be compiled and executed on RTECS. System speed comparisons are made in chapter 4.4 concluding with a discussion of speed vs. program trade-offs.

## 4.2 Compensator Model Development

Chapter 3.4 discussed the model development procedure for RTECS. Stage 1 requires development of a frequency domain compensator. To do this, each device model of the vibration control project is derived and then compiled together into one representative system. The system characteristics are analyzed and a compensator is then designed. The device models are provided briefly with the compensator design to follow.

The first device to model in this system is the fixed-free flexible beam. Under the limits that include linear equations for motion, with boundary conditions for a flexible beam with torque/actuation applied at the hub, and zero applied external forces, the vector matrix model is [8]

$$\begin{bmatrix} J_b+J_h & M_{rq}^{T} \\ M_{rq} & M_{rr} \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{r} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & K_{rr} \end{bmatrix} \begin{bmatrix} \theta \\ r \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & B_m \end{bmatrix} \begin{bmatrix} \tau_l(t) \\ f(t) \end{bmatrix} \qquad (4\text{-}1)$$

where $r = \begin{bmatrix} r_1 \\ r_2 \\ \cdot \\ \cdot \\ \cdot \\ r_n \end{bmatrix}$

The next device to model is the motor/actuator. The DC servo-motor is modeled with Newton's second law incorporating the sum of torques about a shaft, Equation (4-2).

$$M\begin{bmatrix} \ddot{q} \\ \ddot{r} \end{bmatrix} + D\begin{bmatrix} \dot{q} \\ \dot{r} \end{bmatrix} K\begin{bmatrix} q \\ r \end{bmatrix} = Q\, e_a(t) \tag{4-2}$$

The final device model is the modal domain sensor. The foundation for its operation is discussed in Butter and Hocker [11]. When applied to this system, the strain produced in the fiber equals the surface strain induced in the flexible beam then

$$I_f(t) = C_1 \begin{bmatrix} q \\ r \end{bmatrix} \tag{4-3}$$

where $C_1 = \begin{bmatrix} 0 & C_1^1 & C_1^2 & \cdots & C_1^n \end{bmatrix}$

with $C_1^n = K_0 \left. \frac{\partial y_n(z)}{\partial z} \right|_{z=l}$

A composite model of the system with all devices can be made by incorporating the the state space format

$$x = \begin{bmatrix} q \\ r \\ \dot{q} \\ \dot{r} \end{bmatrix}$$

$$\dot{x} = Ax + Be_a \qquad (4\text{-}4)$$

$$y = Cx$$

where $$A = \begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}D \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ M^{-1}Q \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & \cdots & & & & \\ 0 & \cdots & & 0 & 1 & 0 & \cdots \\ & & 1 & & & n & \\ 0 & C_1 & \cdots & C_1 & 0 & 0 & \cdots \end{bmatrix}$$

The first two rows of the matrix C correspond to the potentiometer (position) and the tachometer (velocity) outputs respectively [6].

Given this system model, the parameters of each device were identified through direct measurement, evaluation of frequency response, and evaluation of material characteristics. With this information the control law could be derived. Three nested loops were to be used: one for gain feedback of position, one for gain feedback of velocity, and one dynamic compensator to suppress beam vibration. The compensator was designed using the method of low order compensator design discussed by Celano and Lindner [10].

The low order compensator design below provided active compensation of three vibrational modes of the beam.

$$G(s) = \frac{(s+2)}{(s+7)(s+30)}$$

This second order compensator provided a 16% increase in damping of the first system pole and a 28% increase in the second pole [6].

With the frequency domain function defined, stage 2 of the model design procedure requires transferring the frequency domain expression to a sampled data domain. Table 2-2 in VanLandingham [8] provides for direct substitution.

Stage 1: $$G(s) = \frac{(s+2)}{2} * \frac{7}{(s+7)} * \frac{30}{(s+30)} * \frac{2}{7(30)}$$

Stage 2: $$G(z) = \frac{(z-.995)z}{(z-.9826)(z-.9277)(420.32)}$$

The next step is to transfer the z-domain expression to the time domain for coding. Using the 5 step procedure in Stage 3 results in the following:

Stage 3:

Step 1: $$G(z) = \frac{Y(z)}{U(z)} = \frac{(z-.995)z}{(z-.9826)(z-.9277)(420.32)}$$

Step 2: $Y(z)(z^2-1.910+.0116) = U(z)(.0024)(z^2-.995z)$

Step 3: $Y(z) - (1.910)Y(z)z^{-1}+(.0116)Y(z)z^{-2} = (.0024)(U(z) - U(z)z^{-1})$

Step 4: $Y(k) - (1.910)Y(k-1) + (.0116)Y(k-2) = (.0024)(U(k) - U(k-1))$

Step 5: $Y(k) = (1.910)Y(k-1) + (.0116)Y(k-2) + (.0024)(U(k) - U(k-1))$

### 4.3 Programming the Compensator for RTECS

The final stage of the model development procedure is to incorporate the optimizing techniques of chapter 3.5. The compensator program was incorporated in three passes. To describe the process, the Basic Compensator will be referred to as BC listed in Appendix C, while the Forth Compensator for RTECS will referred to as FC listed in Appendix D.

The first pass was intended to create a program with the look and feel of a Basic program. The goal was to provide the engineers of the BC with a program that was easy to follow from initial observation. In doing so, two features of the FC were required. First, RTECS is a 16-bit machine so that the range of values that can be used is 0 to 65,535 unsigned integers or -32768 to +32768 signed integers. To write FC so that it is equivalent to BC required double precision. The double precision routines added overhead to the execution time, but made the code exactly compatible with BC. Second, RTECS does not include capability to perform calculations with floating point numbers as does Basic. Therefore, number formatting routines were employed that provided scaled integers, scaled integer operations, and dynamic scaling of keyboard inputs based on the precision of the input number.

The second pass at the program took a more refined perspective. The Basic style which made extensive use of variables, compounded multiple memory access cycles which were very time consuming. This was reduced by making greater use of the stack and passing parameters through the stack whenever possible. Also, it became clear that the dynamic range of the BC is limited to the range of the input and the output. Ultimately the range of inputs is limited to 12-bits as is the output. With only minor alterations, this allowed not only the format and scaling routines to be eliminated, but the double precision routines as well. The range of values for each stage of program was considered and placed within the 16-bit range of RTECS. The results of the program remained the same.

The third pass implemented all possible sources of speed increase. Specifically all variables references that could be pushed onto the stack were done so. This further reduced the overhead associated with variable reference. Also the user space was employed. This is a feature that allows a limited number of frequently referenced variables to be accessed in 2 cycles instead of 5. Finally, the word definitions were ordered such that an optimizing

compiler could combine certain Forth words into one cycle. This was applicable to variable fetches followed by ALU instructions.

### 4.4 Results

Figure 4-2 shows the model of the RTECS program operation as it was tested and timed on the Macintosh 68020 and then as it ran on RTECS itself. The Turbo Basic program was supplied for comparison and obviously does not contain any optimization. The version of the compensator program tested on RTECS was slightly different than the one derived in section 4-2. It incorporated a higher order compensator (fourth order) and a number of features including averaging, a low pass filter, and input shaping. To test RTECS on the most recent version of the Basic Compensator was the necessary as it would provide for fair comparison on the most advanced model. The algorithm tested is shown in figure 4-3.

### 4.5 Speed vs. Program Trade-offs

Complexity and size of the compensator or filter design will directly affect the system bandwidth. However, RTECS is an advanced system that utilizes simultaneous I/O and ALU computations. If the program is carefully designed, with consideration of the dynamic range of the system, consideration of accuracy, where optimized instruction execution used, then program execution speed can be improved and memory requirements can be reduced. This is the goal of the Real Time Embedded Control System.

It must be pointed out that RTECS was tested in its 5 MHz configuration and it is possible for system operation at 10 MHz. The affect would be to reduce the time for execution of the third pass to .375 seconds. So the speed improvement is greater than 20 times the PC based system. This provides the ability to add more filters, incorporate additional averaging, or simply increase the sample rate.

| | | | |
|---|---|---|---|
| CPU | 68020 | RTECS | 80286 |
| Float Pt processor | 68881 | - | 80287 |
| CPU speed (MHz) | 15.6 | 5.0 | 16 |
| Operating System | Mach 2 | RTEC-OS | Turbo-Basic |
| ---------- | ------- | ---------- | ---------- |
| First Pass (sec) | 24.0 | 12.7 | 8.0 |
| Second Pass (sec) | 15.93 | 2.6 | -- |
| Third Pass (sec) | 11.2 | .75 | -- |

Figure 4-2. Test Results

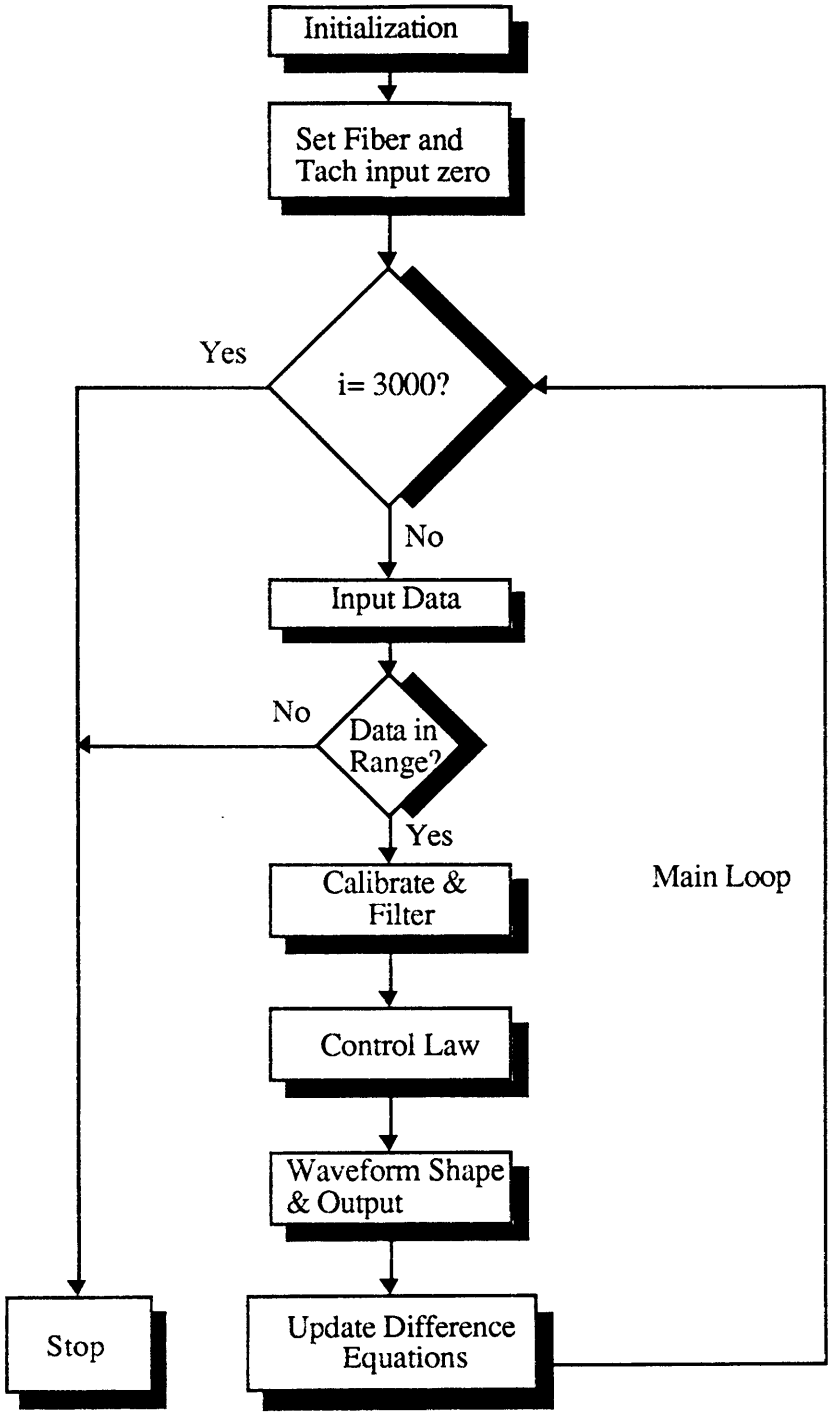



Figure 4-3. Program Flowchart

It should also be pointed out that to maintain the sampling rate of the BC system, that " No Operations " (NOPs) could be used, however, other tasks are possible. For example, a multitasking system like RTECS could be gathering information from other sensors, or maintaining system calibration by initiating calibration cycles in the interface devices. In an embedded system application where stand-alone operation is necessary then this certainly a useful task.

# 5.0 Conclusion

This research has produced a system design ideal for the computing environment of Smart Structures. The Real Time Embedded Control System (RTECS) has a number of characteristics desirable in Smart Structure Applications. It has combines I/O with data processing to increase system bandwidth, it incorporates the speed and efficient design of RISC, its power requirements are small , and its size allows placement in most aerospace or military applications. The system is capable of operating at 10 MHz exhibiting throughput rates of 15 MIPS with burst rates to 40 MIPS. This is achieved in a PC design of 100 mm by 160 mm that draws only 1 Watt of power.

The application of RTECS in this thesis utilized a model development procedure. It provided a step-wise development to transform frequency domain transfer functions to a form suitable for programming on RTECS. Also techniques were introduced to increase speed and thus system throughput. The model development procedure was used in an application of Modal Domain Sensing for Vibration Control. A computing environment functionally equivalent to a PC based system was executed using RTECS. The results showed an increase in operating speed from 8 seconds. on a 16 MHz PC AT to .75 seconds. using RTECS operating at only 5 MHz. The system bandwidth improvement was within that predicted from benchmark comparisons that

include average speed improvements of of 10 to 100 times that of an IBM PC AT with a 80286 processor. However, this is not the limit since the optimizing compiler used with RTECS did not include the ability to optimize all possible instruction sequences. Thus leaving the possibility of even faster program execution.

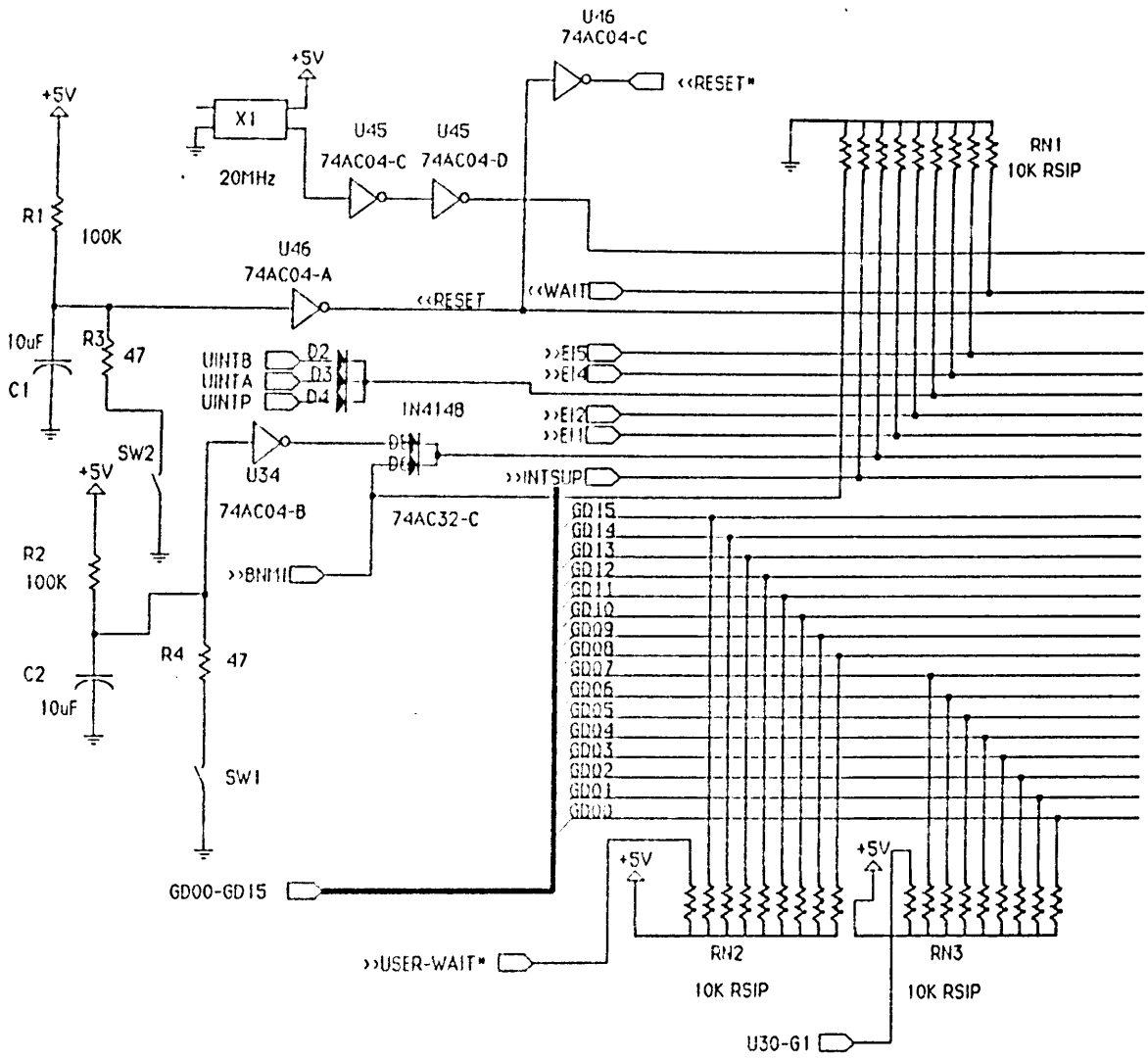
Recently, further increases in the RTECS processor speed have been made proposed. Harris Semiconductor has announced the RTX-2000A. With same functional capabilities of the RTX-2000, the RTX-2000A uses a new fabrication process that allows the microcontroller to operate at 20 MHz. This improvement gives RTECS a two-fold increase in system bandwidth. The present problem with such a system is that the Static RAM must be developed to respond in the decreased memory cycles. Static RAM manufacturers presently provide small density memory that decode at such speeds, but RTECS memory requirements are outside this range.

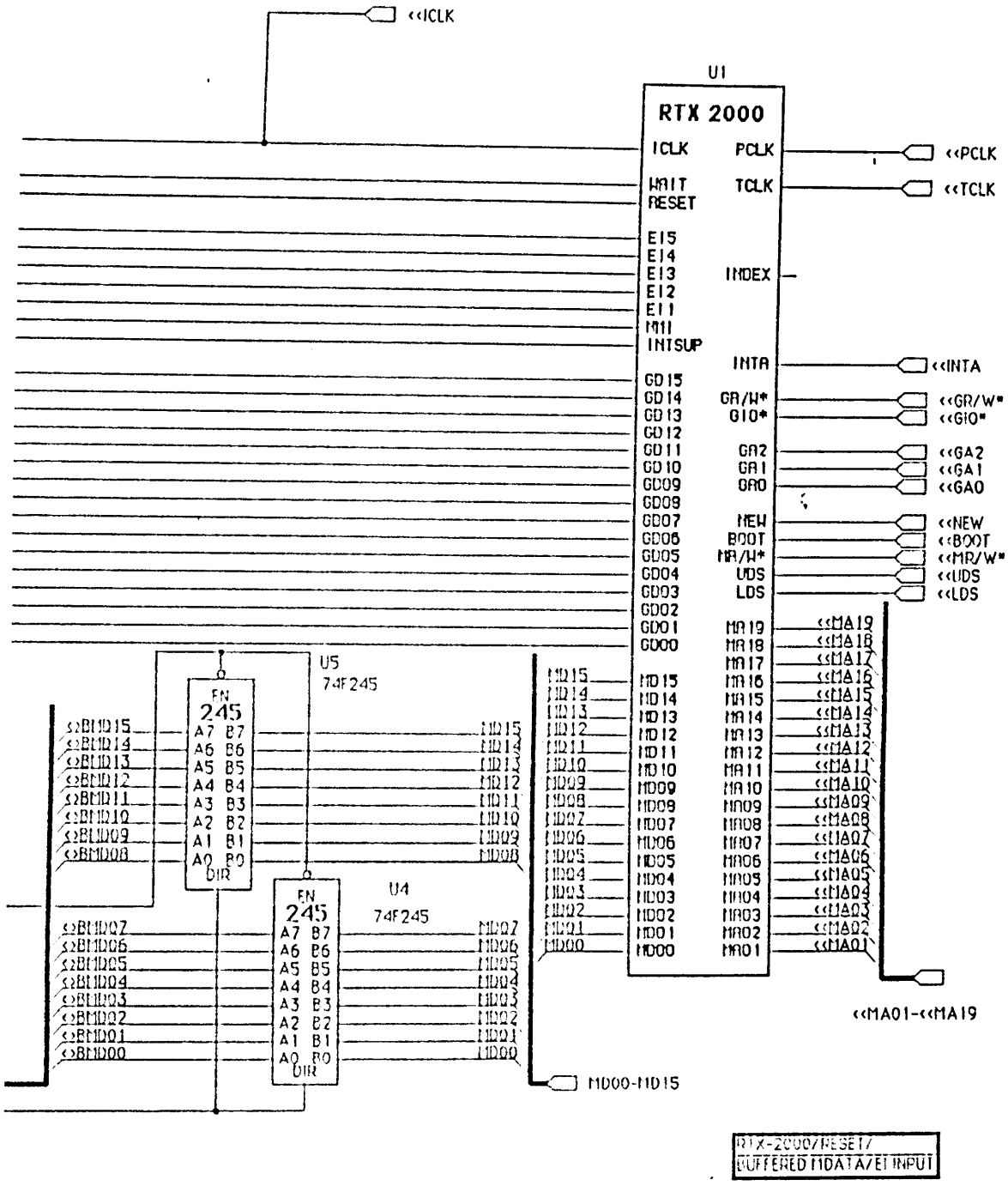
The overall reductions in program execution time and the flexibility of system design achieved with RTECS have made advanced algorithm design possible. Suggested future research includes higher order compensator design of the modal domain sensor for vibration control. System bandwidth of RTECS will allow eighth order compensator design and still maintain I/O capabilities. This system can be targeted for stand-alone operation, allowing for vibration suppression caused by external sources. This would provide an experiment of a smart structure reacting to randomly occurring external events. Also investigation is suggested in vibration suppression with RTECS using shape memory alloys as actuators. The combination of embedded fiber optic sensors and embedded shape memory alloys using a high speed embedded processing system for control provides motivation for continued research.

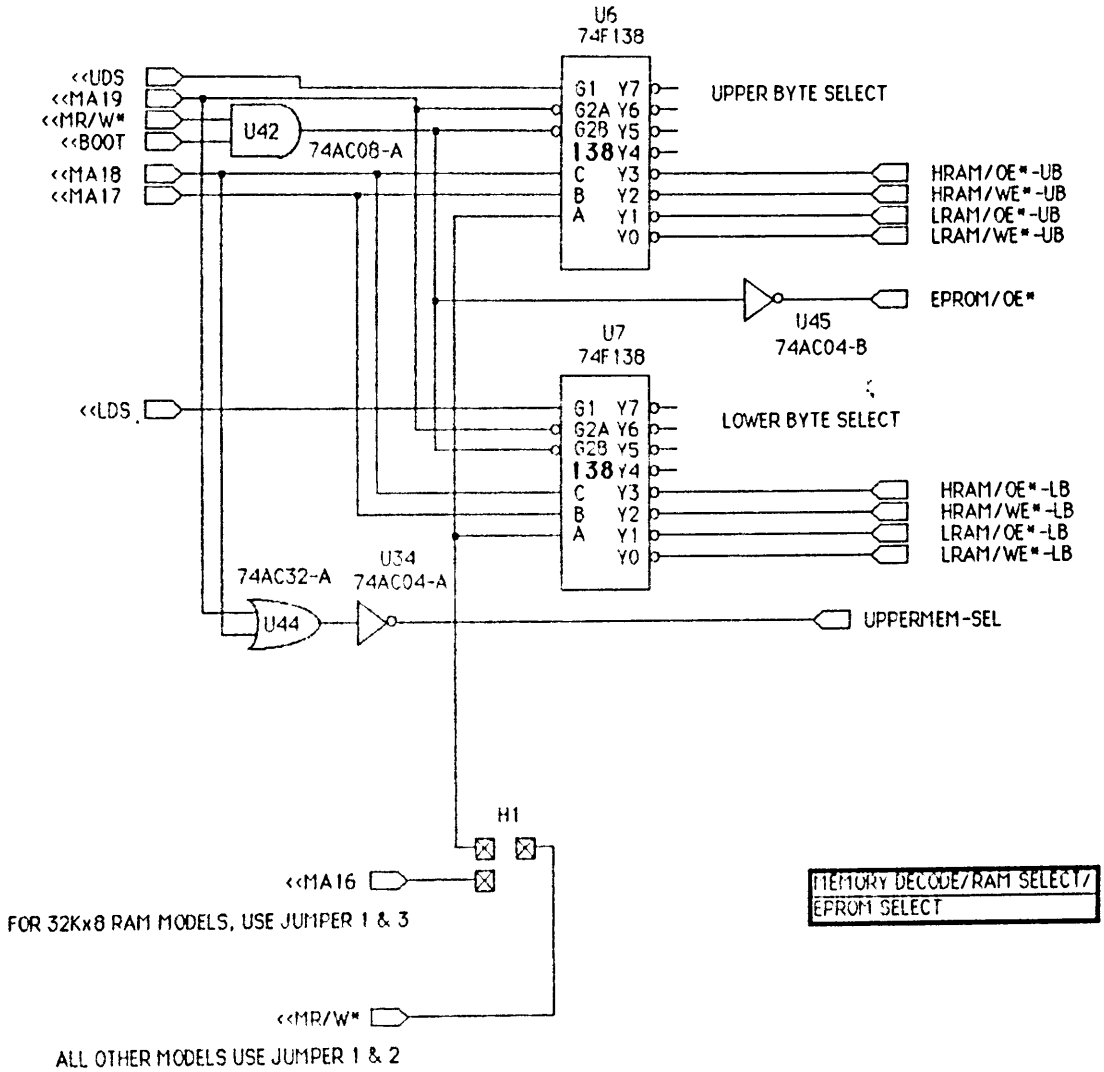
# Appendix A. Schematics for RTECS

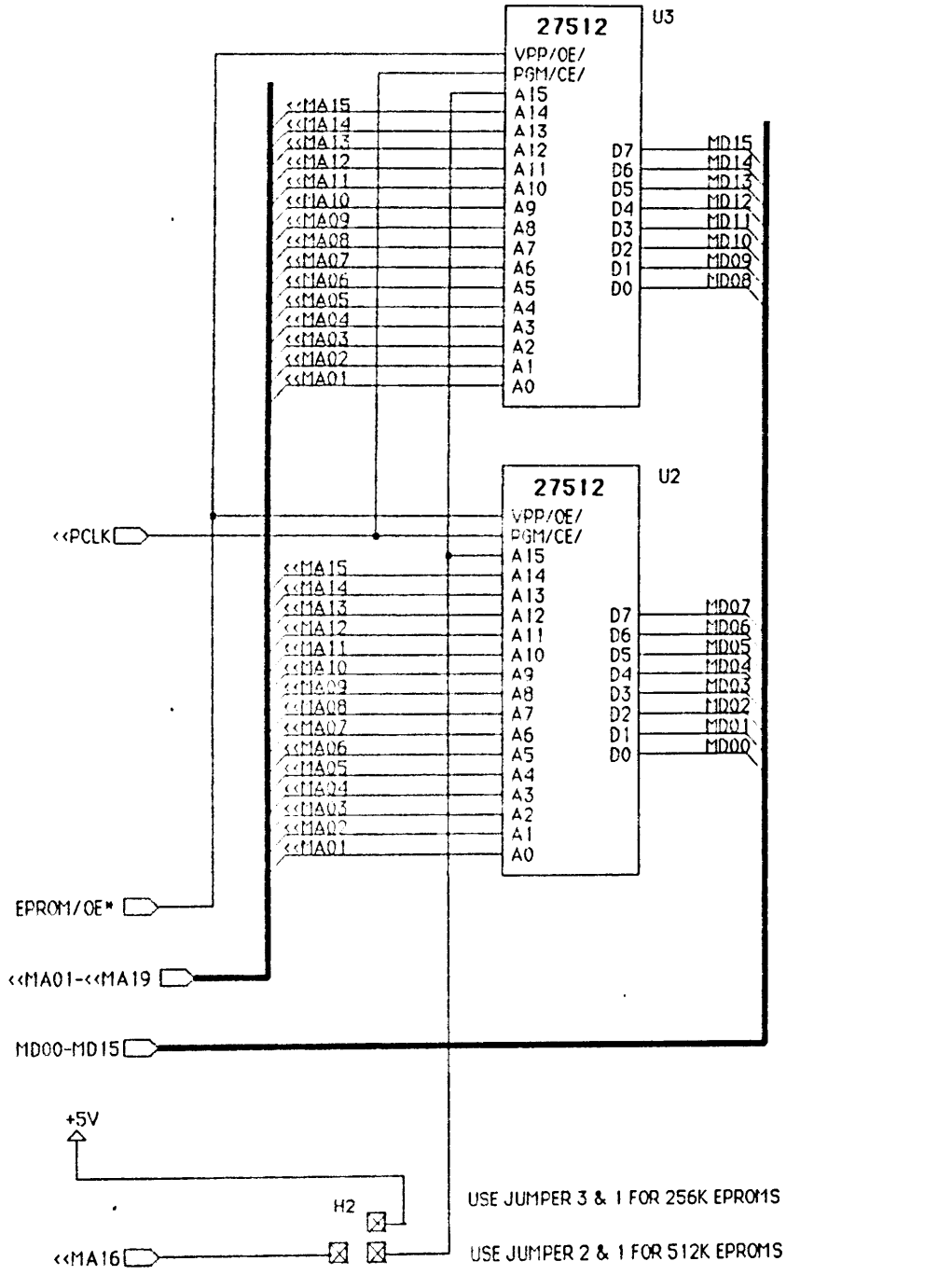
The design of RTECS is provided on the following pages of Schematics. These schematics are referenced repeatedly in chapter 2, System Description, and are organized as follows:

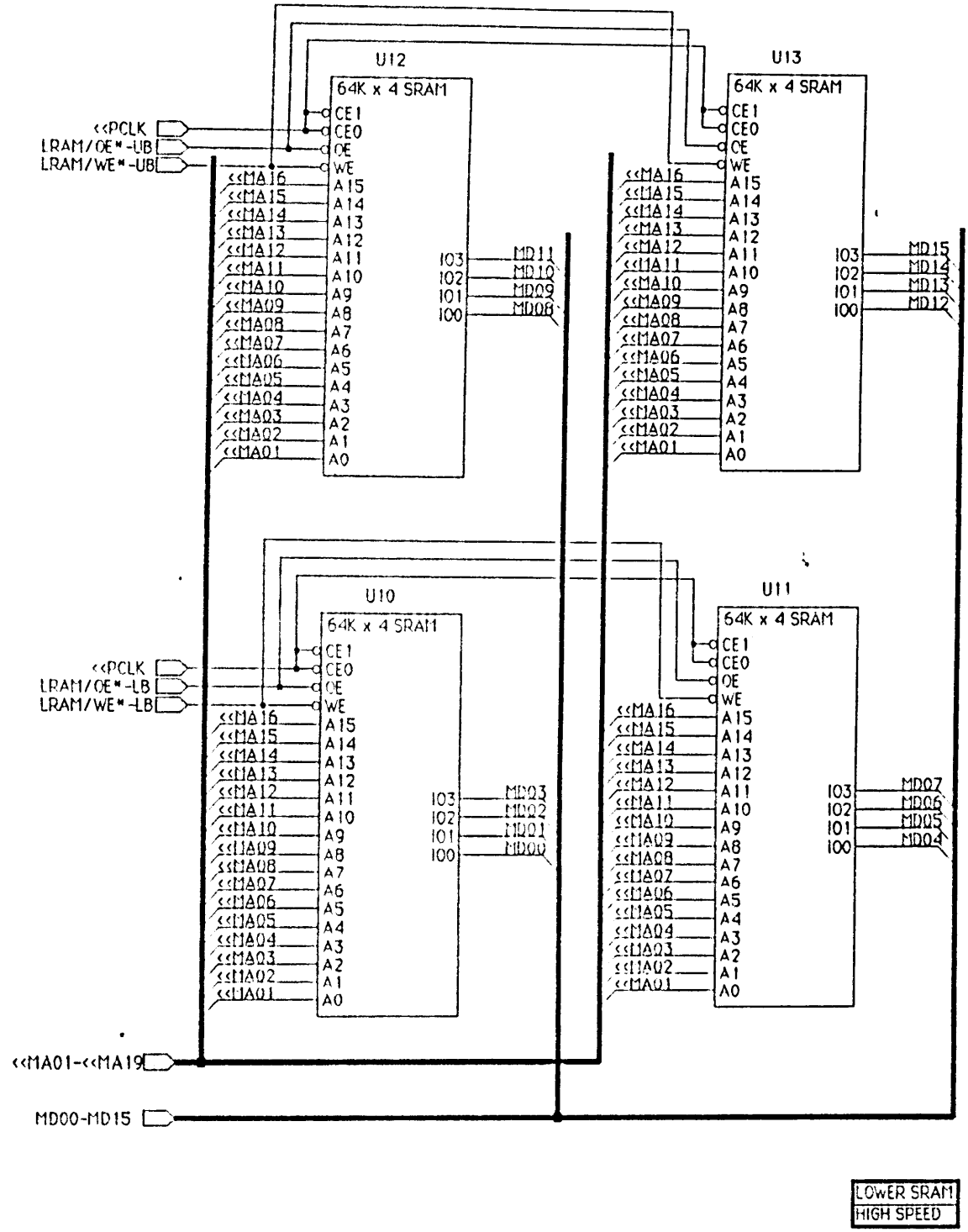
- RTX-2000, Reset circuit, Interrupts, Buffered Memory Data
- Memory Decode, RAM Select, EPROM Select
- EPROM
- Lower RAM (High Speed)
- Upper RAM (High Speed)
- RAM (Nominal Speed)
- G-port, G-Decode, G-Latch, UART, P-port, Real Time Clock, Buffered G-Data
- Programmable Wait State Generator
- VME type Bus connector
- RS-232 serial interface, DB-9, DB-25 connectors
- RF-Regulation
- ADDA Latch, D/A
- A/D, MUX
- References, Phase locked loop, Power Supply connections

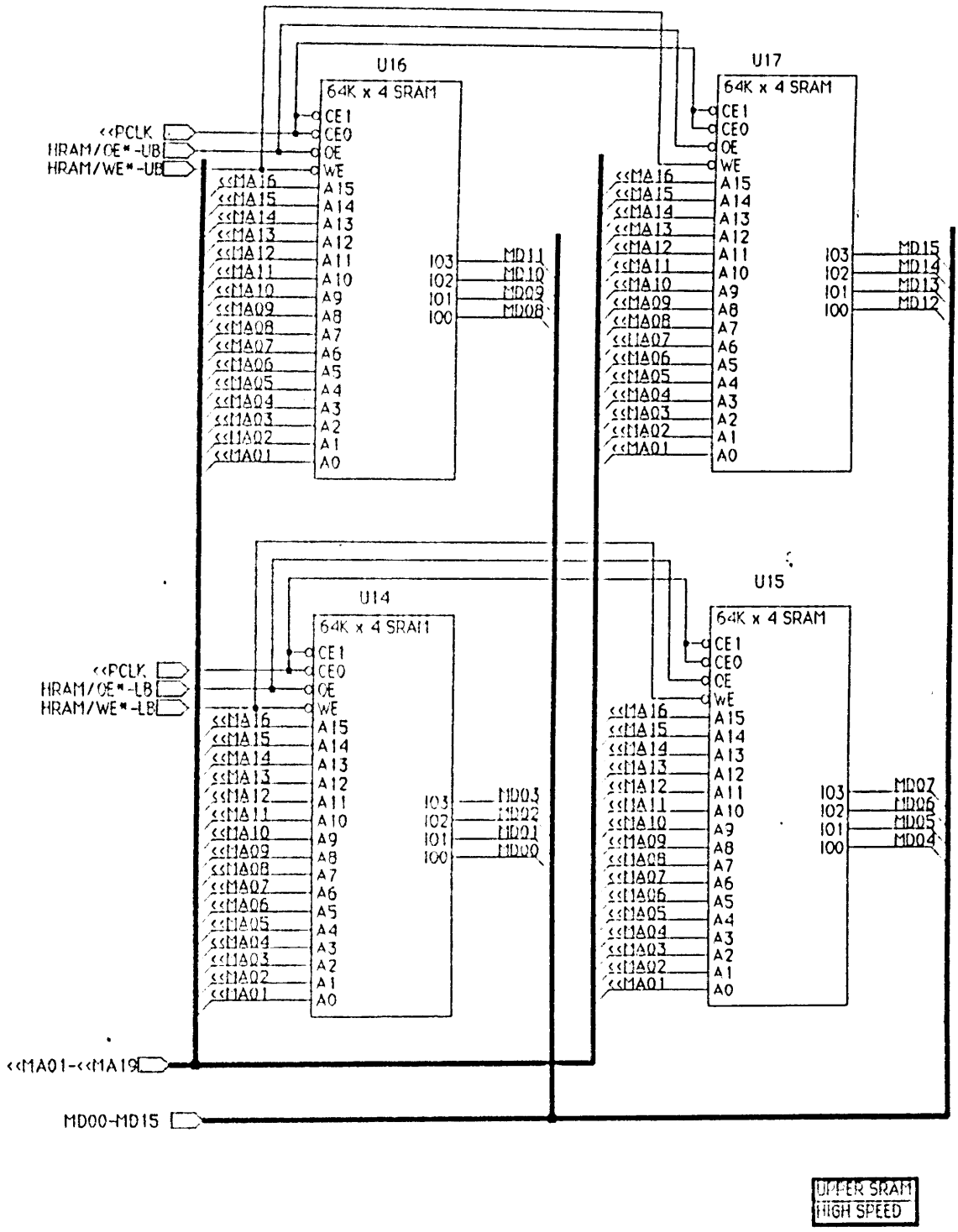

U46
74AC04-C
+5V
<<RESET*
+5V
X1
20MHz
U45
74AC04-C
U45
74AC04-D
RN1
10K RSIP
R1
100K
U46
74AC04-A
<<RESET
<<WAIT
10uF
C1
R3
47
UINTB
UINTA
UINTP
D2
D3
D4
IN4148
>>EI5
>>EI4
>>EI2
>>EI1
SW2
+5V
U34
74AC04-B
74AC32-C
>>INTSUP
R2
100K
>>BNMI
GD15
GD14
GD13
GD12
GD11
GD10
GD09
GD08
GD07
GD06
GD05
GD04
GD03
GD02
GD01
GD00
R4
47
C2
10uF
SW1
GD00-GD15
+5V
+5V
>>USER-WAIT*
RN2
10K RSIP
RN3
10K RSIP
U30-G1
UPPERMEM-SEL
NOTE:
BOARD SIGNALS PROVIDED TO THE USER
ARE SHOWN WITH A PREFIX THAT
INDICATES THE SIGNAL TYPE
PREFIX MEANING EXAMPLE
>> INPUT >>USER-WAIT
<< OUTPUT <<RESET
<> IN/OUT <>GD00
ALSO ALL SIGNALS ARE CONSIDERED
ACTIVE HIGH UNLESS SHOWN WITH
THE POSTFIX "*" AS IN "RESET*".
<>BMD00-<>BMD15
<<M/W*

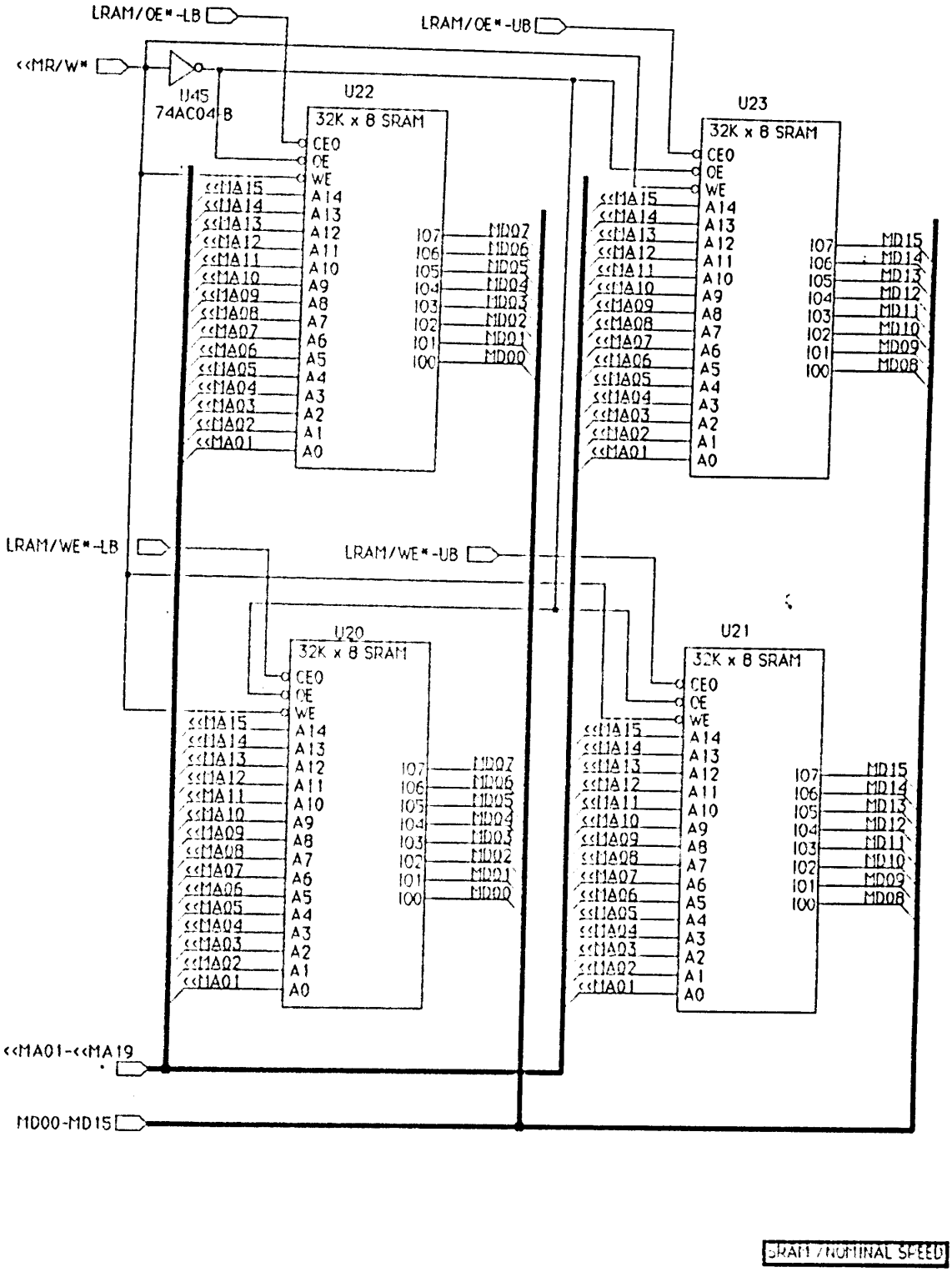

«ICLK
U1
RTX 2000
ICLK
PCLK
«PCLK
WAIT
RESET
TCLK
«TCLK
E15
E14
E13
E12
E11
M11
INTSUP
INDEX
INTA
«INTA
GD15
GD14
GD13
GD12
GD11
GD10
GD09
GD08
GD07
GD06
GD05
GD04
GD03
GD02
GD01
GD00
GR/W*
GIO*
GA2
GA1
GA0
NEW
BOOT
MR/W*
UDS
LDS
«GR/W*
«GIO*
«GA2
«GA1
«GA0
«NEW
«BOOT
«MR/W*
«UDS
«LDS
MA19
MA18
MA17
MA16
MA15
MA14
MA13
MA12
MA11
MA10
MA09
MA08
MA07
MA06
MA05
MA04
MA03
MA02
MA01
«MA19
«MA18
«MA17
«MA16
«MA15
«MA14
«MA13
«MA12
«MA11
«MA10
«MA09
«MA08
«MA07
«MA06
«MA05
«MA04
«MA03
«MA02
«MA01
«MA01-«MA19
MD15
MD14
MD13
MD12
MD11
MD10
MD09
MD08
MD07
MD06
MD05
MD04
MD03
MD02
MD01
MD00
MD00-MD15
U5
74F245
EN
245
A7 B7
A6 B6
A5 B5
A4 B4
A3 B3
A2 B2
A1 B1
A0 B0
DIR
«BMD15
«BMD14
«BMD13
«BMD12
«BMD11
«BMD10
«BMD09
«BMD08
U4
74F245
«BMD07
«BMD06
«BMD05
«BMD04
«BMD03
«BMD02
«BMD01
«BMD00
RTX-2000/RESET/
BUFFERED MDATA/ET INPUT

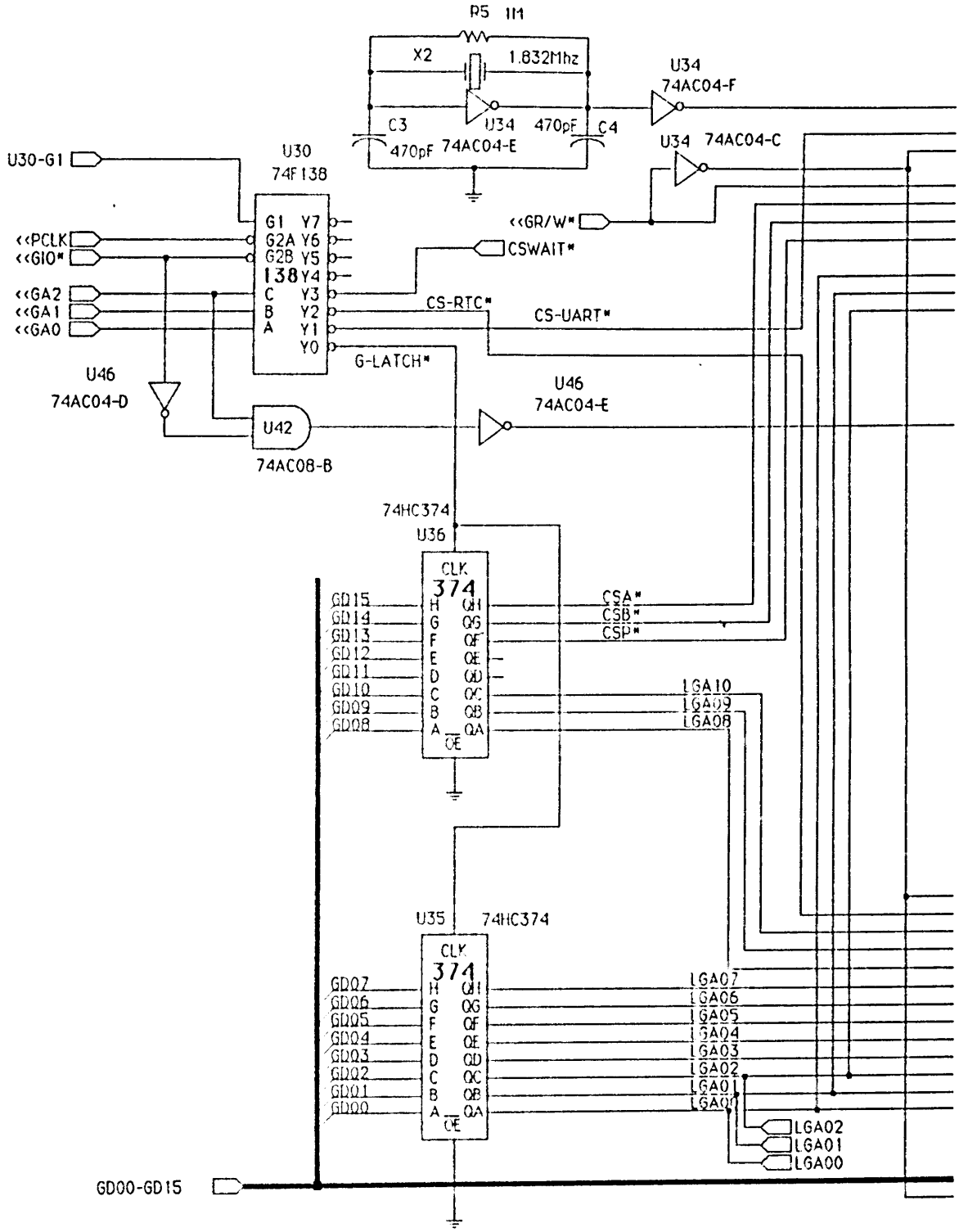

U6
74F138
<<UDS
<<MA19
<<MR/W*
<<BOOT
U42
74AC08-A
<<MA18
<<MA17
G1 Y7
G2A Y6
G2B Y5
138 Y4
C Y3
B Y2
A Y1
Y0
UPPER BYTE SELECT
HRAM/OE*-UB
HRAM/WE*-UB
LRAM/OE*-UB
LRAM/WE*-UB
EPROM/OE*
U45
74AC04-B
U7
74F138
<<LDS
LOWER BYTE SELECT
HRAM/OE*-LB
HRAM/WE*-LB
LRAM/OE*-LB
LRAM/WE*-LB
74AC32-A
U34
74AC04-A
U44
UPPERMEM-SEL
H1
<<MA16
FOR 32Kx8 RAM MODELS, USE JUMPER 1 & 3
<<MR/W*
ALL OTHER MODELS USE JUMPER 1 & 2
MEMORY DECODE/RAM SELECT/
EPROM SELECT

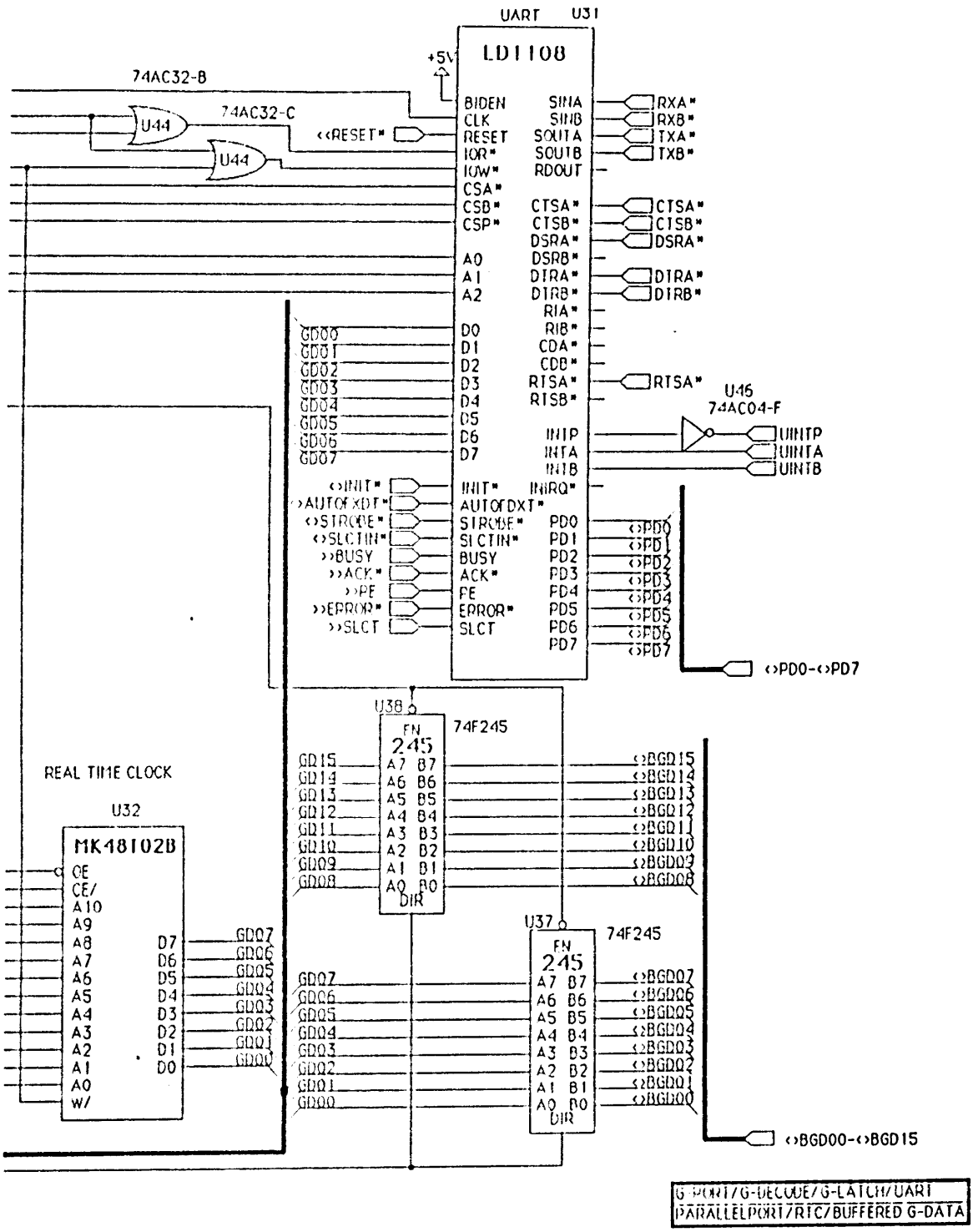

27512
U3
VPP/OE/
PGM/CE/
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
D7
D6
D5
D4
D3
D2
D1
D0
MD15
MD14
MD13
MD12
MD11
MD10
MD09
MD08
<<MA15
<<MA14
<<MA13
<<MA12
<<MA11
<<MA10
<<MA09
<<MA08
<<MA07
<<MA06
<<MA05
<<MA04
<<MA03
<<MA02
<<MA01
27512
U2
<<PCLK
MD07
MD06
MD05
MD04
MD03
MD02
MD01
MD00
EPROM/OE*
<<MA01-<<MA19
MD00-MD15
+5V
H2
<<MA16
USE JUMPER 3 & 1 FOR 256K EPROMS
USE JUMPER 2 & 1 FOR 512K EPROMS
EPROM

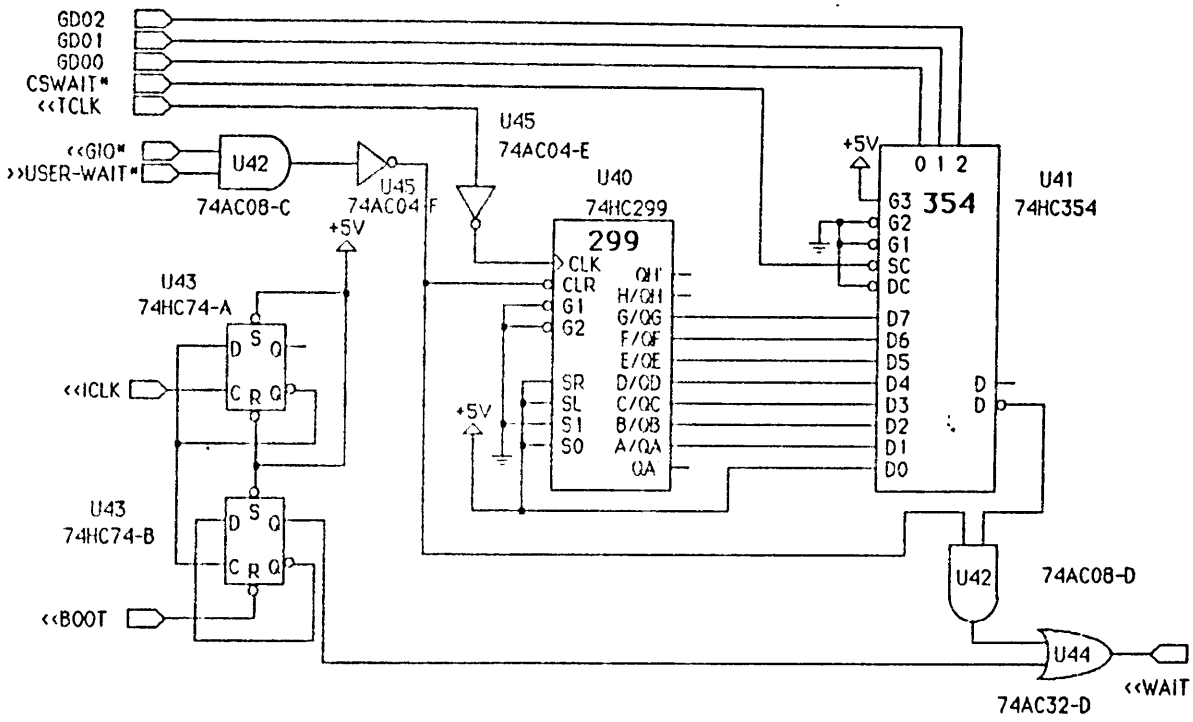

U12
64K x 4 SRAM
U13
64K x 4 SRAM
U10
64K x 4 SRAM
U11
64K x 4 SRAM
<<PCLK
LRAM/OE*-UB
LRAM/WE*-UB
<<PCLK
LRAM/OE*-LB
LRAM/WE*-LB
<<MA01-<<MA19
MD00-MD15
LOWER SRAM
HIGH SPEED


U16
64K x 4 SRAM
U17
64K x 4 SRAM
U14
64K x 4 SRAM
U15
64K x 4 SRAM
<<PCLK
HRAM/OE*-UB
HRAM/WE*-UB
HRAM/OE*-LB
HRAM/WE*-LB
CE1
CE0
OE
WE
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
MD11
MD10
MD09
MD08
MD15
MD14
MD13
MD12
MD03
MD02
MD01
MD00
MD07
MD06
MD05
MD04
<<MA01-<<MA19
MD00-MD15
UPPER SRAM
HIGH SPEED

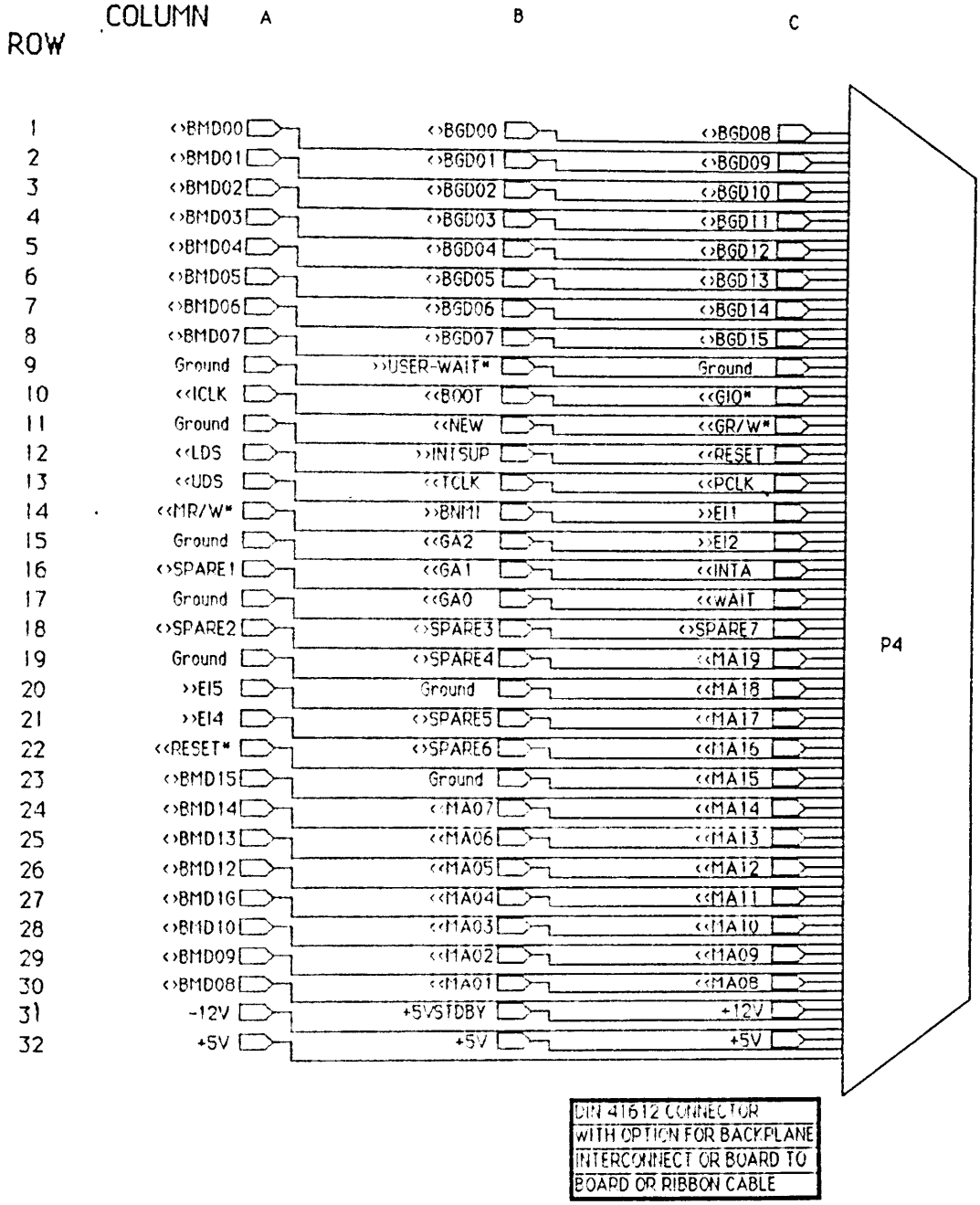

LRAM/OE*-LB
LRAM/OE*-UB
<<MR/W*
U45
74AC04B
U22
32K x 8 SRAM
U23
32K x 8 SRAM
LRAM/WE*-LB
LRAM/WE*-UB
U20
32K x 8 SRAM
U21
32K x 8 SRAM
<<MA01-<<MA19
MD00-MD15
SRAM / NOMINAL SPEED

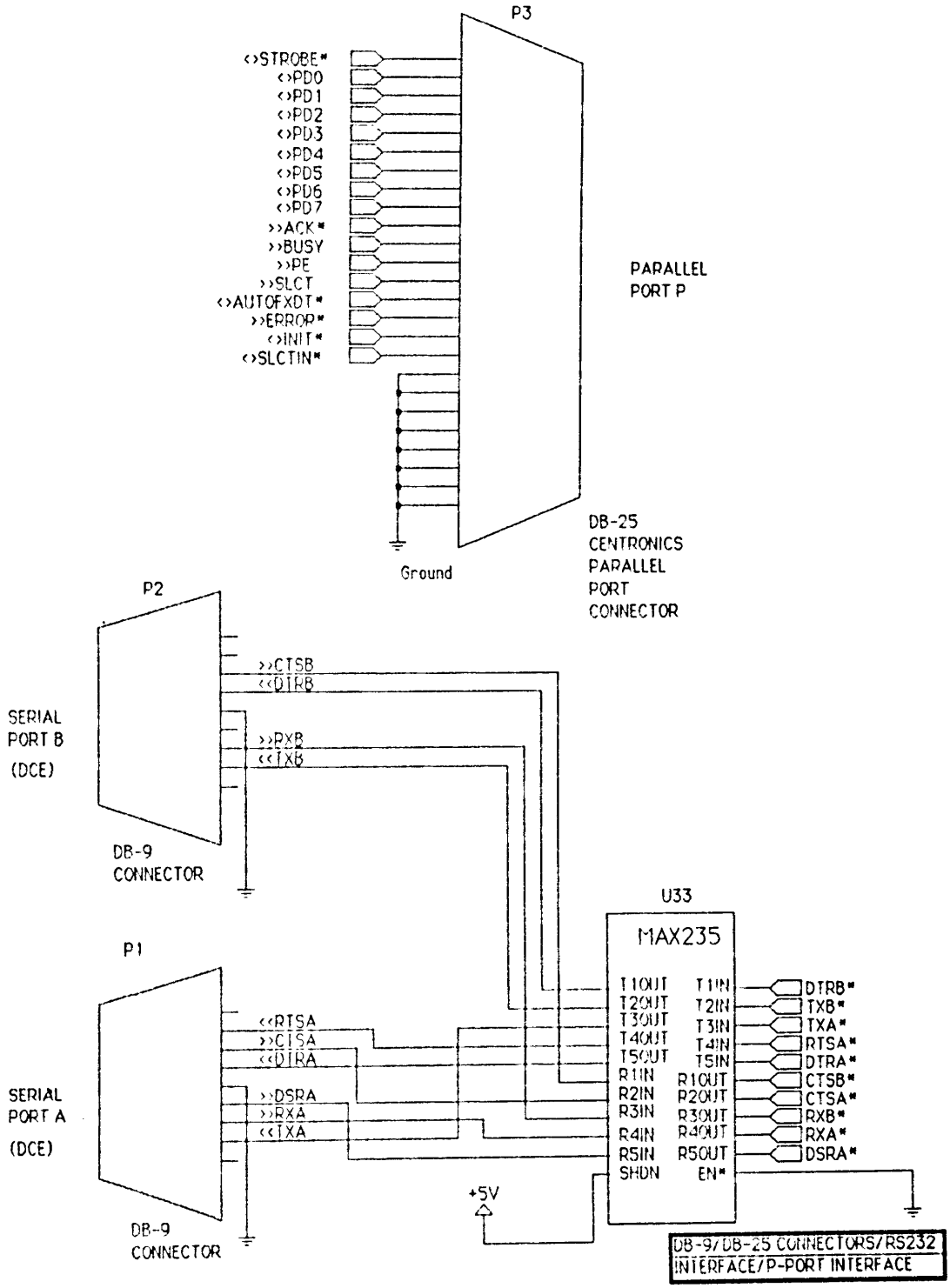

R5 1M
X2
1.832Mhz
C3
470pF
U34
74AC04-E
470pF
C4
U34
74AC04-F
U34 74AC04-C
U30-G1
U30
74F138
G1 Y7
G2A Y6
G2B Y5
138 Y4
C Y3
B Y2
A Y1
Y0
<<PCLK
<<GIO*
<<GA2
<<GA1
<<GA0
<<GR/W*
CSWAIT*
CS-RTC*
CS-UART*
G-LATCH*
U46
74AC04-D
U42
74AC08-B
U46
74AC04-E
74HC374
U36
CLK
374
GD15
GD14
GD13
GD12
GD11
GD10
GD09
GD08
CSA*
CSB*
CSP*
LGA10
LGA09
LGA08
U35
74HC374
GD07
GD06
GD05
GD04
GD03
GD02
GD01
GD00
LGA07
LGA06
LGA05
LGA04
LGA03
LGA02
LGA01
LGA00
LGA02
LGA01
LGA00
GD00-GD15

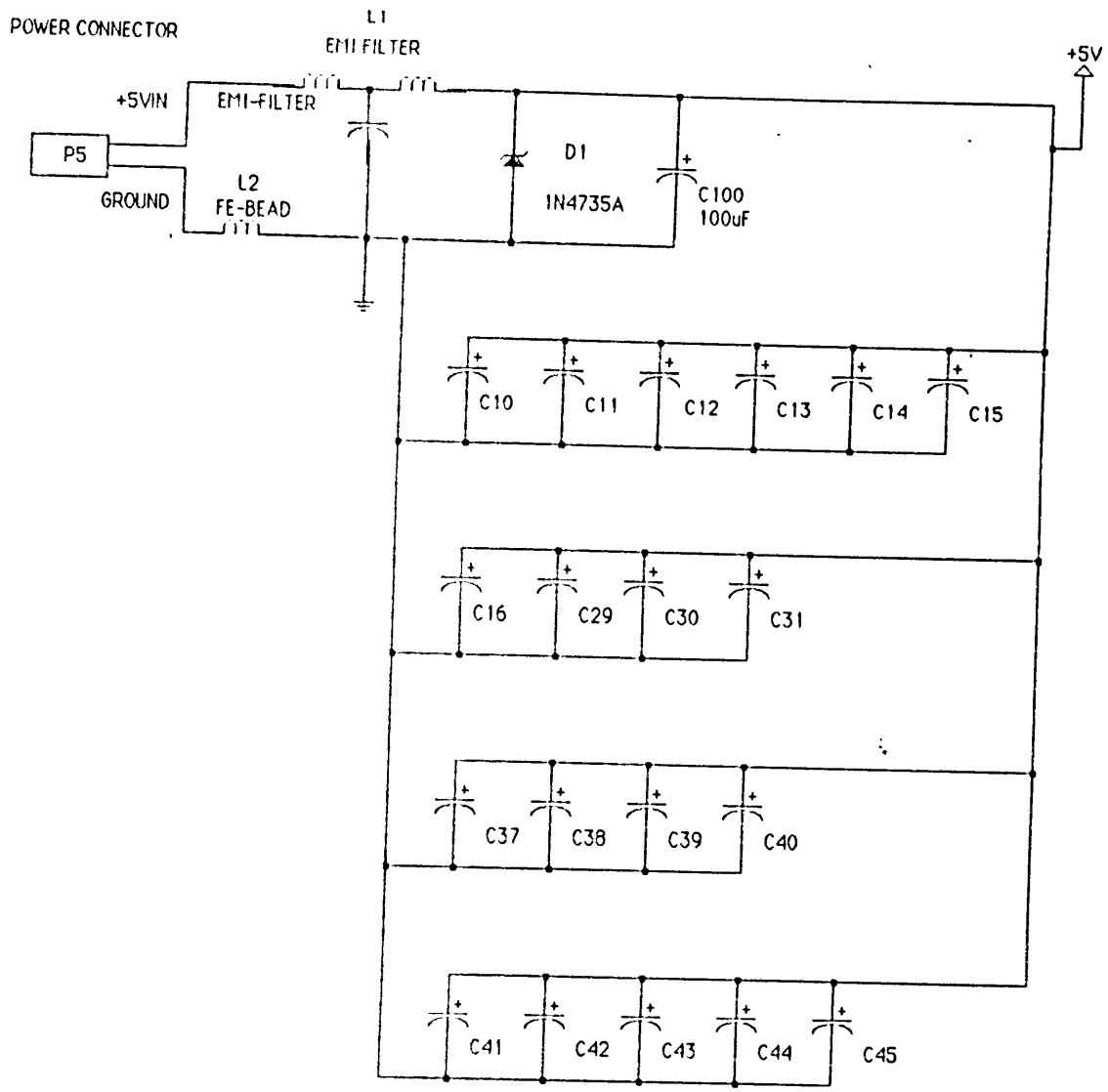

UART U31
LD1108
+5V
74AC32-B
74AC32-C
U44
U44
<<RESET*
BIDEN
CLK
RESET
IOR*
IOW*
CSA*
CSB*
CSP*
A0
A1
A2
D0
D1
D2
D3
D4
D5
D6
D7
INIT*
AUTOFDXT*
STROBE*
SLCTIN*
BUSY
ACK*
PE
ERROR*
SLCT
SINA
SINB
SOUTA
SOUTB
RDOUT
CTSA*
CTSB*
DSRA*
DSRB*
DTRA*
DTRB*
RIA*
RIB*
CDA*
CDB*
RTSA*
RTSB*
INTP
INTA
INTB
INIRQ*
PD0
PD1
PD2
PD3
PD4
PD5
PD6
PD7
RXA*
RXB*
TXA*
TXB*
CTSA*
CTSB*
DSRA*
DTRA*
DTRB*
RTSA*
U46
74AC04-F
UINTP
UINTA
UINTB
<>PD0-<>PD7
U38
74F245
FN 245
DIR
U37
74F245
FN 245
DIR
<>BGD00-<>BGD15
REAL TIME CLOCK
U32
MK48T02B
OE
CE/
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
W/
G PORT/G-DECODE/G-LATCH/UART
PARALLELPORT/RTC/BUFFERED G-DATA

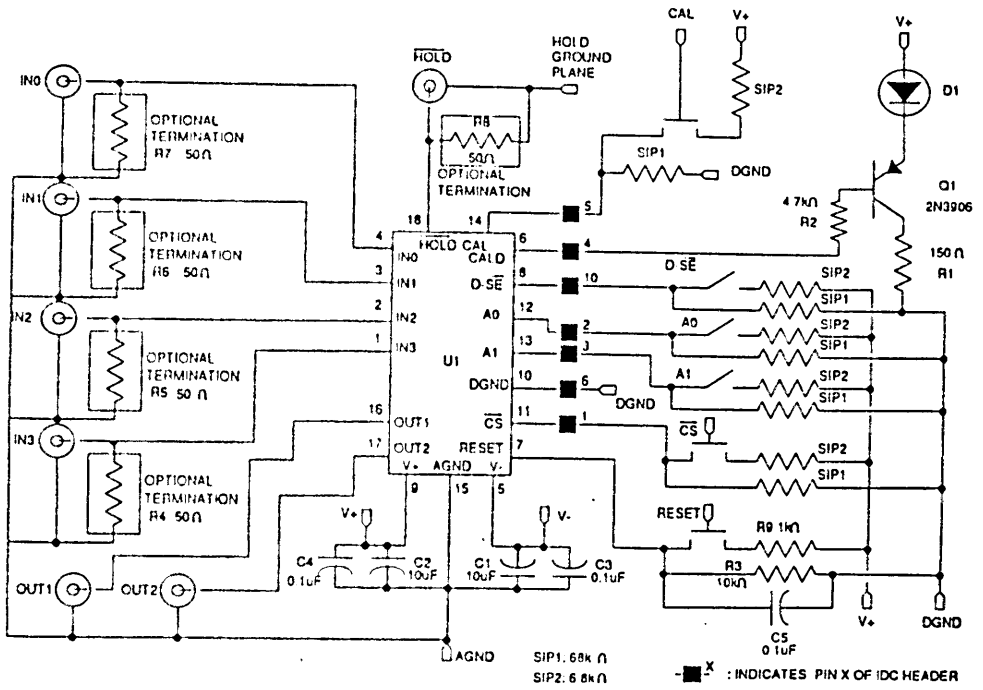


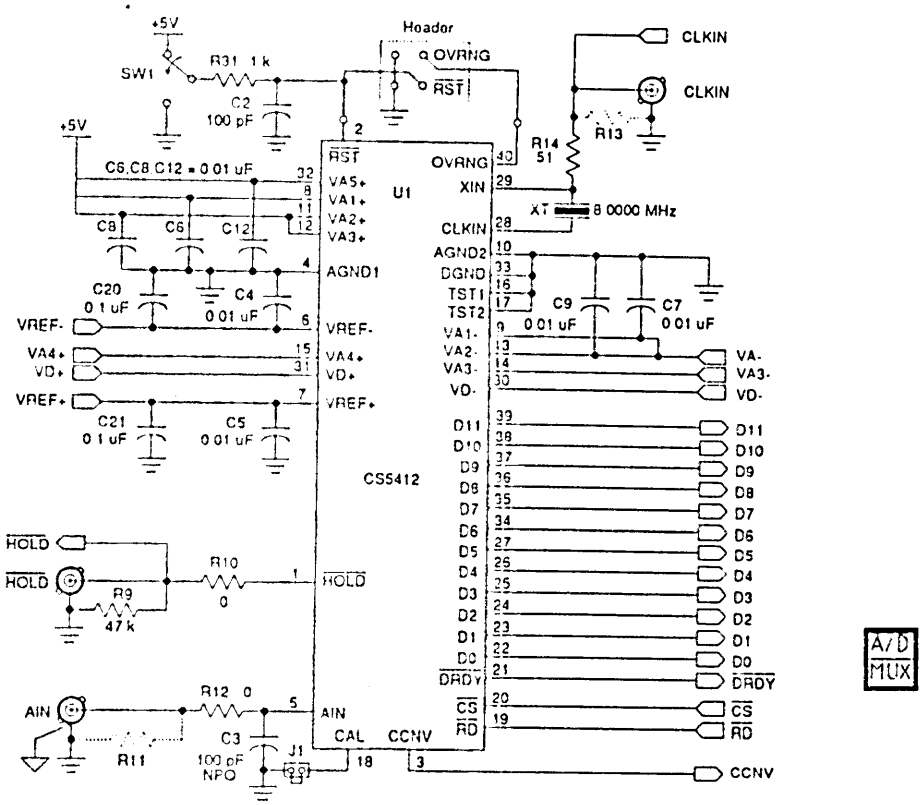

PROGRAMMABLE WAIT STATE GENERATOR

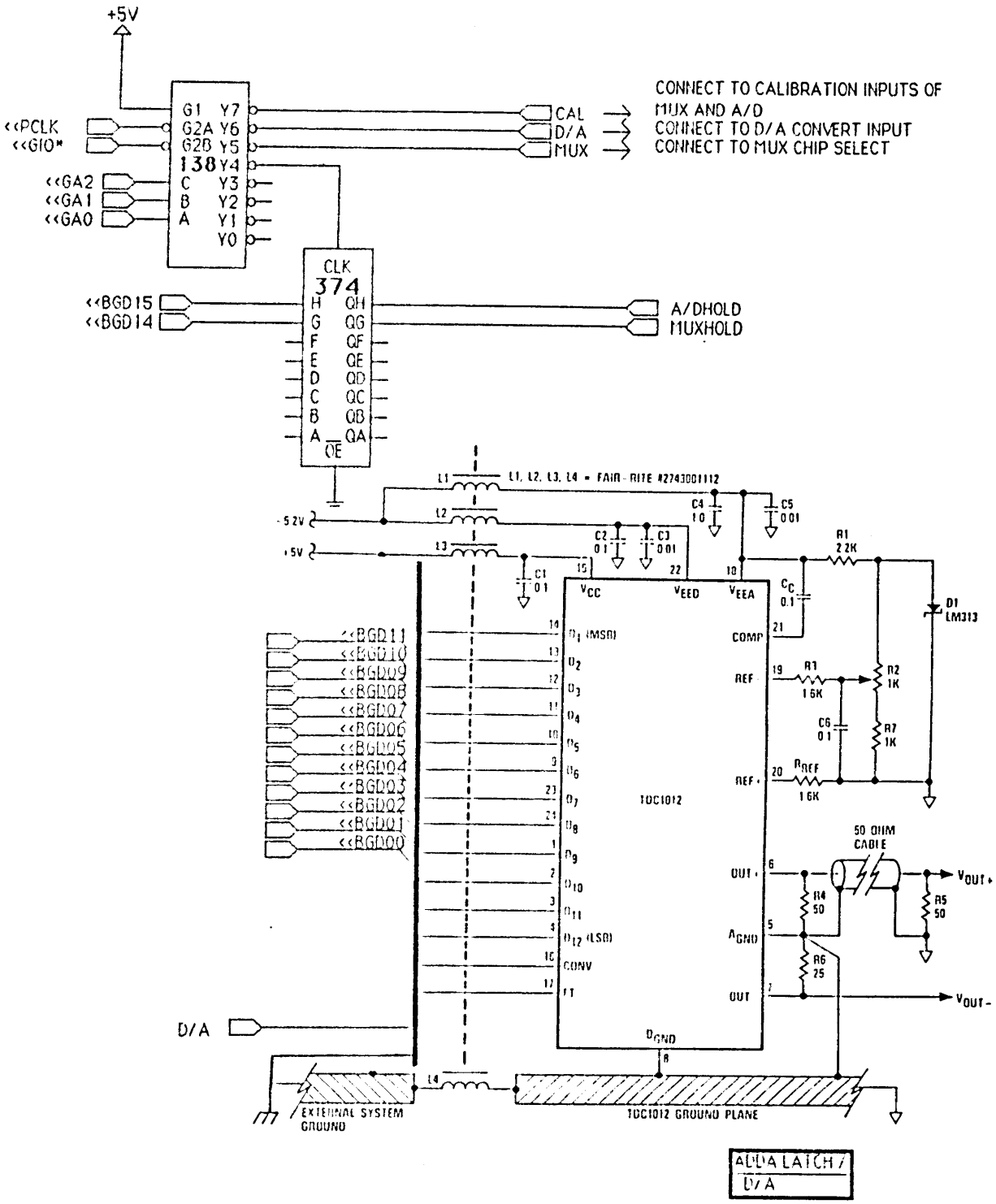

COLUMN
ROW
A
B
C
1 <>BMD00 <>BGD00 <>BGD08
2 <>BMD01 <>BGD01 <>BGD09
3 <>BMD02 <>BGD02 <>BGD10
4 <>BMD03 <>BGD03 <>BGD11
5 <>BMD04 <>BGD04 <>BGD12
6 <>BMD05 <>BGD05 <>BGD13
7 <>BMD06 <>BGD06 <>BGD14
8 <>BMD07 <>BGD07 <>BGD15
9 Ground >>USER-WAIT* Ground
10 <<ICLK <<BOOT <<GIO*
11 Ground <<NEW <<GR/W*
12 <<LDS >>INTSUP <<RESET
13 <<UDS <<TCLK <<PCLK
14 <<MR/W* >>BNMI >>EI1
15 Ground <<GA2 >>EI2
16 <>SPARE1 <<GA1 <<INTA
17 Ground <<GA0 <<WAIT
18 <>SPARE2 <>SPARE3 <>SPARE7
19 Ground <>SPARE4 <<MA19
20 >>EI5 Ground <<MA18
21 >>EI4 <>SPARE5 <<MA17
22 <<RESET* <>SPARE6 <<MA16
23 <>BMD15 Ground <<MA15
24 <>BMD14 <<MA07 <<MA14
25 <>BMD13 <<MA06 <<MA13
26 <>BMD12 <<MA05 <<MA12
27 <>BMD16 <<MA04 <<MA11
28 <>BMD10 <<MA03 <<MA10
29 <>BMD09 <<MA02 <<MA09
30 <>BMD08 <<MA01 <<MA08
31 -12V +5VSTDBY +12V
32 +5V +5V +5V
P4
DIN 41612 CONNECTOR
WITH OPTION FOR BACKPLANE
INTERCONNECT OR BOARD TO
BOARD OR RIBBON CABLE

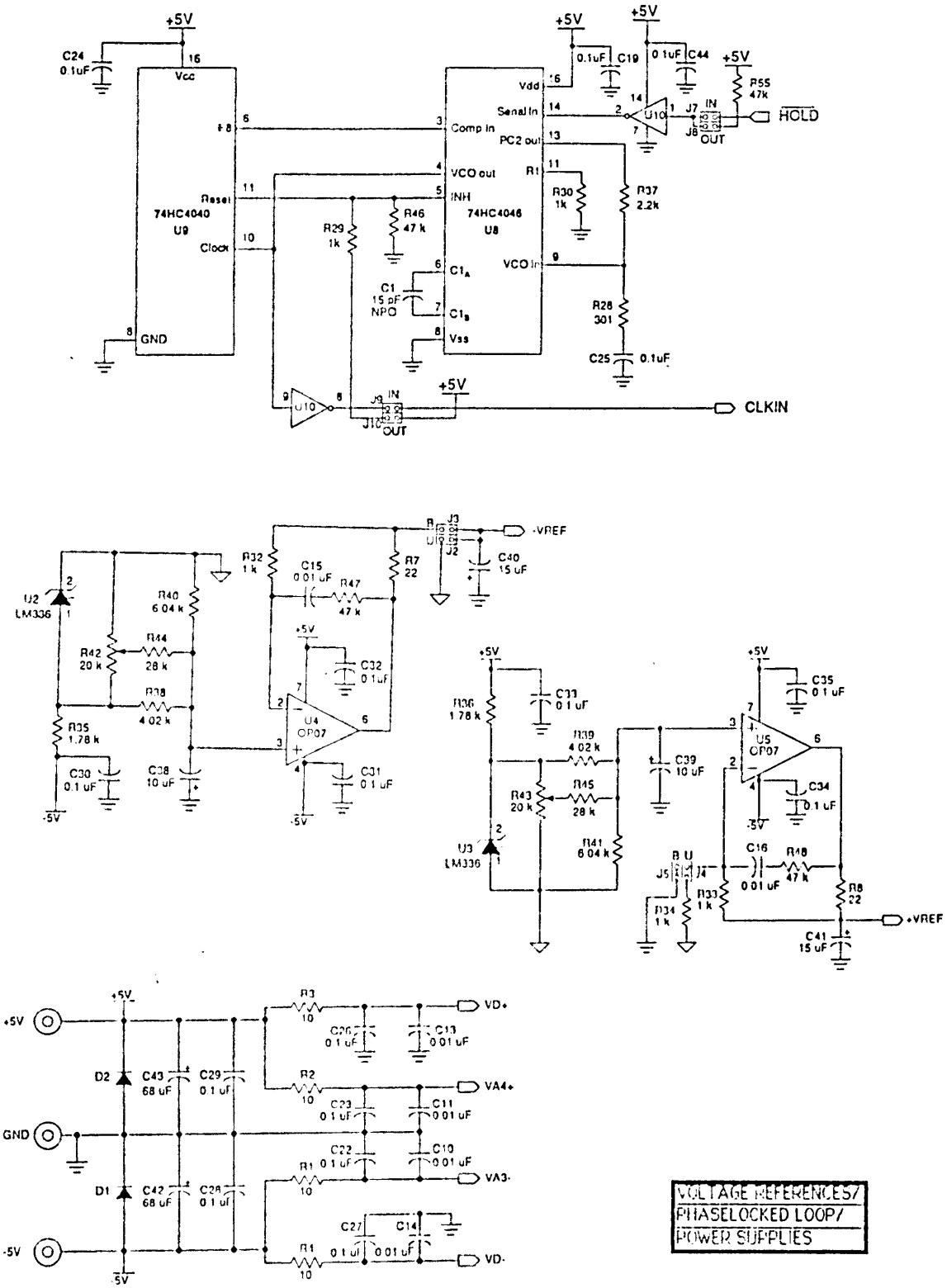

P3
<>STROBE*
<>PD0
<>PD1
<>PD2
<>PD3
<>PD4
<>PD5
<>PD6
<>PD7
>>ACK*
>>BUSY
>>PE
>>SLCT
<>AUTOFXDT*
>>ERROR*
<>INIT*
<>SLCTIN*
PARALLEL
PORT P
Ground
DB-25
CENTRONICS
PARALLEL
PORT
CONNECTOR
P2
SERIAL
PORT B
(DCE)
>>CTSB
<<DTRB
>>RXB
<<TXB
DB-9
CONNECTOR
U33
MAX235
T1OUT
T2OUT
T3OUT
T4OUT
T5OUT
R1IN
R2IN
R3IN
R4IN
R5IN
SHDN
T1IN
T2IN
T3IN
T4IN
T5IN
R1OUT
R2OUT
R3OUT
R4OUT
R5OUT
EN*
DTRB*
TXB*
TXA*
RTSA*
DTRA*
CTSB*
CTSA*
RXB*
RXA*
DSRA*
+5V
P1
SERIAL
PORT A
(DCE)
<<RTSA
>>CTSA
<<DTRA
>>DSRA
>>RXA
<<TXA
DB-9
CONNECTOR
DB-9/DB-25 CONNECTORS/RS232
INTERFACE/P-PORT INTERFACE


POWER CONNECTOR
L1
EMI FILTER
+5V
+5VIN
EMI-FILTER
P5
GROUND
L2
FE-BEAD
D1
1N4735A
C100
100uF
C10
C11
C12
C13
C14
C15
C16
C29
C30
C31
C37
C38
C39
C40
C41
C42
C43
C44
C45


IN0
IN1
IN2
IN3
OUT1
OUT2
OPTIONAL TERMINATION R7 50Ω
OPTIONAL TERMINATION R6 50Ω
OPTIONAL TERMINATION R5 50Ω
OPTIONAL TERMINATION R4 50Ω
HOLD
HOLD GROUND PLANE
R8 50Ω OPTIONAL TERMINATION
CAL
V+
SIP2
SIP1
DGND
D1
Q1 2N3906
4.7kΩ R2
150Ω R1
D-SE
A0
A1
CS
RESET
R9 1kΩ
R3 10kΩ
C5 0.1uF
U1
HOLD CAL
CALD
D-SE
A0
A1
DGND
CS
RESET
V+ AGND V-
C4 0.1uF
C2 10uF
C1 10uF
C3 0.1uF
V+
V-
AGND
SIP1: 68k Ω
SIP2: 6.8kΩ
X : INDICATES PIN X OF IDC HEADER


+5V
SW1
R31 1k
C2 100 pF
Header
OVRNG
RST
CLKIN
CLKIN
R13
R14 51
XT 8 0000 MHz
C6,C8,C12 = 0.01 uF
C8
C6
C12
C20 0.1 uF
C4 0.01 uF
C9 0.01 uF
C7 0.01 uF
C21 0.1 uF
C5 0.01 uF
U1
CS5412
RST
VA5+
VA1+
VA2+
VA3+
AGND1
VREF-
VA4+
VD+
VREF+
OVRNG
XIN
CLKIN
AGND2
DGND
TST1
TST2
VA1-
VA2-
VA3-
VD-
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
DRDY
CS
RD
CAL
CCNV
VA-
VA3-
VD-
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
DRDY
CS
RD
CCNV
HOLD
HOLD
R9 47k
R10 0
AIN
R11
R12 0
C3 100 pF NPO
J1
A/D MUX


+5V
G1 Y7
G2A Y6
G2B Y5
138 Y4
C Y3
B Y2
A Y1
Y0
<<PCLK
<<GIO*
<<GA2
<<GA1
<<GA0
CAL
D/A
MUX
CONNECT TO CALIBRATION INPUTS OF MUX AND A/D
CONNECT TO D/A CONVERT INPUT
CONNECT TO MUX CHIP SELECT
CLK
374
H QH
G QG
F QF
E QE
D QD
C QC
B QB
A QA
OE
<<BGD15
<<BGD14
A/DHOLD
MUXHOLD
L1, L2, L3, L4 = FAIR-RITE #2743001112
-5.2V
+5V
C1 0.1
C2 0.1
C3 0.01
C4 10
C5 0.01
R1 2.2K
D1 LM313
CC 0.1
R1 1.5K
R2 1K
C6 0.1
R7 1K
RREF 1.6K
<<BGD11
<<BGD10
<<BGD09
<<BGD08
<<BGD07
<<BGD06
<<BGD05
<<BGD04
<<BGD03
<<BGD02
<<BGD01
<<BGD00
TDC1012
VCC
VEED
VEEA
COMP
REF-
REF+
OUT+
AGND
OUT-
DGND
CONV
RT
D1 (MSB)
D12 (LSB)
50 OHM CABLE
R4 50
R5 50
R6 25
VOUT+
VOUT-
D/A
EXTERNAL SYSTEM GROUND
TDC1012 GROUND PLANE
ADDA LATCH / D/A




VOLTAGE REFERENCES/
PHASELOCKED LOOP/
POWER SUPPLIES

# Appendix B. Low Pass Filter Example

The listing on the next pages is a Forth Implementation of a low pass filter discussed in section 3.4. It is provided here for two purposes; first it provides justification for the model development procedure of 3.4 and second it provides a simple example of Forth programming.

```
*************************************************************************
*       This program performs the calculations necessary to implement a *
*Low Pass Filter in Forth.  The Filter is designed with fc= 100 Hz and the *
*sampling rate T=.001 sec.  It is executed on the Macintosh under MACH II. *
*The first section is a number format section to provide easy input of decimal *
*and scaled numbers.  The following section lists the LPF comments for provide *
*an association to a BASIC program                                      *
*************************************************************************
*
\  ***************NUMBER  FORMAT  ***********************************
Only Forth Also Mac
Decimal
: EndIF [Compile] Then ;                \ makes for easier reading.
        Immediate

\ Notation:
\ n = unscaled integer.
\ s = scaled integer.

Variable Buffer 64 Vallot

   32 Constant BL                       \ ascii value of a space
    5 Constant Factor                   \ scaling factor
   50 Constant Fudge                    \ round up constant

: 10^ ( n -- 10^n )
        1 swap 0 Do 10 * Loop ;

: +1 ( -- "SCALED-ONE" )        \ Gives 4 places before AND after decimal pt
        [ Factor 10^ ] Literal ;    \ 10^Factor

: *. ( s s -- s )                       \ multiply two scaled integers
        +1 */ ;

: /. ( s s -- s )                       \ divide two scaled integers
        +1 SWAP */ ;

: #>$ ( s -- addr cnt )                 \ convert scaled integer to string
        dup abs <# Factor 0 Do # Loop Ascii . hold #S sign #> ;

: .# ( s -- )                                   \ print out the scaled integer on the stack.
        #>$ type ;

: Scale ( n factor -- s )    \ scale n by 'factor' decimal places
        dup 0< IF abs 0 Do 10 / Loop exit EndIF
        dup   IF    0 Do 10 * Loop Else drop EndIF ;

: N>S ( n -- s )
        Factor DPL @ - Scale ;
```

```
: ns1  ( n -- s ) N>S ;                \ for the lazy typist
\ : ns2 dpl ! ns1 ;
\ : ns state @ if dpl @ [compile] literal ns2 else ns1 then ;
\ immediate

: ns state @ if dpl @ [compile] literal compile dpl compile ! compile ns1 else ns1 then ;
 immediate

: S>N ( s -- n )
        Fudge + +1 / ;
: sn  ( s -- n ) S>N ;                 \ for the lazy typist

: $># ( addr -- s )                            \ convert string to scaled integer
        number? IF N>S EndIF ;

: Get# ( -- s )                        \ input a scaled number from the keyboard.
        Buffer 1+ 64 Expect            \ get input string
        Span @ Buffer C!                       \ save its length
        BL Buffer dup C@ + 1+ C!       \ terminate string with a blank space
        Buffer $># ;                           \ convert it
Create SinTable                                        \ create a table for sin function
        180 4* allot                           \ allocate space in memory for table

: >Sin ( index -- adr )
        4* SinTable + ;

\ Use FP to calc table

        Also Sane FP

        : CalcTable ( -- )
                FP
                        180 0 Do
                                I I>F Deg>Rad Fsin 1e5 F* F>I I >Sin !
                        Loop
                INT ;

        INT
Only Forth Also Mac

CalcTable

: Sin ( n -- Sin[n] )          \ 'n' is in degrees
        dup 0< swap
        abs 180 /mod swap >R
        0= 0= xor R> >Sin @
        swap IF negate EndIF ;

: Cos ( n -- Cos[n] )
        90 + Sin ;
```

```
3.14159265 ns Constant PI

: Rad>Deg ( s -- n2 )
        [ 180 ns ] Literal *. PI /. ;

: Deg>Rad ( s -- n2 )
        PI *. [ 180 ns ] Literal /. ;

: rSin ( s -- Sin[n] )        \ 's' is in scaled radians
        Rad>Deg S>N Sin ;

: rCos ( s -- Sin[n] )        \ 's' is in scaled radians
        Rad>Deg S>N Cos ;

: DumpSine ( -- )
        CR ." rads   Sine" CR
        361 0 Do
                I n>s deg>rad .# 2 spaces I sin .# CR
        Loop ;
\**********************LPF   EXAMPLE*************************************

\       INITITIALIZATION
VARIABLE TIME                                   \ DEFINE VARIABLES
VARIABLE OUT0
VARIABLE OUT1
VARIABLE IN0
0 TIME !                                        \  INITIALIZE  VARIBLES
AND CONSTANTS
0 OUT1 !
0.00100 ns CONSTANT T                           \ T=.001
: PRINT CR ."   TIME    INPUT    OUTPUT " ;        \ Heading


\       MAIN LOOP
: MLOOP CR PRINT
 200 1 DO                                       \ FOR K=1 TO 200
        TIME @ T + TIME !                       \ TIME= TIME + T
        TIME @ 6.2830 ns *. 25  * sn rSIN       \  SIN(6.283*25*TIME)
        IN0 !                                   \  IN0= SIN(6.283*25*TIME)
        OUT0 @ OUT1 !                           \ OUT1=OUT0
        .4665 ns IN0 @ *. .5335 ns OUT1 @ *. +
        OUT0 !                                  \ OUT0=(.4665*INO)+
                                                \       (.5335*OUT1)
        TIME @ .#                               \ Type time
        ."    "
        IN0 @ .#                                \ Type INO
        ."    "
        OUT0 @ .#                               \ Type OUT
        CR
LOOP ;
```

# Appendix C. Basic Compensator (BC)

The following listing is provided for comparison. It shows the Basic program used by Cox et. al. in the Vibration Control project. Its comments and code coincide with the comments and code of the next section Appendix D. Forth Compensator (FC).

```
*************************************************************
*                      By  Dave Cox                         *
*                          3-14-89                          *
*  This program is used as digital controller for the       *
*  slewing experiment.                                      *
*                                                           *
*                                                           *
*************************************************************
$include "c:\rhw\ad\adda.bas"
cls
dim d(3,3005)
Kp=-200/2.4                    ' position gain
'Kv=-.04/2.4                   ' rate gain
Kf=4500                        ' fiber gain


pz%=2020                       'position zero

sgz%=2047                      'strain gauge zero

*************************************************************
                        ADC CONSTANTS
*************************************************************

p.gain%=2
p.gain=1.0/(204.7*2^p.gain%)              'volts
p.calibration=3.2                         ' convert volts to
radians

r.gain%=3
r.gain=1.0/(204.7*2^r.gain%)              'volts
r.calibration=-72*2*3.14159/60            ' convert volts to
radians/sec

fb.gain%=3
fb.gain=1.0
fb.calibration=.00025/(1000)

'sg.gain%=2
'sg.gain=1.0/(204.7*2^sg.gain%)
'sg.calibration=-.00069

'ref.gain=0
'ref.gain=1.0/(204.7*2^ref.gain%)

*************************************************************
  INITIALIZE A/D BOARDS  SET QUADRATURE POINT
*************************************************************

Call init(&H2EC)                          ' Initialize A/D
board
Call daout(&H2EC,0,2047)                  ' place all control
voltages at zero
Call daout(&H2EC,1,1229)                  ' set up to power amp
```

```
********************* Set Fiber and Tach zeros ***********

fave=0
rave=0
for l=1 to 800
 Call adin(&H2EC,fb.gain%,4,fbz%)
 Call adin(&H2EC,r.gain%,3,rz%)
 fave=fave+fbz%
 'rave=rave+rz%
next l
fbz%=fave/800
rz%=rave/800-20
print "Rate Zero ";rz%
print "Fiber Zero "fbz%
print "Ready";
input ans
print
print "Press any key to stop"
time=timer
i=0

*****************************************************************
                          MAIN LOOP
*****************************************************************

while not instat and i<3000          ' loop until a key is
pressed
 i=i+1

*********************** Input Signals **********************
  Call adin(&H2EC,p.gain%,2,p%)
  Call adin(&H2EC,r.gain%,3,r%)
  Call adin(&H2EC,fb.gain%,4,fb%)

    ' Call adin(&H2EC,sg.gain%,5,sg1%)
    ' Call adin(&H2EC,ref.gain%,6,ref%)   ' Reference input

****** This section excites the beam with a sinusoid *******

'  if i<870 then
'   pz%=100*cos(152*i/435)+2047
'  else
'   pz%=2047
 ' end if


**********************SoftwareStop*****************************

  if p%>2350 or p%<1700 or pz%>2350 or pz% <1700 then
    call daout(&H2ec,0,2047)
    beep
    cls
    print "Error in pot position or commanded position"
```

```
    print "  Commanded position ";pz%
    print "    Present position ";p%
    end
  end if

************** Calibrate and Filter**************************

  p=(p%-pz%)*p.gain*p.calibration              ' POSITION
(radians)
  r=(r%-rz%)*r.gain*r.calibration              ' RATE
(radians/sec)
  fb=(fb%-fbz%)*fb.gain*fb.calibration         ' FIBER
UNFILTERED

 ' fbf=(fbf%-fbz%)*fb.gain*fb.calibration      ' FIBER
FILTERED
 ' sg1=(sg1%-sgz%)*sg.gain*sg.calibration      ' STRAIN
GAUGE 1

  fbf=fb-fbold(1)+.998*fbfold(1)               ' DIGITAL
FILTER

pf=(p+pold(1)+pold(2)+pold(3)+pold(4)+pold(5)+pold(6)+pold(7
))/7.0
********************* Control Law **************************

  l1=pf*Kp

  uf=Kf*(fbf-3.74265*fbfold(1)+5.39345*fbfold(2)-
3.55606*fbfold(3))
  uf=uf+Kf*(.905751*fbfold(4))
  uf=uf+3.127084*ufold(1)-
3.620631*ufold(2)+1.841627*ufold(3)
  uf=uf-.348572*ufold(4)


  l1=l1+uf

  If l1> 10 then l1= 10
  If l1<-10 then l1=-10
******************* Input Shaping **************************

    l1f=l1

 '  l1f=.02068419*l1old(1)+.01825255*l1old(2)
 '  l1f=l1f+1.648353*l1fold(1)-.6872893*l1fold(2)

***************** Plotting arrays **************************
  jj=i/5
  if jj=int(i/5) then d(1,jj)=p:d(2,jj)=fbf:d(3,jj)=l1f

  l1f%=l1f*204.7+2047
  Call daout(&H2EC,0,l1f%)
```

```
*************************************************************
    SET UP ARRAYS FOR DIFFERENCE EQUATION

*************************************************************
  l1old(2)=l1old(1)
  l1old(1)=l1
  l1fold(2)=l1fold(1)
  l1fold(1)=l1f
  ufold(4)=ufold(3)
  ufold(3)=ufold(2)
  ufold(2)=ufold(1)
  ufold(1)=uf
  fbold(1)=fb
  fbfold(4)=fbfold(3)
  fbfold(3)=fbfold(2)
  fbfold(2)=fbfold(1)
  fbfold(1)=fbf
  pold(7)=pold(6)
  pold(6)=pold(5)
  pold(5)=pold(4)
  pold(4)=pold(3)
  pold(3)=pold(2)
  pold(1)=p
  Wend


*************************************************************
    SAMPLING FREQ    INSTANT GRAPH    DATA STORAGE
*************************************************************

timef=timer
max=i/5
Call daout(&H2EC,0,2047)      'place all control voltages at
zero


screen 9

window(0,-1)-(max,1)
line(0,0)-(max,0)
for j= 1 to max
 line(j,d(1,j))-(j+1,d(1,j+1))
next j
 input "POSITION,  CONTINUE "; ans
 cls

window(0,-3.14)-(max,3.14)
line(0,0)-(max,0)
for j = 1 to max
 line(j,d(2,j))-(j+1,d(2,j+1))
next j
input "VELOCITY,   CONTINUE ";ans
cls
```

```
 window(0,-.00015)-(max,.00015)
 line(0,0)-(max,0)
 for j = 1 to max
   line(j,d(2,j))-(j+1,d(2,j+1))
 next j
 input "FIBER  CONTINUE ";ans
 cls

window(0,-7)-(max,7)
line(0,0)-(max,0)
line(0,-5/2.4)-(max,-5/2.4)
line(0,5/2.4)-(max,5/2.4)
for j = 1 to max
  line(j,d(3,j))-(j+1,d(3,j+1))
next j
input "MOTOR VOLTAGE,  CONTINUE ";ans

 print using "####.## Sampline Freq  Hertz";i/(timef-time);
 print
 input "Save data ";ans$

 if ans$="y" then
  input "filename ";filnam$
  open filnam$ for output as #1
  for j= 1 to max
    print #1, (timef-time)/max*j,d(1,j),d(2,j)
    next j
  close #1
  print "Done"
 else
  print "Your loss"
 end if
 end
```

# Appendix D. Forth Compensator (FC)

The listing on the next pages is the equivalent of the Basic program of Appendix C. It is executable in the form provided running under Mach II on an Apple Macintosh Computer. It is also the source code for the RTECS program (only minor modifications) which compiled and ran yielding the times listed in the table of figure 4-2.

```
\   ********************************************************************
\                   SLEWING BEAM CONTROLLER                            *
\   ********************************************************************

empty
include" no.format"              \ REQUIRES LOADING OF NO.FORMAT FOR
                                 \        PROPER OPERATION
Only Forth
Decimal

: EndIF [Compile] Then ;     \ makes for easier reading.
   Immediate

\ Notation:
\ n = unscaled integer.
\ s = scaled integer.

Variable Buffer 64 Vallot

   32 Constant BL        \ ascii value of a space
    3 Constant Factor    \ scaling factor
   5 Constant Fudge      \ round up constant

: 10^ ( n -- 10^n )
   1 swap 0 DO 10 * LOOP ;

: +1 ( -- "SCALED-ONE" )     \ Gives 4 places before AND after decimal pt
   [ Factor 10^ ] Literal ; \ 10^Factor

: *. ( s s -- s )            \ multiply two scaled integers
   +1 */ ;

: /. ( s s -- s )            \ divide two scaled integers
   +1 SWAP */ ;

: #>$ ( s -- addr cnt )      \ convert scaled integer to string
   dup abs <# Factor 0 DO # LOOP Ascii . hold #S sign #> ;

: .# ( s -- )                \ print out the scaled integer on the stack.
   #>$ type ;

: Scale ( n factor -- s )    \ scale n by 'factor' decimal places
   dup 0< IF abs 0 DO 10 / LOOP exit EndIF
   dup   IF   0 DO 10 * LOOP Else drop EndIF ;

: N>S ( n -- s )
   Factor DPL @ - Scale ;
: ns1  ( n -- s ) N>S ;      \ for the lazy typist
\ : ns2 dpl ! ns1 ;
\ : ns state @ if dpl @ [compile] literal ns2 else ns1 then ;
```

```
\ immediate

:ns
 state @ if dpl @ [compile] literal compile dpl compile ! compile ns1
 else ns1 then ;
 immediate

: S>N ( s -- n )
   Fudge + +1 / ;
: sn  ( s -- n ) S>N ;       \ for the lazy typist

: $># ( addr -- s )          \ convert string to scaled integer
   number? IF N>S EndIF ;

: Get# ( -- s )              \ input a scaled number from the keyboard.
   Buffer 1+ 64 Expect       \ get input string
   Span @ Buffer C!          \ save its length
   BL Buffer dup C@ + 1+ C! \ terminate string with a blank space
   Buffer $># ;              \ convert it

3.142 ns1 Constant PI

: Rad>Deg ( s -- n2 )
   [ 180 ns1 ] Literal *. PI /. ;

: ** ( s u -- s ) swap dup rot    \ The power function s**u
   1 do swap dup rot *. loop swap drop ;


\   **********************************CONSTANTS*****************************

\ CONSTANTS- SCALED (S)  UNSCALED (N)- INDICATED WITH " % " ON VARIABLES
2047 CONSTANT ZERO%              \ D/A "0" VOLTS (N)
-200 NS 2.4 ns /. CONSTANT Kp    \ POSITION GAIN -200/2.4 (S)
-.04 NS 2.4 NS /. CONSTANT Kv    \ RATE GAIN -.04/2.4 (S)
4500 NS CONSTANT Kf              \ FIBER GAIN (S)
2020 NS CONSTANT PZ              \ POSTION ZERO (S)
2047 NS CONSTANT SGZ             \ STRAIN GAUGE ZERO (S)

2 CONSTANT POSG%                 \ POSITION INTEGER(%) GAIN
1 NS 204.7 NS 2 NS POSG% ** *.
/. CONSTANT PGAIN                \ POSITION IN VOLTS
3.2 NS CONSTANT PCAL             \ VOLTS TO RADIANS

3 CONSTANT RATEG%                \ RATE INTEGER(%) GAIN (N)
1 NS 20.47 NS 2 NS RATEG% ** *.
/. CONSTANT RGAIN                \ RATE IN VOLTS (S)
-72 NS 2 NS PI *. *. 60 ns /. CONSTANT RCAL  \ VOLTS TO RADIANS (S)

3 CONSTANT FIBG%                 \ FIBER INTEGER(%) GAIN (N)
1 NS CONSTANT FGAIN              \ FIBER GAIN (S)
```

```
.00025 NS  CONSTANT FCAL              \ FIBER CALIBRATION (S)     ***NOTE CHANGE

4 CONSTANT CELL                  \ EACH VARIABLE SPACE IS 4 BYTES WIDE
: CELL* ( N – N )   CELL *  ;          \ IT IS 2 ON THE RTX 2000



\  ******************************VARIABLES**********************************

\ PROVIDE AN ARRAY TO CAPTURE GRAPHING DATA
CREATE POSITION 605 ALLOT    \ POSITION INPUT FROM POTENTIOMETER
CREATE RATE 605 ALLOT        \ RATE INPUT FROM TACH
CREATE FIBER 605 ALLOT       \ FIBER SIGNAL INPUT
\ **********************Create space and define test Userspace variabes*****************
 Variable Userspace 31 4* Vallot
: U@ ( N -- S )
   4* USERSPACE + @ ;
: U! ( N -- )
  4* USERSPACE + ! ;
\  ************TEST  VARIABLE  ***************
VARIABLE A/D 4 CELL* VALLOT

VARIABLE FAVE%
VARIABLE RAVE%
VARIABLE PAVE%
VARIABLE TIMER


\  ************  USER  SPACE  VARIABLES  ************************************
0 CONSTANT FBZ%
1 CONSTANT RZ%
2 CONSTANT PZ%
3 CONSTANT FB
4 CONSTANT FBOLD
5 CONSTANT FBF
6 CONSTANT FBFOLD
7 CONSTANT FBFOLD1
8 CONSTANT FBFOLD2
9 CONSTANT FBFOLD3
10 CONSTANT P
11 CONSTANT POLD
12 CONSTANT POLD1
13 CONSTANT POLD2
14 CONSTANT POLD3
15 CONSTANT POLD4
16 CONSTANT POLD5
17 CONSTANT POLD6
18 CONSTANT UF
19 CONSTANT UFOLD
20 CONSTANT UFOLD1
```

```
21 CONSTANT UFOLD2
22 CONSTANT UFOLD3
23 CONSTANT L1
24 CONSTANT L1OLD
25 CONSTANT L1OLD1
26 CONSTANT L1F
27 CONSTANT L1FOLD
28 CONSTANT L1FOLD1



\   **********************************INITILIZATIONS*****************************

: CLRVAR                \ CLEAR VARIABLES BY WRITING ZERO'S
   USERSPACE 31 CELL* 0 FILL
;
\ ********** TEST I/O *************
: ADIN ( CH, GAIN% -- N%) DROP  CELL* A/D + @ ;       \ TEMPORARY A/D INPUT WORD
: DAOUT ( CH, V% -- ) DROP DROP ;         \ TEMPORARY D/A OUTPUT WORD
: TEST1  2000 A/D !                \ STORES 2000 IN A/D ONPUT VARIABLES
      2000 A/D CELL + !             \ USE TO ESTABLISH ZERO'S OF P%, R%, FB%
      2000 A/D CELL 2* + !
;
: TEST2  2300 A/D !                \ AFTER ZEROING, USE AS A/D INPUTS
      2200 A/D CELL + !
      2100 A/D CELL 2* + !
;


\ SET FIBER AND TACH ZEROS BY SAMPLING 20 PTS AND AVG
: 20!
0 FAVE% !
0 RAVE% !
0 PAVE% !
 20 0 DO
   0 FIBG% ADIN  FAVE% @ + FAVE% !
   1 RATEG% ADIN  RAVE% @ + RAVE% !
   2 POSG% ADIN  PAVE% @ + PAVE% !
LOOP ;

: ZERO'S 20!
   FAVE% @ 20  / FBZ% U!
   RAVE% @ 20  / RZ% U!
   PAVE% @ 20  / PZ% U!
   CR ." UNSCALED RATE ZERO = " RZ% U@ .
   CR ." UNSCALED FIBER ZERO = " FBZ% U@ .
   CR ." UNSCALED POSITION ZERO = " PZ% U@ . ;

: INIT   TEST1 ZERO'S TEST2      \ INITIALIZE A/D, AND INPUT VALUES
;
```

```
: READY
   CLRVAR            \ CLEARS VARIABLES
   INIT              \ INITIALIZES A/D AND ZEROS D/A
;

\  **********************RUN  TIME  CODE***********************************

\ INPUT SIGNALS
: INPUTAD ( -- N,N )        \ LEAVES ( -- FB%,P% )
0 POSG% ADIN
2 FIBG% ADIN  ;

\ SOFTWARE STOP
: SOFTSTOP ( N -- N)              \ USES N= P%
   DUP 2350 > IF 0 ZERO% DAOUT
      CR ." ERROR "  ." PRESENT POSITION IS TOO LARGE, P= " . ABORT
     THEN
   DUP 1700  < IF 0 ZERO% DAOUT
   CR ." ERROR "  ." PRESENT POSITION IS TOO SMALL, P= " . ABORT
THEN ;

\ *****************SET UP PROCESSING OF CALIBRATION AND
FILTERING*************
: FBPROC ( N -- )                 \ USES FB%

   FBZ% U@ - NS FGAIN *. FCAL *. DUP FB U!
   FBOLD U@ SWAP OVER - [ .998 NS ] LITERAL *.  + FBF U!
;
: PPROC ( N -- )                  \ USES P%
   PZ% U@ - NS PGAIN *. PCAL *. DUP P U!
   POLD U@ + POLD1 U@ + POLD2 U@ +
   POLD3 U@ + POLD4 U@ + POLD5 U@ + POLD6 U@ +
   [ 7 NS ] LITERAL /. ;

: CAL&FIL ( N,N -- S )            \ CALIBRATION AND FILTER USES (P%,FB% -- PF)
   FBPROC
   PPROC
   ;

: CONTROLLAW ( S -- S )           \ USES ( PF -- L1 )
Kp *. L1 U!
FBF U@ [ 3.743 NS ] LITERAL FBFOLD U@ *. - FBFOLD1 U@ [ 5.394 NS ] LITERAL *. +
FBFOLD2 U@ [ 3.556 NS ] LITERAL *. - FBFOLD3 U@ [ .9506 NS ] LITERAL *. +
Kf *. UFOLD U@ [ 3.127 NS ] LITERAL *. + UFOLD1 U@ [ 3.621 NS ] LITERAL *. -
UFOLD2 U@ [ 1.842 NS ] LITERAL *. + UFOLD3 U@ [ .3486 NS ] LITERAL *. + DUP UF !
L1 U@ + DUP L1 U! ;

\ PREVENT OVERFLOW
: OUTCHK ( S -- S )               \ USES ( L1 -- L1 )
DUP 10 NS > IF [ 10 NS ] LITERAL L1 U! THEN
DUP -10 NS < IF [ -10 NS ] LITERAL L1 ! THEN ;
```

```
: LPF ( S – S )                    \ INPUT SHAPING LPF USES ( L1 – LIF )
L1F U!
L1OLD U@ [ .021 NS ] LITERAL *. L1OLD1 U@ [ .0183 NS ] LITERAL *. +
L1FOLD U@ [ 1.648 NS ] LITERAL *. + L1FOLD1
U@ [ .6873 NS ] LITERAL *. - DUP L1F U! ;

: OUTPUT ( S – S )                 \ USES ( L1F -- )
[ 204.7 NS ] LITERAL *. [ 2047 NS ] LITERAL + SN
0 SWAP DAOUT ;

\ STORE PLOTTING DATA
: STORPLOT ( Index -- ) 5 MOD 0= IF
   P U@ POSITION !
   FB U@ FIBER ! THEN ;


: DIFFEQNVARS              \ SET DIFFERENCE EQUATION VARIABLES
L1OLD U@ L1OLD1 U!
L1 U@ L1OLD U!
L1FOLD U@ L1FOLD1 U!
L1F U@ L1FOLD U!
UFOLD2 U@ UFOLD3 U!
UFOLD1 U@ UFOLD2 U!
UFOLD U@ UFOLD1 U!
UF U@ UFOLD U!
FB @ FBOLD U!
FBFOLD2 U@ FBFOLD3 U!
FBFOLD1 U@ FBFOLD2 U!
FBFOLD U@ FBFOLD1 U!
FBF U@ FBFOLD U!
POLD5 U@ POLD6 U!
POLD4 U@ POLD5 U!
POLD3 U@ POLD4 U!
POLD2 U@ POLD3 U!
POLD1 U@ POLD2 U!
POLD U@ POLD1 U!
P U@ POLD U! ;

\ MAIN LOOP

: MAINLOOP 3000 1 DO          \
INPUTAD              \ INPUT SIGNALS
SOFTSTOP             \ CHECK FOR BAD INPUT DATA
CAL&FIL              \ CALIBRATION AND FILTER
CONTROLLAW                 \ CONTROL LAW
OUTCHK               \ PREVENT OVERFLOW
LPF                  \ INPUT SHAPING LPF
OUTPUT               \ OUTPUT TO D/A
I STORPLOT                 \ STORE PLOTTING DATA
DIFFEQNVARS                \ UPDATE DIFFERENCE EQUATION VARIABLES
```

```
LOOP
;

\ TIME CHECK
: TIME1!                \ START TIMER
  CALL TICKCOUNT TIMER !
\   TC1!                \ RTX EQUIV
;

: TIME1@                \ STOP TIMER
  CALL TICKCOUNT TIMER @  - NS 60 NS /. CR ." TIME = " .#
\   TC1 @               \ RTX EQUIV
;

: GO                    \ START PROCESS WORD
   TIME1!               \ STARTS TIMER NO. 1
   MAINLOOP             \ MAINLOOP
  TIME1@                \ RECORDS AND PRINTS EXECUTION TIME
   ;
```

# References


1. McKeeman J., Lecture "Smart Structures" EE6100, Spring 1988.
2. Thursby M., Grossman B., "Smart Structures Incorporating Artificial Neural Networks, Fiber Optic Sensors and Solid State Actuators," Proceedings for SPIE International Symposium and Exhibition on Fiber Optics, Optoelectronics and Laser Applications, (Boston MA), September 1989.
3. Maher M., DeWeerth S., Mahawold M., Mead C., "Implementing Neural Architectures Using Analog VLSI Circuits," *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 5, May 1989,pp. 643-652.
4. Culhane A., Peckerar M., Marrian C., " A Neural Net Approach to Dicrete Hartley and Fourier Transforms," *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 5, May 1989,pp. 695-703.
5. Habib M., Akel H., " A Digital Neuron -Type Processor and Its VLSI Circuits," *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 5, May 1989,pp. 739-746.
6. Thomas D., Cox D., Lindner D., Claus R., " Optical Fiber Sensors and Signal Processing for Intelligent Structure Monitoring," Fiber and Electro-Optics Research, Center Bradley Department of Electrical Engineering, Virginia Tech, Final Report for NASA, July 1989.
7. Doug Jaeger, "T-Compiler, A Flexible Cross-Compiler for System Development," Future Inc. Document, May 1989.

8. VanLandinham H., *Introduction to Digital Control Systems,* Macmillan, New York, pp. 99-103. 1985.

9. Kuo B., *Automatic Contol Systems,* Prentice Hall, New Jersey, 1982.

10. Celano T., Lindner D., " Design of Low Oder Compensators for Vibration Suppression in Flexible Structures," submitted to *Journal of Guidance Control and Dynamics,* June !989.

11. Butter C., Hocker G., " Fiber Optics Stain Guage," *Applied Optics,* Vol. 17, 1978, pp. 2867-9.

12. Harris Semiconductor, Data Sheets, RTX-2000. 1988.

13. TRW LSI Products Division Data Sheet, TDC 1012, 1987.

14. Short K., *Microprocessors and Programmed Logic,* Prentice Hall, New Jersey,1981.

15. Crystal Semiconductor, *Smart Analog Data Book,* 1988.

The vita has been removed from the scanned document