This is to certify that the

dissertation entitled

IMPROVING THE EFFICIENCY OF DATA COLLECTION
FOR MASS SPECTROMETRY/MASS SPECTROMETRY

presented by

Michael Joseph Kristo

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Chemistry

Christie G Enke
Major professor

Date September 9, 1987

IMPROVING THE EFFICIENCY OF DATA COLLECTION FOR MASS SPECTROMETRY/MASS SPECTROMETRY

By

Michael Joseph Kristo

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Chemistry

1987

ABSTRACT

IMPROVING THE EFFICIENCY OF DATA COLLECTION
FOR MASS SPECTROMETRY/MASS SPECTROMETRY

By

Michael Joseph Kristo

The development of triple quadrupole mass spectrometry (TQMS) has led to the widespread acceptance of mass spectrometry/mass spectrometry (MS/MS) as a useful analytical technique. This dissertation explores improvements to the TQMS instrument and new types of TQMS experiments which improve the efficiency of data collection. These improvements in TQMS fall into four categories: improving the speed of computer control, improving the accuracy and speed of data reduction, improving the accuracy and dynamic range of ion current measurements, and improving the detection of collisionally assisted reaction (CAR) products.

The speed of computer control has been improved with the development of two control systems, a multi-microprocessor control system and a system based on a high-speed FORTH processor. Both systems achieved faster scan speeds and greater amounts of signal averaging than conventional control systems.

The task of reducing sequential intensity data to position/intensity pairs for each peak in the mass spectrum (peak-finding) is especially difficult in MS/MS, because of the increased dynamic range, the variable resolution, and

the varied peak shapes. However, several key features were found to increase the reliability of peak-finding algorithms for MS/MS. The speed of peak-finding was increased substantially with the design and implementation of a programmable electronic peak-finder.

Simultaneous acquisition of analog and ion-counting data with a dual output continuous dynode ion multiplier was implemented. This dual mode control system achieves the full dynamic range available in MS/MS (10^9) and provides absolute ion intensities. This system corrects errors in ion-counting due to pulse overlap and eliminates the effect of variations in multiplier gain with ionic species.

A method of trapping ions and varying their average residence time in the central quadrupole of a TQMS instrument has been developed and characterized. Lengthening the residence time in the collision chamber increases the yield of stable CAR products and allows the observation of many products not seen in conventional CAR.

ACKNOWLEDGMENTS

I would like to thank Professor Chris Enke for his guidance on this project, as well as the freedom and support needed to bring it to fruition. I would also like to thank Professor Victoria McGuffin for the outstanding job that she did for me as second reader. I further acknowledge the challenging members of my committee- Professors David Fisher, Jack Watson, and William Reusch.

Here also, I would like to acknowledge the help and guidance of my fathers in faith, Drs. Bruce Newcome and Carl Myerholtz. Their moral support and technical expertise helped me during the darkest days. Bruce was especially key to the finishing of the multiprocessor, the design of the hardware peak-finder, and the initial design of many of the instrument interfaces for the Novix control system.

I would not have made it this far, though, without the help of my other friends and cohorts- Keiji Asano (the Ace-Man), Pete Palmer (Beware the Dwarves!), Dan Sheffield (who is responsible for several drawings in this dissertation), Mike Nawrocki, and Chris Marsh, and the few women in my life, especially Karen Reeves who helped me make it through the past several months intact mentally (relatively). The good times that we shared helped make the bad times a little

more bearable. I would especially like to thank my parents for their help and support.

I also acknowledge the initial guidance of Dr. Frank Swicker of Christian Brothers High School, Memphis; the Chemistry electronics and machine shops; Marty Rabb, the departmental electrical engineer; El Azteco Restaurants, purveyors of fine food at reasonable prices; Charles Moore, for his genius in inventing the FORTH language; the proprietors of Paul Revere's Tavern, home of the Big Mick; the Pepsi and Coca Cola bottlers of America, Coke Adds Life It's the Real Thing; the makers of fine alcoholic beverages the world over; Domino's Pizza ; and the many others behind the scenes who must remain nameless.

Thank our Gracious Lord That's Over. (Also for rain, sea, and much summer thunder.)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

A MULTIMICROPROCESSOR CONTROL SYSTEM FOR A TRIPLE QUADRUPOLE MASS SPECTROMETER

1. Background

The Triple Quadrupole Mass Spectrometer

The development of triple quadrupole mass spectrometry (TQMS) by Yost and Enke (1-3) has enhanced the widespread acceptance of tandem mass spectrometry as a useful and practical method of analysis. Greater concern over instrument control, intelligent data collection, and meaningful treatment of collected data has been an additional consequence, which arises from the nature of TQMS itself. The TQMS instrument, shown in Figure 1.1, has 5 distinct scan modes, 22 physical devices and 4 software attributes (Table 1.1) which must be controlled during scanning. Furthermore, the TQMS instrument can generate large amounts of data very quickly. All of these qualities make TQMS a model problem in instrument control and data treatment (storage, retrieval, and analysis).

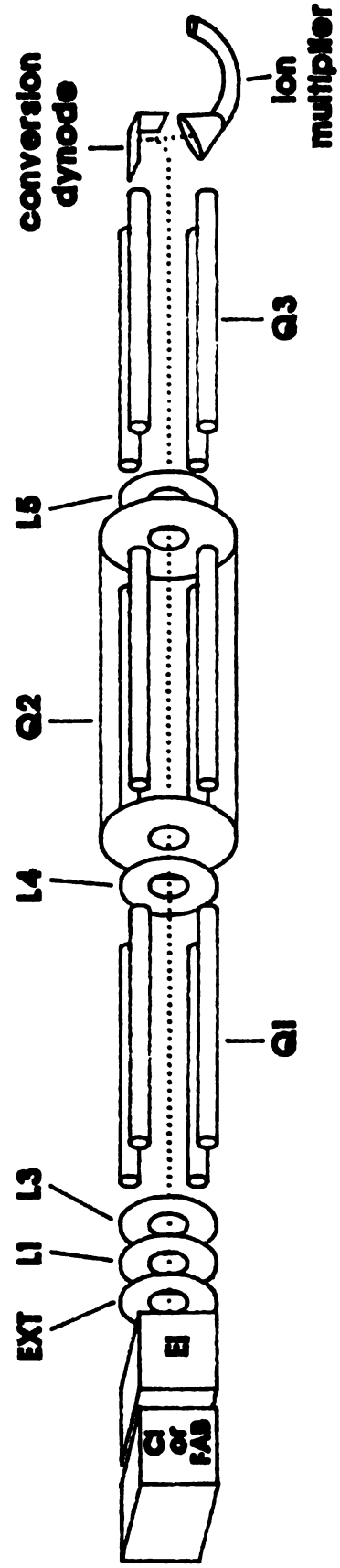


Figure 1.1 The TQMS Instrument

Table 1.1

TQMS devices and their mnemonic names

NAME	DEVICE	NAME	DEVICE
EV	Electron energy	Q2	Quad 2 DC offset
REP	Repeller	Q3	Quad 3 DC offset
EIV	EI ion volume	MHV	Multiplier voltage
CIV	CI ion volume	M1	Mass for quad 1
EXT	Extraction Lens	DM1	Quad 1 delta mass
L1	Ion source lens 1	RS1	Quad 1 resolution
L2	Ion source lens 2	M2	Mass for quad 2
L3	Ion source lens 3	M3	Mass for quad 3
L4	Interquad lens 1-2	DM3	Quad 3 delta mass
L5	Interquad lens 2-3	RS3	Quad 3 resolution
Q1	Quad 1 DC offset	P2	Quad 2 pressure

TQMS Software Attributes

NAME	ATTRIBUTE
THR	Threshold
RTE	Scan Rate
PWD	Minimum Peak Width
MWD	Maximum Peak Width

A TQMS instrument basically consists of an ion source, three quadrupole mass filters, and an ion detector, usually an ion multiplier. The TQMS instrument may also have several lenses to focus the ion beam. The first and third quadrupole mass filters can be operated in either the integral (RF) mode, which passes ions of all mass-to-charge ratios, or the normal (DC) mode, which passes ions of a specific mass-to-charge ratio (mass filtering). The transmission characteristics in the normal mode are further controlled by the resolution and delta mass controls of the quadrupole controllers. The central quadrupole, also called the collision cell, always operates in the integral mode in TQMS and is used for changing the mass of the ions

transmitted by the first quadrupole. This mass change can occur through various methods, including collision of the incident ions with an inert collision gas (CAD- collisionally-activated dissociation) (1), reaction of the incident ions with a reactive collision gas (CAR- collisionally-assisted reaction) (4,5), and absorption of light (LD- laser dissociation) (6,7).

Scan Modes

The option of operating quadrupoles one and three in either RF or DC mode during a scan yields the five different scan modes. Scanning either the first or third quadrupole in the DC mode, while holding the other quadrupole in the RF mode, generates a primary mass spectrum, similar to a conventional mass spectrum. Scanning quadrupole one in this manner (commonly called a 1SCAN) yields identical information to scanning quadrupole three (3SCAN).

If both quadrupoles one and three are held in the DC mode, then three mass spectrometry/mass spectrometry (MS/MS) scan modes are possible (Figure 1.2). In two of the MS/MS modes, one quadrupole passes only ions of a specific mass-to-charge ratio, while the other is scanned. If quadrupole one is held fixed, a daughter scan (DSCAN) is generated, which identifies the mass-to-charge ratio of all ionic species produced from the modification of the ions selected by quadrupole one. If quadrupole three is held fixed, a parent scan (PSCAN) is generated, which identifies the mass-

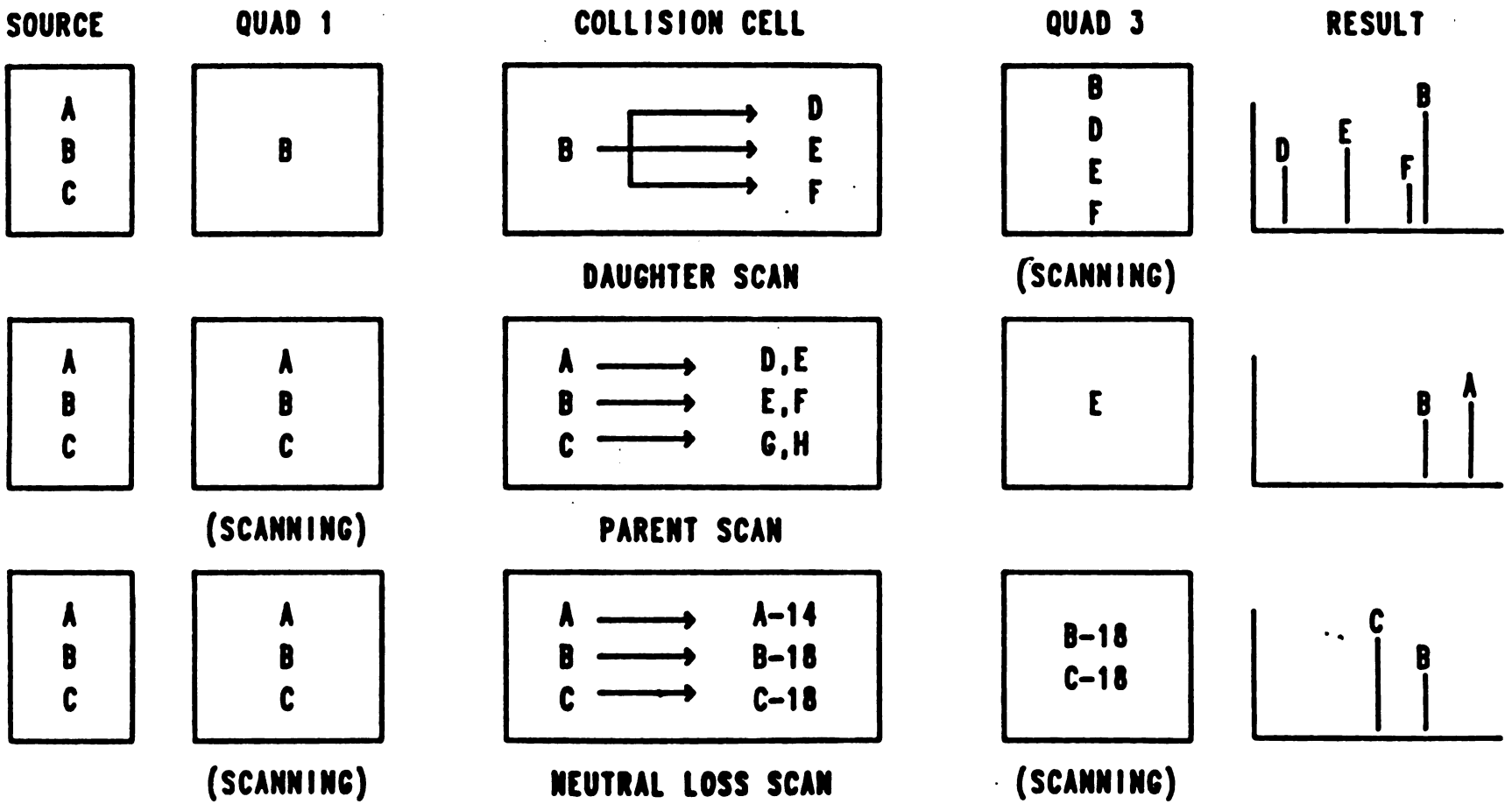


Figure 1.2 TQMS MS/MS Scan Modes

to-charge ratio of all ions which produce ionic species of the mass-to-charge ratio selected by quadrupole three upon modification. In the third mode, quadrupoles one and three are scanned simultaneously, usually with a fixed mass offset producing a neutral loss scan (NSCAN). An NSCAN identifies all parent ions which undergo modification, producing products which have either gained or lost a specific mass fragment.

Since one can generate a daughter spectrum for each selected parent ion, the triple quadrupole mass spectrometer provides three dimensions of intensity data (parent ion mass, daughter ion mass, and intensity). This three dimensional field can be scanned in various 2-dimensional slices, yielding the MS/MS scan modes described above. In addition, the values of the various physical entities in the ion path (source voltages and collision gas pressure, for instance) can change the spectra. This yields a potential multidimensional data field.

The TQMS instrument provides useful chemical information in two ways (2). First, the TQMS instrument can elucidate the structure of unknown compounds, since the additional fragmentation step can help to identify fragment ions in the primary mass spectrum, possibly establishing the existence of certain substructures in the molecule. Second, the TQMS is useful in direct mixture analysis. If a method of ionization is chosen which produces predominately ions characteristic of the intact molecule (the molecular ion or

the protonated molecule), then the first quadrupole can separate the components of the mixture. Fragmentation and mass analysis in the third quadrupole can produce identifying structural information on this component.

Prototype TQMS Instrument

All of the developments in multimicroprocessor control of a triple quadrupole mass spectrometer described in the rest of this chapter were conducted on the prototype instrument in our laboratory. This TQMS instrument consists of: 1) a dual EI/CI Simulscan ion source with an accompanying Ion Controller/Ion Optics module (8), 2) a three-element Einzel lens (L1,L2,EXT), 3) three Extrel 3/8-inch quadrupoles with 150-QC quadrupole controllers (8), 4) single-element lenses between quadrupoles one and two (L3) and between quadrupoles two and three (L4), and 5) a Galileo Channeltron (9) ion multiplier. The three quadrupoles, the interquad lenses, and the ion multiplier are arranged in a removable "stack" configuration. The TQMS vacuum chamber has three regions, which are differentially-pumped by turbomolecular pumps. Before completion of the multimicroprocessor project, this prototype instrument was refurbished by replacing homebuilt parts (ion source controller and central quadrupole) with commercially available ones and upgrading older parts with newer ones more amenable to computer control (ion source and quadrupole controllers).

Comparison studies for a single microprocessor control system were conducted on an Extrel (8) 400/3 TQMS system. This system is identical in configuration to the prototype instrument, except that the vacuum chamber has only two differentially-pumped regions: the ion source region is pumped by a turbomolecular pump and the analyzer region is pumped by a diffusion pump.

Computer Control of the TQMS Instrument

It was obvious from the start of the TQMS project that only a computer could maintain tight control over all of the values for the physical devices during the wide range of possible experiments. Several benefits accrue from a computer control system. First, the control system can be easily programmable by the users of the instrument to meet changing experimental needs. Second, the control system offers flexibility in the methods of user control. For instance, the user can change the voltage on a lens by entering the new value of the voltage into a tabular menu, using a computer-controlled knob, or issuing a direct command. Third, the control system can be helpful in optimizing and keeping track of instrument parameters. For instance, in real-time, the computer can display information on the effect of a particular parameter on the ion current or display peaks widely separated in mass for more accurate tuning of the instrument. Fourth, the computer can can control the instrument and generate data very quickly. This

speed further allows a large degree of intelligent real-time control over instrument parameters or experiment selection.

The first computer control system for our TQMS instrument was based on the SDK-85 microcomputer (10), but was subsequently replace by an 8085-based microcomputer of the Newcome-Enke design (11). The Newcome-Enke microcomputer was the initial development in a long-term design project to create a modular microcomputer system for scientific instrumentation (12). The 8085 microcomputer was eventually replaced with a more powerful 8088 module for the same bus and, finally, with a distributed processing system consisting of four 8088 Newcome-Enke type microcomputers, one master and three slaves (12,13).

The microcomputers themselves were capable of running many different operating systems or language environments. Thus, programs written in a conventional high-level language like BASIC or FORTRAN could have controlled the instrument, but they have several shortcomings which limit their usefulness as languages for instrument control. First, an ideal instrument control language should be fast, approaching the speed of assembled code, maybe even incorporating an assembler for time-critical operations. Both compiled BASIC and FORTRAN are generally considered slow, because of the inefficient code created by compilation. Interpretive BASIC is even slower, since the text for the BASIC programs are interpreted at the time of program execution. Second, an ideal instrument control

language should have an interpretive nature, so that code can be easily modified for ease in debugging and trouble shooting. BASIC can be interpretive, but FORTRAN, being compiled, is difficult to modify. Third, an ideal instrument control language should allow the creation of an application-specific syntax, providing the commands with a highly mnemonic content specific for the instrument operations being controlled and appropriate to a user unfamiliar with programming. Fourth, the ideal control language should be extensible, that is, the user should have the ability to create new programs by merely concatenating predefined commands, allowing him to create sequences and methods for frequent use. Neither FORTRAN or BASIC is inherently extensible and the best that can be done for an application-specific syntax is to give the programs a mnemonic name.

Instrument control languages can, of course, be fashioned from assembly-level routines. However, the assembly-level programming needed to create such an instrument control language is laborious and time-consuming. Using a language which has the low-level control features of an assembler (direct control over registers and memory locations) with the input/output routines of a higher-level language allows the programmer to create the instrument control language much more efficiently. Two common languages for creating instrument control software are C and FORTH. C has the advantages of widespread popular support

and the abilities inherent in a compiled language, e. g., floating-point support. However, C, being a compiled language, is not easily modified nor is it inherently extensible. FORTH meets all of the above criteria for the ideal instrument control language, but it has several features uncommon in conventional languages. For instance, all FORTH operations require postfix notation, that is, mathematical operators follow their arguments, e. g., 2 3 + would add two and three. Also, FORTH inherently uses a push-down stack, an area of memory in which numbers are stored and removed in a last in-first out (LIFO) basis.

Both of these initial microcomputers ran an instrument control language written by Hugh Gregg called SLOPS (Symbolic Language Operating System) (14), which was cross-assembled from a PDP-11 minicomputer (15). SLOPS was a minimum system which had a kernel of basic subroutines and relied heavily on the network with the PDP-11. SLOPS was an attempt to create a FORTH-like environment without the unusual FORTH features mentioned above.

A decision was eventually made to abandon SLOPS in an attempt to increase the vitality of the system and the ease with which the system could be maintained and repaired. The new software control system was based entirely on the programming language FORTH (16). The choice of FORTH created a robust environment with interactive extensibility for quick and easy modification, inherent modularity, a compiled nature for creating compact code, and an

interactive assembler for maximum speed in vital applications. A flexible software control system for TQMS was developed for a single microprocessor control system as the Master's research of Carl Myerholtz (17). Myerholtz further modified the basic FORTH kernel to accommodate the new multimicroprocessor system as part of his Ph. D. dissertation (13). This author and Adam Schubert adapted the single microprocessor TQMS software control system to the existing multimicroprocessor environment.

The Advantages of Multiprocessing

The TQMS instrument is a textbook example of the trend towards more and more complex instrumentation. These more complex instruments place greater demands on their control systems. This demand, in turn, has fueled a search for greater computing power in the laboratory at reasonable costs. However, increasing computer power by purchasing a single, more powerful processor can be prohibitively expensive. In general, linear increases in computation speed increase the cost of the processor more than linearly. Even with the competition present in the electronics industry today, the fastest microprocessors can cost tens of thousands of dollars (18) and the fastest computers can cost several million dollars (19). The data system can easily become the most expensive part of a computer-controlled instrument. Besides the cost, the larger and more powerful computers were designed primarily for multiuser,

computationally intensive environments and not for real-time instrument control. Using these types of computers for instrument control necessitates the design of awkward and complex interfaces to the instrument itself, further increasing the cost and possibly lowering the overall performance of the system.

An alternative approach is to use several less-powerful computers working together in a distributed processing environment. This type of environment allows separation and distribution of tasks among the different processors. If the overall goal of the multiple processor system has inherent parallelism, that is, subtasks which can be performed concurrently, then several advantages result (Table 1.2). These advantages can be generally classified as faster execution, independent task execution, and modularity in both hardware and software. These advantages occur because separate processors are assigned tasks which can be executed concurrently and are designed with the appropriate hardware and software to accomplish the task. In a distributed processing system, as long as additional parallelism is exploited, computing power increases linearly with the addition of processors of the same computing power. Cost also increases linearly with the addition of extra processors, at least after the purchase of initial interprocessor hardware, resulting in a linear increase in cost with increasing computing power. Again, this only occurs if additional parallelism is exploited.

Table 1.2

Advantages of Distributed Processing Systems

Faster Execution
- Parallel Execution
- Less time spent in "overhead"
- Simpler Addition of hardware controllers and processors

Independent Task Execution
- Non-interference of tasks
- Elimination of task interleaving programs
- Elimination of priority assignment programs
- Simpler task program modification

Modularity of Hardware and Software
- Consolidation of related tasks
- Simpler extension of instrument capability
- Simpler debugging and trouble shooting

There are a variety of possible multiple processor systems. Some systems contain like processors; other systems contain processors of various types. Some systems have dynamic load-sharing, which assigns tasks during program execution; other systems have static load-sharing with the assignment of tasks occurring in the design of the system. Multiple processor systems can also be classified by the manner in which they link their tasks: tightly coupled if the tasks communicate on the microsecond time scale or loosely coupled if the tasks communicate on the millisecond time scale or longer.

Multiple processor systems can also be classified by their interconnection topology (20). Several system topologies are shown in Figure 1.3. The fairly simple ring topology requires only two physical connections for each processor. Communications can be fairly slow, however,

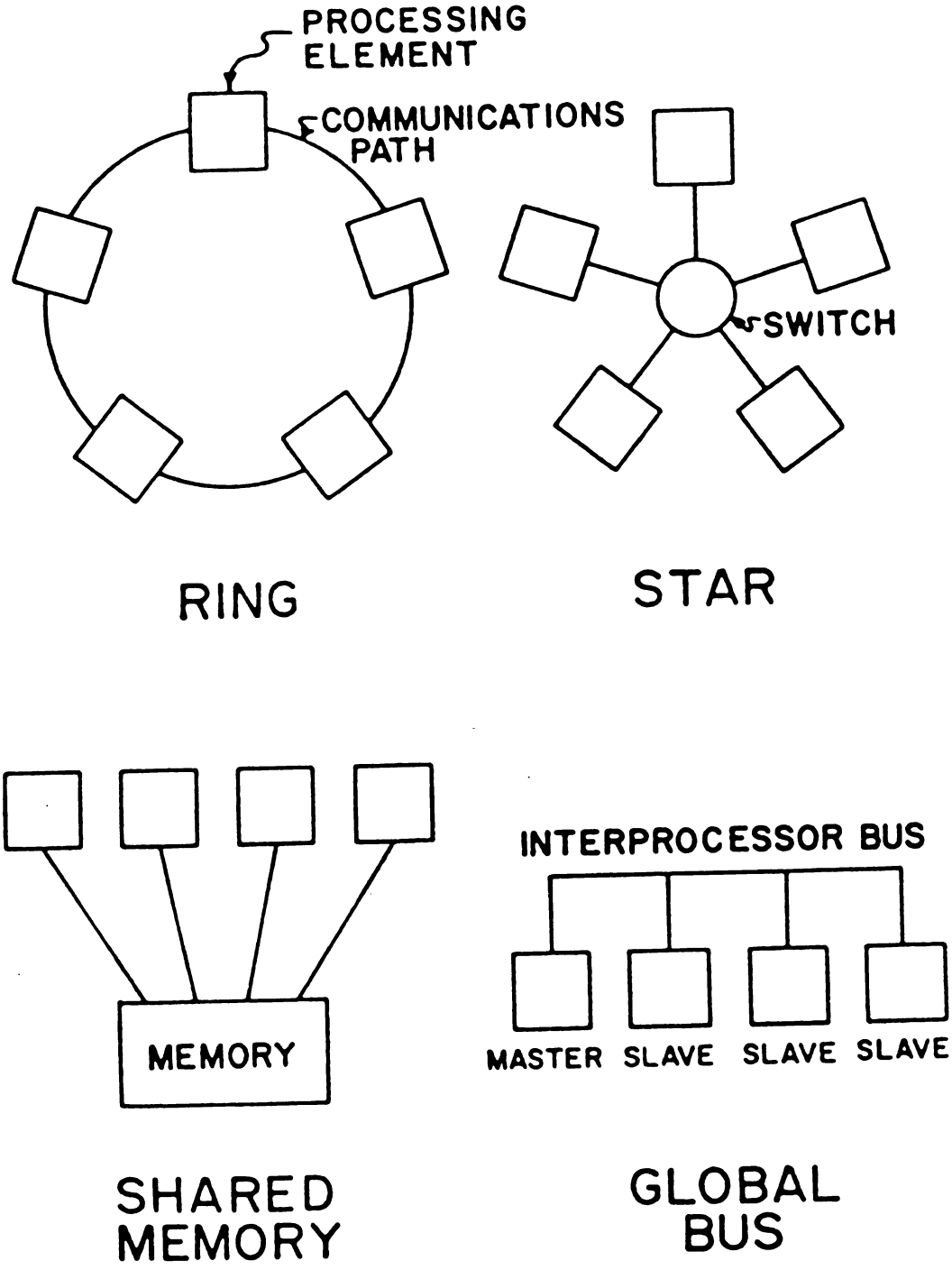


Figure 1.3 Multiprocessor Topologies

since messages between two processors must pass through all intervening processors on the ring. The star configuration uses a switch (either another computer or dedicated hardware) to connect all of the processors in the system. Communications for the star configuration can be moderate to fast, depending on the speed of the switch. Shared memory is a common interconnection technique in which processors communicate by placing messages and data in memory that all processors can access. This is generally a very fast method of communication and synchronization. In the global bus configuration, all processors share a common communications pathway. The global bus is moderately fast; the speed of communications is limited by the bandwidth of the bus (21,22).

TQMS Multiprocessor Controller

The multiple processor system designed to control the TQMS instrument can be classified as a distributed processing system with equal processors and static load-sharing. The triple quadrupole multimicroprocessor system is tightly coupled since much of the synchronization during scanning occurs through the flipflops and first-in first-out buffers of a linking and synchronization circuit on the microsecond time scale. The multimicroprocessor control system employs the global bus topology, where the four microcomputers share a common interprocessor bus for communication. The multimicroprocessor system further

specifies one of the processors as the master and the other three as slave processors. The master has the job of coordinating the activities of all of the slave processors and of communicating with the user and various intelligent peripherals, such as printers and disks.

Three methods of interprocessor communications, beyond the linking and synchronization circuit, are provided in hardware. Bulk data or program transfer can be performed by direct memory transfer (DMT). In DMT, the master places the appropriate slave on hold and directly writes to or reads from the slave's memory. Slave memory actually resides in the upper half of the master's address space.

Command transfer (CT), the transfer of commands and data from the master to the slave, occurs through a series of first-in first-out (FIFO) buffers. The master places the command or datum into the appropriate slave's FIFO buffer and the slave sequentially reads and executes each command (specified as a program execution address) or reads and places each datum on its last-in first-out (LIFO) stack. The FIFO value consists of 24 bits, the high byte of which specifies the lower two bytes as data or a command.

A final method of communication is the status bus (SB). Each microcomputer has a hardware status byte and a software status byte. Each bit in the status bytes is either defined in hardware or can be defined in software. The microcomputer can then write into its own status bytes, informing all other processors of its status. All

processors can read all status bytes. Thus certain bits in the status byte can be used as communication flags, such as the signal for the end of a scan. The SB is updated every 4 microseconds in hardware, so there is a certain latency in using the SB as a flag.

The key to the performance of the multimicroprocessor control system for the TQMS instrument is the separation of tasks into separate processors to exploit the parallelism inherent in scanning a mass spectrometer for data acquisition. The data transform concepts proposed by Yourdon and Constantine (23) formed the criteria for assigning tasks to the various processors. They divided functions into afferent, efferent, and central. Afferent functions convert data input streams into an internally usable format. Efferent functions prepare internal data for output transmission. Central functions perform internal operations on the data. In scanning a triple quadrupole mass spectrometer, controlling all of the device values in the ion path is an efferent function; acquiring and formatting ion current data is an afferent function; and data reduction and interpretation is a central function.

Slave 1, also known as the Ion Path Slave, performs the afferent function of controlling all physical devices. It plays a central role in optimizing the conditions of an experiment- "tuning" and in scan generation. Slave 1 is, therefore, provided with eight 12-bit DAC outputs as well as digital control lines to the ion source controller. The ion

source controller maintains the voltages on all of the regions and lenses of the source. Slave 1 also has access to remote DAC outputs through a differential transceiver circuit. The remote DAC outputs control the voltages on the interquad lenses, as well as all signals to the quad controllers' mass command, resolution control, delta mass control, and quadrupole offset voltage. Slave 1 has 32 Kbytes of memory, enough to permit extensive look-up tables for tracking devices other than the primary device being scanned (linked-device scans). In this way, performance of the instrument can be optimized in software for different mass regions during the scan itself.

Slave 2, the Reduction Slave, controls the central function of data reduction and display. It receives device values for the scanned device from Slave 1 and ion intensity values from Slave 3, reduces them with a peak-finding algorithm (24) into peak position/peak intensity pairs, and stores them in a data buffer. The master can then retrieve the data with a DMT. Slave 2 can also perform real-time or offline graphics display. The user can then see the data that he or she is collecting as it is accumulating, as well as check up on the peak-finding algorithm by matching actually acquired data with peak data on the real-time display. Slave 2 is equipped with a NEC 7220 graphics controller (25), an Intel 8087 numeric coprocessor (10) for complex numerical calculations, and 48 Kbytes of memory to support a large data buffer.

Slave 3, the Detection Slave, performs the afferent functions of data acquisition and formatting. It controls an intelligent data acquisition circuit designed by Dr. Bruce Newcome (26). This data acquisition circuit converts a selected analog voltage (the preamplified ion current) to a 20-bit integer intensity value. The data acquisition circuit also performs a user-selectable number of conversions and returns the 20-bit average. Slave 3 has 16 Kbytes of memory, an interface for the data acquisition circuit, an 8087 numeric coprocessor, and a versatile counter/timer circuit used in ion counting. Slave 3 performs the vital role in dual mode acquisition which will be described in Chapter 5.

Figures 1.4 and 1.5 show the timing of the sequence of functions for two types of scanning and data acquisition in the multimicroprocessor system: sweeps (Figure 1.4), which acquire raw intensity data, and scans (Figure 1.5), which reduce the raw intensity data to mass spectral peak positions/peak intensities (peak-finding). In a sweep, almost all functions have a predecessor-successor relationship and there is little inherent parallelism. Slave 1 sets the new value for the scanned device, notifies Slave 3 that it has done so, writes the new value to Slave 2, and waits for a signal from Slave 3 to proceed. When Slave 3 receives notice that the scanned device has been stepped, it acquires a new intensity value, notifies Slave 1 that the acquisition is done, writes the new value to Slave

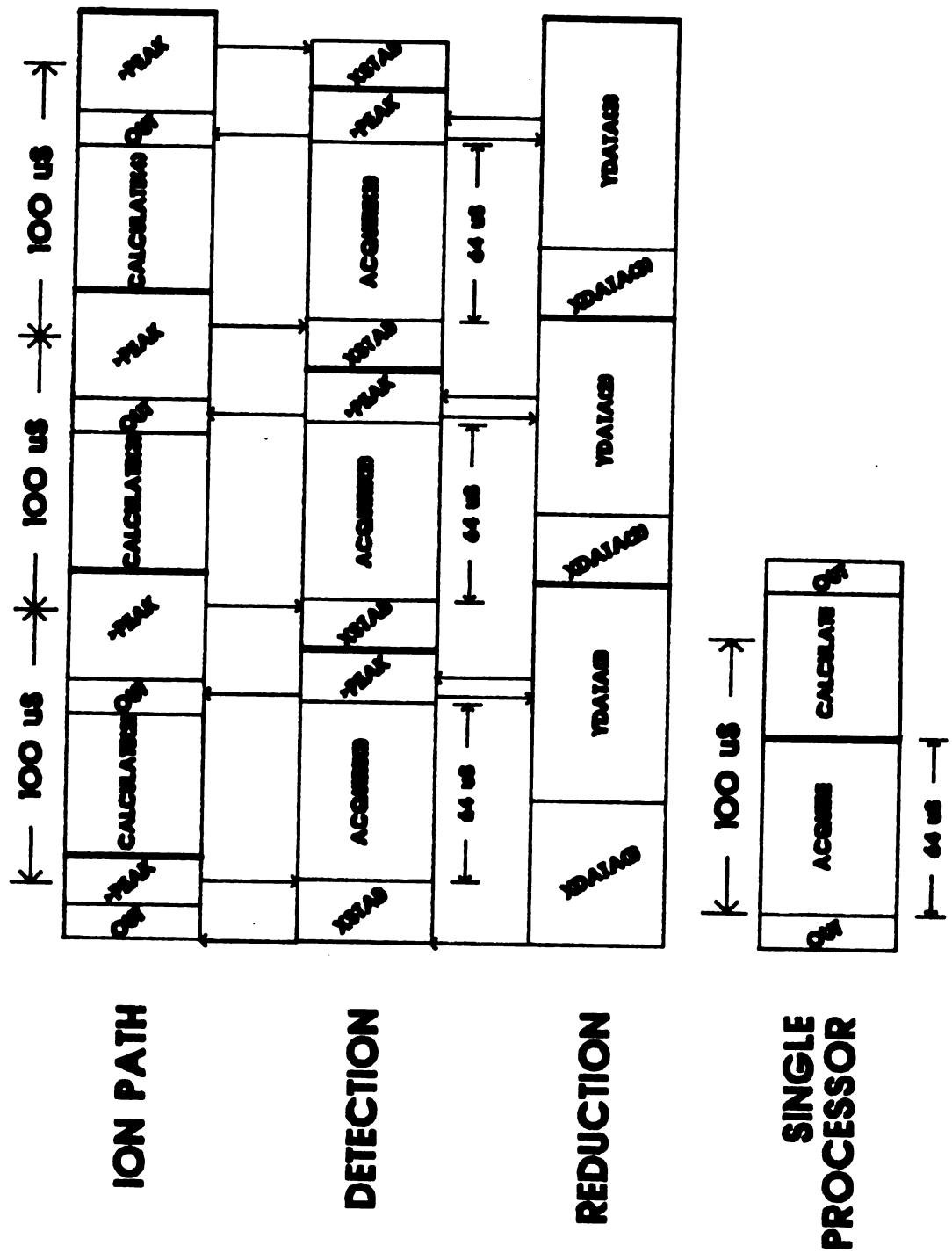


Figure 1.4 Timing Relationships for a SWEEP

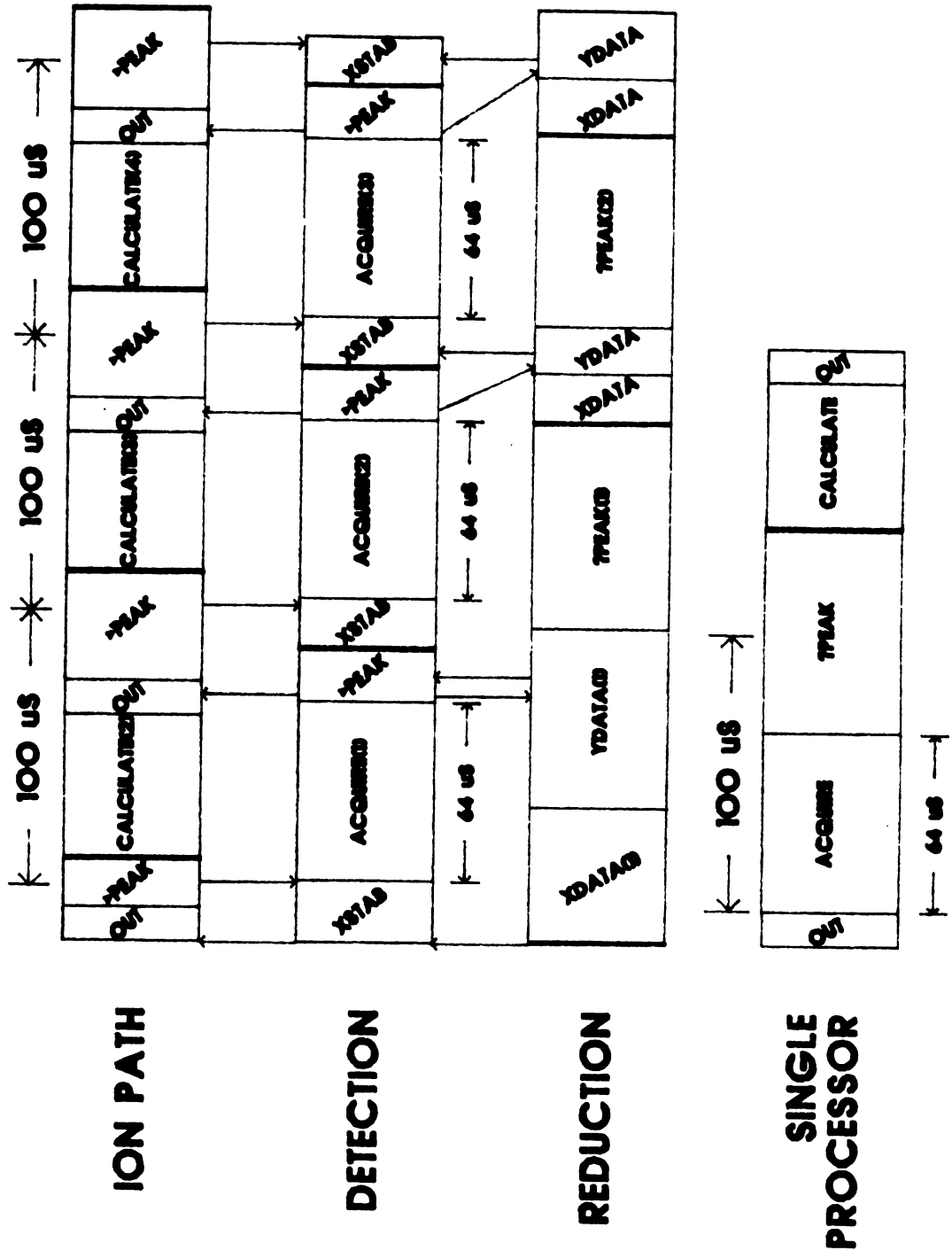


Figure 1.5 Timing Relationships for a NSCAN

2, waits for the signal that Slave 2 has received and processed the new intensity value, and then waits for notification that there is a new device value. Slave 2 receives device values from Slave 1 and intensities from Slave 3 and stores them in a data buffer. The progression for a sweep is: update the scanned device, acquire an intensity value, and store both in a data buffer. This is highly linear, so no speed advantage will be gained from a multiple processor environment (27).

In a scan, data reduction or "peak-finding" is a time-consuming task, but it can be performed in parallel with the setting of ion path values or ADC conversion. The only limitation to its being a wholly parallel function is the necessity of not overflowing the input buffers of the linking and synchronization circuit, thus losing data in the process. Thus, Slave 3 must wait for Slave 2 to acknowledge receipt of the intensity datum, in order to keep all processors "in step." The activities of Slave 1 and Slave 3 are the same during a scan as during a sweep, but Slave 2 performs peak-finding and only stores peak data in the data buffer. Data reduction is still the main limitation to greater scan speeds, even in a multiple processor system. Since much data reduction can be performed in parallel, there is a distinct increase in scanning speeds for the multimicroprocessor control system over the single microprocessor control system (27).

2. Completion of the Multimicroprocessor Control System

The project for this thesis was, in theory, simple: to take the existing interprocessor software and hardware, as well as the first-approximation multiple processor triple quadrupole control system of Adam Schubert, and make them work. A second goal was to infuse a degree of reliability into a system made unreliable by its complexity and thousands of handmade solder connections. Also, with the increased computing power of the multimicro control system, several new features were made practical, such as complex linked-device scanning and real-time graphics.

Making the Multiprocessor System Work

At the start of the project, the theory of the TQMS multiprocessor system had been worked out by Newcome and Myerholtz. The interprocessor computer, including many of the instrument interfaces, was complete, although many of the modules had fallen into disrepair. The interprocessor software was also complete. Very little application software for the multiprocessor system had been written, although the methods of user control developed for the single processor control system were merely duplicated in the multiprocessor environment. Furthermore, much of the original equipment on the prototype instrument had been replaced, but not tested. Basically, no module (software, computer electronics, or instrument hardware) could be assumed to be working at the start of this project.

Completion of the multiprocessor system required careful testing of select modules and then using those modules to discover and eliminate malfunctions in other parts of the control system in a repetitive operation.

Several new modules were added to the multiprocessor computer to complete the control system. A 12-bit DAC circuit was added to the Ion Path Slave to control a Model 216 Granville-Phillips Pressure Controller (28) for admitting collision gas into the central quadrupole. An interface for the intelligent data acquisition circuit mentioned earlier was designed and replaced the older 12-bit ADC and formatter circuits on the Detection Slave. This circuit further improved the scanning speed of the multiprocessor system. Also, a pulse-counting circuit was designed to allow the Detection Slave to handle the ion counting necessary for dual-mode acquisition (Chapter 5).

The linking and synchronization circuitry on the Detection Slave had to be modified from the original plans in order to allow addition of an 8087 numeric coprocessor. The original circuit latched the synchronizing strobes from the Ion Path and Reduction Slaves and fed them to the microprocessor's TEST line. The microprocessor could then execute a WAIT instruction, halting the processor until the TEST line went high, indicating that the strobe had been received. This was the fastest way of waiting for the strobes, but the 8087 uses the TEST line to synchronize the two processors. In order to accommodate the 8087, the

latched strobes are now fed into a parallel input/output port, which can be read by the processor until the appropriate bit is high.

Another problem that remained to be resolved was the protocol for ending a scan (or sweep). The three slave processors are intricately linked during the scan, but only the Ion Path Slave has a definite limit to the number of steps in its scanning loop. The other slaves wait in an indefinite loop, testing for the end-of-scan signal after each step in the scan. At first, it was thought that the Ion Path slave would give a general end-of-scan signal to all slaves through the status bus after the last step in the scan. In practice, the Reduction Slave would finish one step too soon, the last point would be missed, and the Detection Slave would be left waiting for confirmation of receipt of the last intensity value. The solution was to have the Ion Path Slave signal the Detection Slave through its software status byte and, then, have the Detection Slave signal the Reduction Slave through its software status byte. After processing the last datum, the Reduction slave notifies the Master processor that the scan is finished through its software status byte.

Another problem which had to be solved was mass calibration. In mass calibration, the control system determines the DAC value (for the mass command input to the quadrupole controller) which corresponds the maximum intensity of a given mass spectral peak. Up to 15 such

peaks are selected by the user (usually the major peaks present in the spectrum of a mass calibrant). The resulting mass calibration table consists of the calibration masses and their corresponding DAC values. The final DAC value determined for each mass is the average value obtained from 10 sweeps over the mass spectral peak. Thus, mass calibration consists of up to 15 sequences of 10 sweeps. After the calibration sweeps for each peak, the user has the option of repeating the sequence, accepting the determined value, or entering a desired value manually. Each mass sweep is synchronized by the linking and synchronization circuitry and the end-of-scan sequence discussed above. Each processor is also in a loop to execute the mass sweep 10 times. Within the loop, each slave is coordinated by the detection slave which writes the index of the loop (the number of the current mass sweep, 1-10). Each slave reads the Detection Slave's status byte, in order to avoid starting the next sweep before the other slaves are ready. After the ten sweeps are finished the Reduction Slave notifies the Master processor through its software status byte. The Master then reads the 10 DAC values which correspond to the point of maximum intensity for the 10 sweeps from the memory of the Reduction Slave, completing the calibration for one peak.

The final step in the completion of the project was the determination of the maximum utilization of each slave in the scanning sequence. First, the number of averages that

could be performed for a given scan rate were empirically determined and are shown in Table 1.3. Furthermore, upon analysis, the Ion Path Slave was found to have extra time in the scan sequence, allowing the development of the linked-device sweeps described later.

Improvement in Scanning Speed

The multimicroprocessor control system, together with the prototype triple quadrupole mass spectrometer, affectionately known as the U. S. S. Enke, now form a viable and powerful TQMS system. Only subsequent users will determine its place in the Enke research program, but the increase in scanning speed and flexibility in adding new features is clear. Tables 1.3 and 1.4 show the resulting increase in scan speed over the existing Extrel 400/3 (8) Multibus-I control system at equivalent signal acquisition times (number of averages), as well as the increased averaging over the Extrel control system for equivalent scan speeds. For further comparison, Finnigan Corporation's new state-of-the-art triple quadrupole mass spectrometer, the TSQ-70 (29) advertises a scan rate of 4000 AMU/second for 10 digital samples per AMU. Tables 1.3 and 1.4 show that the multimicroprocessor system can scan 4 times as fast as the single processor Extrel system and yet provide sixteen times the averaging (an increase in signal-to-noise ratio of four). For equivalent amounts of averaging, the multimicroprocessor system scans 10 (128 or 256 averages) to

25 (1024 averages) times as fast as the Extrel control system.

Table 1.3
Multimicroprocessor Scanning Functions

Rate	Pts/S	Amu/S	Time/Pt	# Averages/Pt
0	10000	1000	100 uS	16
1	5000	500	200	32
2	2500	250	400	64
3	1000	100	1 mS	128
4	500	50	2	256
5	250	25	4	1024
6	100	10	10	2048
7	50	5	20	4096
8	25	2.5	40	4096
9	10	1	100	4096
10	5	0.5	200	4096
11	2.5	0.25	400	4096
12	1	0.10	1 S	4096

Table 1.4
Multibus-I Scanning Functions

Rate	Pts/S	Amu/S	Time/Pt	# Averages/Pt
2	2500	250	400 uS	1
3	1000	100	1 mS	4
4	500	50	2	16
5	250	25	4	32
6	100	10	10	128
7	50	5	20	256
8	25	2.5	40	512
9	10	1	100	1024
10	5	0.5	200	2048
11	2.5	0.25	400	4096
12	1	0.10	1 S	4096
13	0.5	0.05	2	4096
14	0.25	0.025	4	4096

Direct comparison of the multimicroprocessor control system to the Extrel system can be misleading. Several improvements have been added to the multiprocessor system that are not present on the Extrel single processor system, namely the intelligent data acquisition circuit. The Multibus I-based Extrel data system must address its

instrument interface through the slow L-Bus (the peripheral bus for Extrel control systems). Also, some high-level FORTH routines were rewritten in FORTH 8086/8088 assembler, in order to regain some of the speed lost in interprocessor communication. On the other hand, the 5 MHz 8086 Matrox processor on the Extrel system is faster than the 5 MHz 8088 microcomputers used for the master and reduction processors and about as fast as the 8 MHz 8088 microcomputers used for the Ion Path and Detection processors, because it has a 16-bit external data bus as opposed to the 8-bit external data bus of the 8088. Thus, communications outside the microprocessor take place twice as fast with the 8086 as with the 8088 at equivalent clock speeds. The data in Tables 1.3 and 1.4 represent scanning speeds imposed by a rate synchronization circuit- the AMU timer.

Troubleshooting a Multiprocessor System

Liebowitz and Carson (30) define reliability as "the probability that a system will operate for a specified time interval. . . . This reliability is determined by its hardware design, software design, and ease of operation." Our experience has also added construction to this list. Once the software is verified as working, it remains working and does not add to the unreliability of the system. In point of fact, though, once the aged hardware was brought up to speed, the initial cause of unreliability was software bugs. Once these bugs were fixed, hardware malfunctions

again became the primary contributor to system unreliability. There are elements needed in a multiple processor system to improve recovery in the event of hardware failure: detection, isolation, and correction.

First, a problem must be detected. Unfortunately, for a scientific control system, the operator must be the detector for many kinds of instrument and computer failure. For instance, if the data acquisition circuit continually returns null intensity values, this could be a legitimate scan with all zero data or a malfunctioning circuit. Without further information, the computer cannot determine the legitimacy of the null intensity values, so the operator must. Most multiple processor systems are run in continual operation, which makes self diagnosis more important than in the case of the scan-and-stop operation of the multimicroprocessor control system. The master does monitor the status of all slaves through a variable which the slave updates with a unique value upon reset. If the slave is not operating, the value will not be updated. If the slave is operating, but incorrectly, or if it malfunctions later on, it will usually overwrite the correct value in the variable with an incorrect one. Once all slaves are ostensibly working, the most common mode of failure is the "hanging" of the system, in which the system does not respond to input from the keyboard. This could be caused by the occurrence of an unusual event, such as noise on the bus, that threw one of the processors out of synchronization. If so,

resetting the slaves or the entire system should restore everything to normal. Otherwise, a chronic problem could exist which must be diagnosed.

Second, the problem should be isolated. A number of programs have been written to help isolate the problem. In addition, a manual has been written to help present and future users to cope with the complexities of detecting and repairing malfunctions in the multimicroprocessor system. This process should isolate the cause of the problem to a specific processor and a specific circuit.

Third, the problem must be corrected. The debugging manual provides hints for this process. Most multiple processor systems use the correction schemes of redundancy, where spare processors assume the duties of the failed processor, or replacement, where a huge stock of identical circuits is kept to replace failed ones. These approaches have the advantage of being fast, but the disadvantage of being too costly for an academic laboratory. Replacing malfunctioning parts with parts from other Newcome-Enke Bus computers is a viable method of isolation, but keeping a lot of spare parts is not practical for our laboratory.

New Features

Two new features are now possible with the multimicroprocessor control system. The Ion Path Slave computer was not fully utilized in the usual scanning cycle and spent a lot of time waiting for the signal from the

detection slave to proceed. This extra time allows more complex scanning procedures to be used, such as updating more than one or two devices, that is, linked-device scanning. Now, new scan modes allow up to eight devices to be linked in a scan procedure. Unfortunately, real-time calculations of these values is not possible. However, all of the values can be calculated beforehand and stored in a RAM buffer. The processor then updates the devices by sequentially accessing the values in the buffer.

The FORTH code for these new scans can be found in Appendix A. The user controls the algorithm for scanning selected devices through an offline interface. The user first selects the number of devices to be linked, then he specifies the actual devices. Finally, for each device, he can specify a number of mass-device value pairs, that is, the value of that device at a particular value of the mass filter being scanned. The interface then fits these points to a spline curve and calculates the value of the device at integral mass-to-charge ratio values (every 64 DAC units). Specifying the scanning function in this way allows the user to determine experimentally the optimum value for a particular device at a particular mass-to-charge ratio value and use this value to create a complex scanning function without having to create a complex analytical function for the experimental phenomenon. This allows the user to optimize the tune of the instrument for different mass ranges, rather than compromising between conditions

favorable for high-mass sensitivity and conditions favorable for low-mass sensitivity. These tables of calculated device values are stored on disk and written into the Ion Path slave's memory with a DMT. When the user specifies a linked-device scan or sweep (LSWEEP, L1SCAN, L3SCAN, etc.) The scanning routines automatically update the selected device along with the mass filter being scanned.

The second feature is real-time scanning graphics (the FORTH code for these routines can be found in Appendix B). During a real-time graphics sweep or scan, all intensity data are displayed on the graphics monitor as they are received by the reduction slave. This allows the user to check on the peak-finding algorithm as well as receive real-time information on the scan. Real-time sweeps also display their data in real-time. Unfortunately, displaying this increased real-time visual information takes extra time and limits scan rates. The real-time scans and sweeps add the real-time graphics task to the other tasks that the Reduction Processor must handle during a scan, namely, receiving device value and intensity data and reducing or storing that data. Furthermore, graphics display in itself is a very time-consuming task, even with a graphics controller.

3. Conclusion

A multimicroprocessor control system has been fully implemented. It provides a maximum scan speed 1.9 times

faster than the Extrel 400-3 control system. The multimicroprocessor has been made reliable and recovery possible with the implementation of processor self-checking, debugging routines, and the writing of a debugging manual. This multiprocessor system has made possible complex linked-device scanning and real-time scanning graphics, useful tools for the mass spectrometrist.

This work demonstrates the possibilities inherent in multimicroprocessor scientific control by exploiting inherent parallelism in scanning an instrument. Multimicroprocessor control provides a relatively inexpensive way of increasing scientific computing power and, after the initial idea of Newcome, Myerholtz, and Enke, many commercial instruments, such as the TSQ-70 now feature multimicroprocessor control systems. The drawback to multimicroprocessor computers are their inherent complexity and increased unreliability compared to single processor systems. Nevertheless, multiprocessing is an attractive solution to the need for faster and more complex control of scientific instruments. This need will only increase as scientists devise more complex experiments requiring computer control, develop instruments capable of providing data at greater bandwidths, and pursue applications requiring maximum utilization of the ion signal available.

REFERENCES

1. R. A. Yost, C. G. Enke, D. C. McGilvery, D. Smith, and J. D. Morrison, Int. J. Mass Spectrom. Ion Phys., 30 (1979) 127.

2. R. A. Yost and C. G. Enke, Anal. Chem., 51 (1979) 1251A.

3. R. A. Yost and C. G. Enke, Org. Mass Spectrom., 16 (1981) 171.

4. J. D. Morrison, K. Stanney, and J. M. Tedder, J. Chem. Soc Perkin Trans. II, 1981, 838-841.

5. J. D. Morrison, K. Stanney, and J. M. Tedder, J. Chem. Soc. Perkin Trans. II, 1981, 967-969.

6. D. C. McGilvery and J. D. Morrison, Int. J. Mass Spectrom. Ion Phys., 1978, 28, 81-92.

7. M. L. Vestal and J. H. Futrell, Chem. Phys. Lett., 1974, 28, 559-560.

8. Extrel Corp., Pittsburgh, PA.

9. Galileo Corp., Sturbridge, MA.

10. Intel Corp., Santa Clara, CA.

11. B. H. Newcome and C. G. Enke, Rev. Sci. Instrum., 55 (1984) 2017.

12. B. H. Newcome, Ph. D. Dissertation, Michigan State University, 1983.

13. C. A. Myerholtz, Ph. D. Dissertation, Michigan State University, 1983.

14. H. R. Gregg, Ph.D. Dissertation, Michigan State University, 1986.

15. Digital Equipment Corporation, Marlboro, MA.

16. Forth, Inc., Hermosa Beach, CA.

17. C. A. Myerholtz, M. S. Thesis, Michigan State University, 1982.

18. N. Mokhoff, Computer Design, 26(4) (1987) 63.

19. N. Mokhoff, Computer Design, 26(6) (1987) 53.

20. G. A. Anderson and E. D. Jenson, Computing Surveys, 7(4) (1975) 197.

21. R. E. Dessy, Anal. Chem., 54(11) (1982) 1167A.

22. R. E. Dessy, Anal. Chem., 54(12) (1982) 1295A.

24. M. Kristo, in preparation.

25. NEC Electronics, San Mateo, CA.

26. B. H. Newcome, private communication.

27. A. J. Schubert, Ph.D. Dissertation, Michigan State University, in preparation.

28. Granville-Phillips Corp., Boulder, CO.

29. Finnigan Corp., San Jose, CA.

30. B. H. Liebowitz and J. H. Carson, Multiple Processor Systems for Real-Time Applications (Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1985).

CHAPTER TWO

CONTROL OF A TRIPLE QUADRUPOLE MASS SPECTROMETER WITH A NOVIX FORTH PROCESSOR

1. Introduction

The choice of FORTH as the language in which to write the software control system for TQMS provided an unexpected opportunity for further gains in scanning speed by using the Novix NC4000 Forth processor (1,2). The NC4000 is a 16-bit microprocessor that is optimized for the direct execution of FORTH. The version of the NC4000 used in the creation of this control system operates at a clock speed of 6 MHz, but executes one instruction per clock cycle for an overall processor speed of 6 million hardware instructions per second (MIPS). Since, in many instances, several basic FORTH instructions can be combined into one hardware instruction, the processor speed is specified as 8 million FORTH instructions per second.

The Novix Beta board development system for the NC4000 can be purchased for less than $3600. Figure 2.1 shows the price and performance of other well-known, high-powered computer systems. If it were shown in Figure 2.1, the NC4000 would be classified as a general-purpose, multi-application, uniprocessor minisupercomputer (based on its 6 MIPS performance). The expected price for such performance, based on the data in Figure 2.1, would be $50,000 to $500,000. The remarkable price/performance ratio for the Novix NC4000 allows one to create a fast, flexible

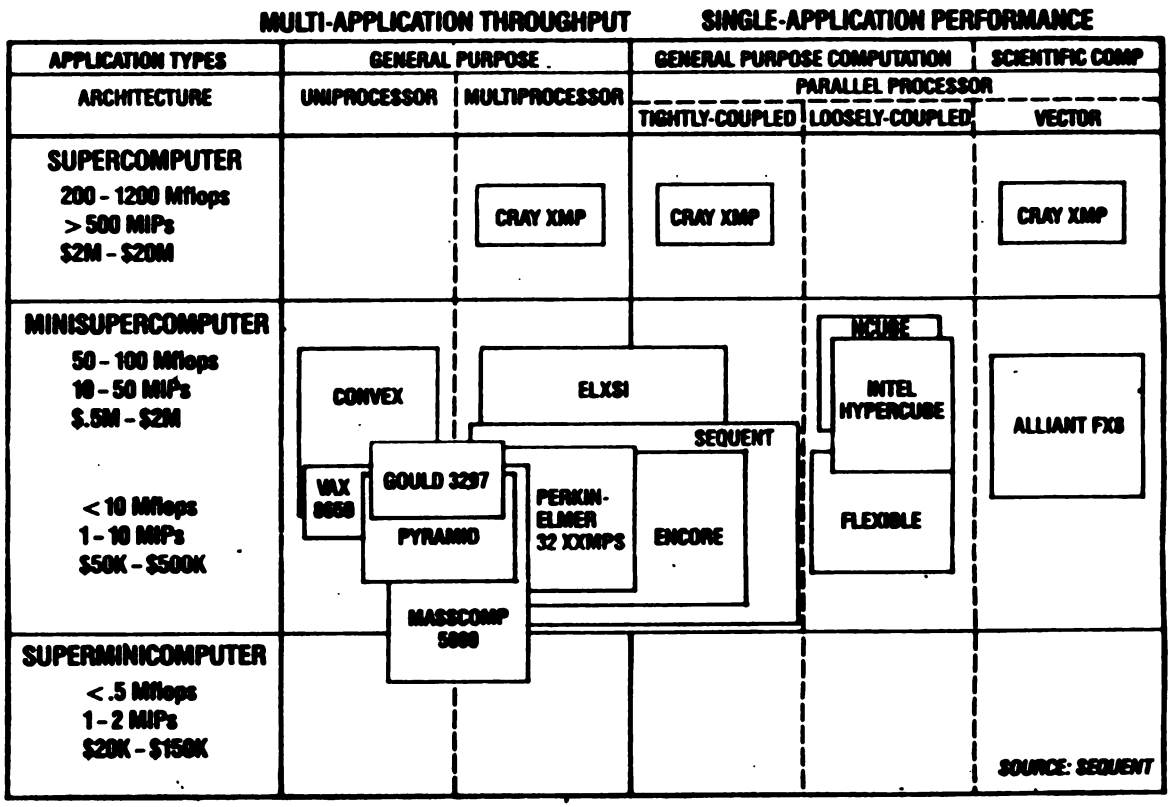


Figure 2.1 Relative price and performance characteristics of several computing systems. (Electronic News, Oct. 13, 1986)

laboratory control system for a moderate price. Furthermore, most of the systems mentioned in Figure 2.1 were not designed for instrument control, but for intensive calculations. Often their input/output functions, which comprise a large part of the task of a control processor, are quite slow.

Some applications cannot effectively use higher performance control systems; no new capabilities would be added with such systems. However, MS/MS control systems can use as fast a control system as possible for several reasons. First, the system can then have the ability to perform new experiments not presently possible because of a need for a high degree of computer control in a short span of time. For instance, real-time experimental optimization and selection is now possible with the NC4000; known experimental facts and rules together with the MS/MS information currently being acquired can be used to change conditions dynamically or select future experiments to elucidate the structure of an unknown compound in real-time. Second, a faster MS/MS control system can better characterize samples that do not last long in the ion source, such as chromatographic eluents or flash pyrolysis products. Thirdly, the less time spent making calculations, performing peak-finding and other housekeeping details, the more time is available to acquire data, given the same scan rate. Obviously, the greater fraction of time the control system spends on data acquisition, the better quality data

the mass spectrometrist will receive and the more effectively the sample will be used. The experimental benefits of control system speed are summarized in Table 2.1.

Table 2.1

Benefits of a Fast MS/MS Control System

Better characterization of transient samples (GC/MS/MS, Py/MS/MS, etc.)

More efficient utilization of the ion current

Real-time experimental optimization and selection (also depends on software)

More efficient use of an operator's time (also depends on software)

Less operator fatigue (also depends on software)

2. The Novix Approach

History of Computer Architectures

In general, large increases in processor performance also mean large increases in processor price. The development of high-level computer languages has traditionally followed the development of the sets of machine-level instructions for the particular microprocessors (or processors) on which they were to run. These sets of machine-level instructions tended towards larger sets with more complex instructions, in order to reduce program development time (each machine-level instruction accomplishes more of the overall task) and

conserve program memory. This progress forced compilers (programs to translate high-level languages, such as FORTRAN, into machine-executable code) to become "smarter," that is, more aware of the variety of machine-level instructions available and how to use them to create the most efficient code. However, compilers often tended to be simpler than the task required and produced unoptimized code, i. e., code which accomplished a task more slowly than necessary, by having unneeded machine-level instructions or by using awkward programming constructions. Recently, however, multipass compilers have become more efficient for certain processors and control systems (languages for the IBM PC (3), for instance).

Lately, the RISC (Reduced Instruction Set Computer) approach has come into vogue, especially for supercomputers. In RISC, the computer uses a smaller set of instructions which accomplish basic tasks, such as arithmetic instructions, register-to-register movement, register-to-memory movement, etc. The RISC computer is optimized for speedy execution of these instructions, since some studies have shown that a large portion of the computer's time is spent on such basic instructions. So, even though the RISC computer may accomplish the complex and infrequent instructions more slowly than the CISC computer (Complex Instruction Set Computer), it more than compensates for the loss in time by executing the simple, but frequent, instructions very quickly. Furthermore, RISC allows the

compilers themselves to become simpler and more efficient, since they only have to handle a small number of instructions. Nevertheless, hardware development preceded software development for the RISC computers as well (4,5).

There are, of course, other approaches to faster computers. Design of bit-slice computers (6) for special applications is one approach. In fact, Metaforth Computers of Great Britain (7) have designed a FORTH processor based upon bit-slice technology. Bit-slice has the advantage of tailor-making the processor to a given task, giving it great speed. However, designing a bit-slice computer for a specific task will obviously take longer than buying a given microprocessor off the shelf and, eventually, any bit-slice processor will be limited by the speed of discrete transistor-transistor logic (TTL) (or whatever the chosen technology) components. Processors with large scale integration (LSI), on the other hand, tend to execute functions more quickly than the same function emulated in discrete integrated circuits.

NC4000 Architecture

The Novix NC4000 embodies the architecture of an already existing high-level language, FORTH, in an integrated circuit. In other words, the NC4000 runs FORTH, rather than a typical assembly language, as its native code. The NC4000 has been called "a stack machine," since FORTH itself is a stack-oriented language. A stack is an area of

memory used for temporary storage of various values. The data on the stack are normally placed and removed in a last-in first-out (LIFO) manner. The NC4000 is also optimized for subroutine calls (one every clock cycle) and subroutine-threaded code is its normal mode of operation. This means that new programs for the NC4000 are compiled as a series of addresses for the execution of its component programs. FORTH words (programs) consist of concatenated words which have been previously defined or are part of the basic kernel of instructions. To execute FORTH on other processors, the subroutine-threaded code is not inherent and must be created in the software by the FORTH compiler and interpreter.

The NC4000 has simultaneous access to both its data and return stacks through separate 16-bit data buses and 8-bit address buses, apart from the 16-bit address and data bus for main memory (Figure 2.2). Most FORTH instructions implicitly operate on items on the data stack, while the return stack is used to direct the return from subroutine calls. Thus, optimization of the access to these two stacks, which normally exist in main memory on the typical microprocessor, is essential to the NC4000's performance. Furthermore, the top two data stack elements are actually registers on the NC4000, making operations on those two elements especially fast.

Finally, the bit patterns within each instruction provide the actual control signals to the various components of the microprocessor, e. g., the arithmetic logical unit

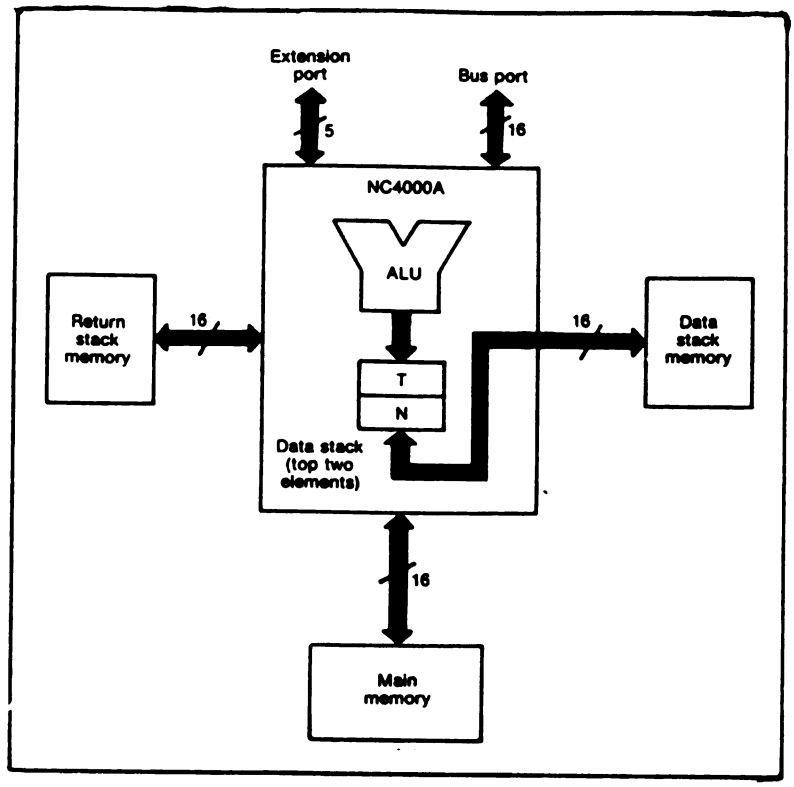


Figure 2.2 Block diagram of the NC4000 showing the separate data paths to main memory, its data stack, and its return stack (2).

(ALU) or the address multiplexer (Figure 2.3). The elimination of internal microcode, which is standard on conventional microprocessors, further increases the speed of the NC4000.

3. System Description

Hardware

The Novix BetaBoard Forth computer is an evaluation board, which includes the NC4000 microprocessor, 32 Kwords (1 word = 16 bits) of fast memory (35 nS), data and return stack memory (256 words each), two RS-232 serial ports, a SCSI (Small Computer Systems Interface) port, and a timer/counter for performance measurements. The system further provides two sets of connectors for stackable interface modules which allow access to vital processor signals (address and data lines, system clock, etc.). Thus, the Beta board only needs a moderately complex interface board in order to control the EL 400/3 triple quadrupole mass spectrometer.

The Novix Beta board comes equipped in an IBM PC (3) support configuration, in which a serial link to the PC allows the PC to serve as a terminal and file server. The first step in preparing the Beta board to control a triple quadrupole mass spectrometer was to create a development system by generating the appropriate code by target compilation and programming a set of ROMs with this code in

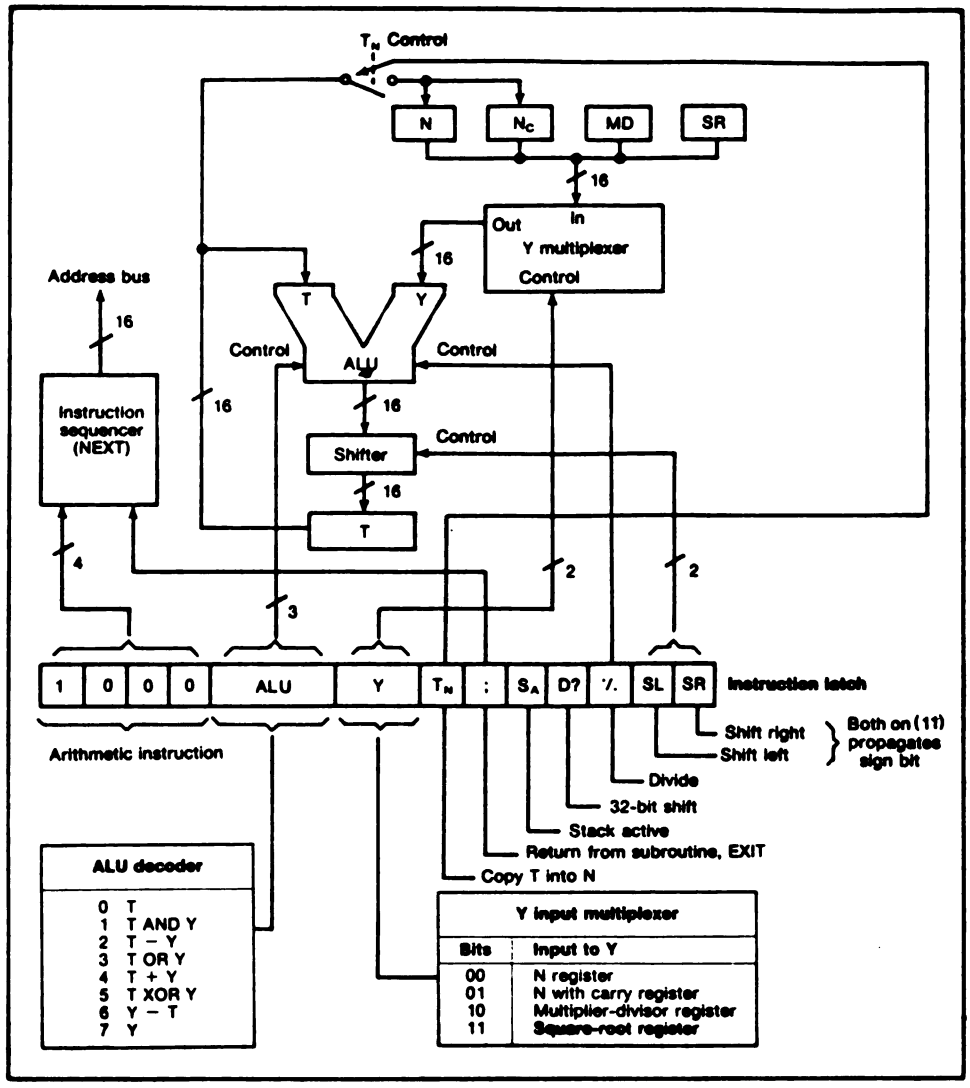


Figure 2.3 Block diagram of the NC4000 which shows the direct control that the latched instruction exercises over the ALU and registers (2).

order to create the stand-alone configuration. Then, equipped with a DTC 520 DB (8) disk controller, a hard disk drive, and a floppy disk drive, all controlled through the SCSI port, the Beta board was ready for hardware and software development.

Several functions had to be provided on the Novix-triple quadrupole mass spectrometer interface board in order to mimic the existing Extrel 400/3 control system. First, an interface was needed to the Extrel L-Bus, a peripheral bus which contains all of the digital-to-analog converters (12 and 16-bit), an analog-to-digital converter with 8 channel multiplexer, and several parallel ports. The NC4000 can now control all of the triple quadrupole mass spectrometer's physical devices through the analog and digital outputs on the L-Bus by using the L-Bus interface on the Novix TQMS Control Board. The schematic for the L-Bus interface is shown in Figure 2.4.

The mass spectrometer interface board needed an interface to the intelligent data acquisition circuit designed by Dr. Bruce Newcome (9) in order to acquire intensity data efficiently. The data acquisition circuit allows parallel operation with the control computer through use of a status byte and/or an "acquisition-done" interrupt. The data acquisition interface, whose schematic is shown in Figure 2.5, now allows the NC4000 to control this data acquisition circuit.

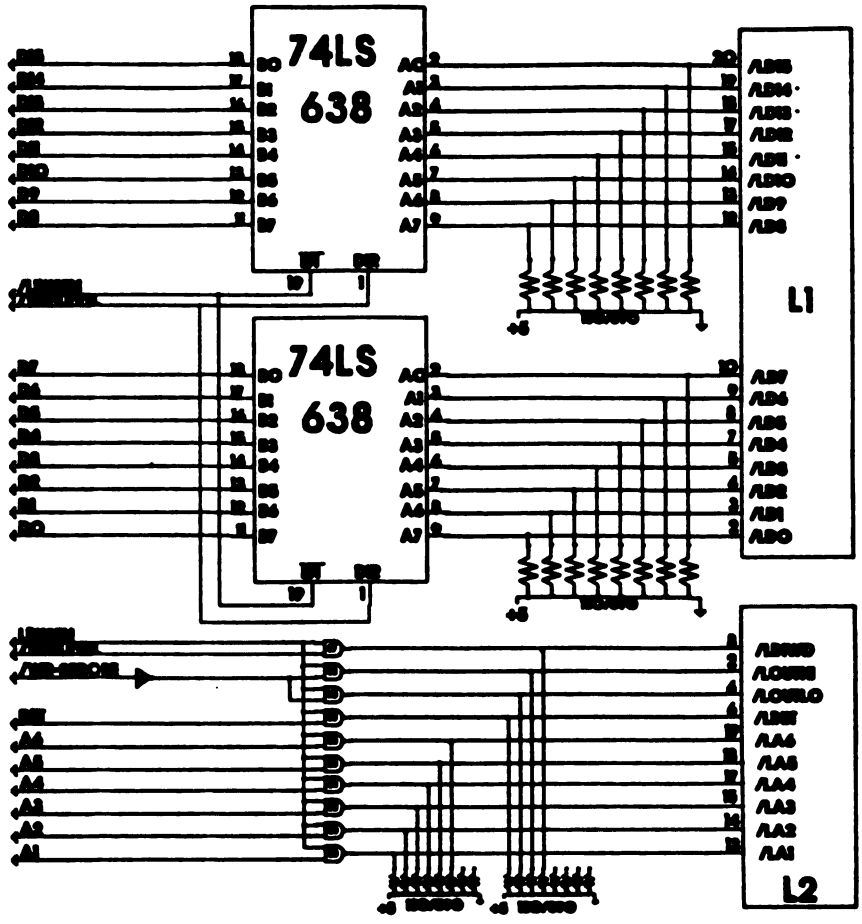


Figure 2.4 Schematic of the L-Bus Interface.

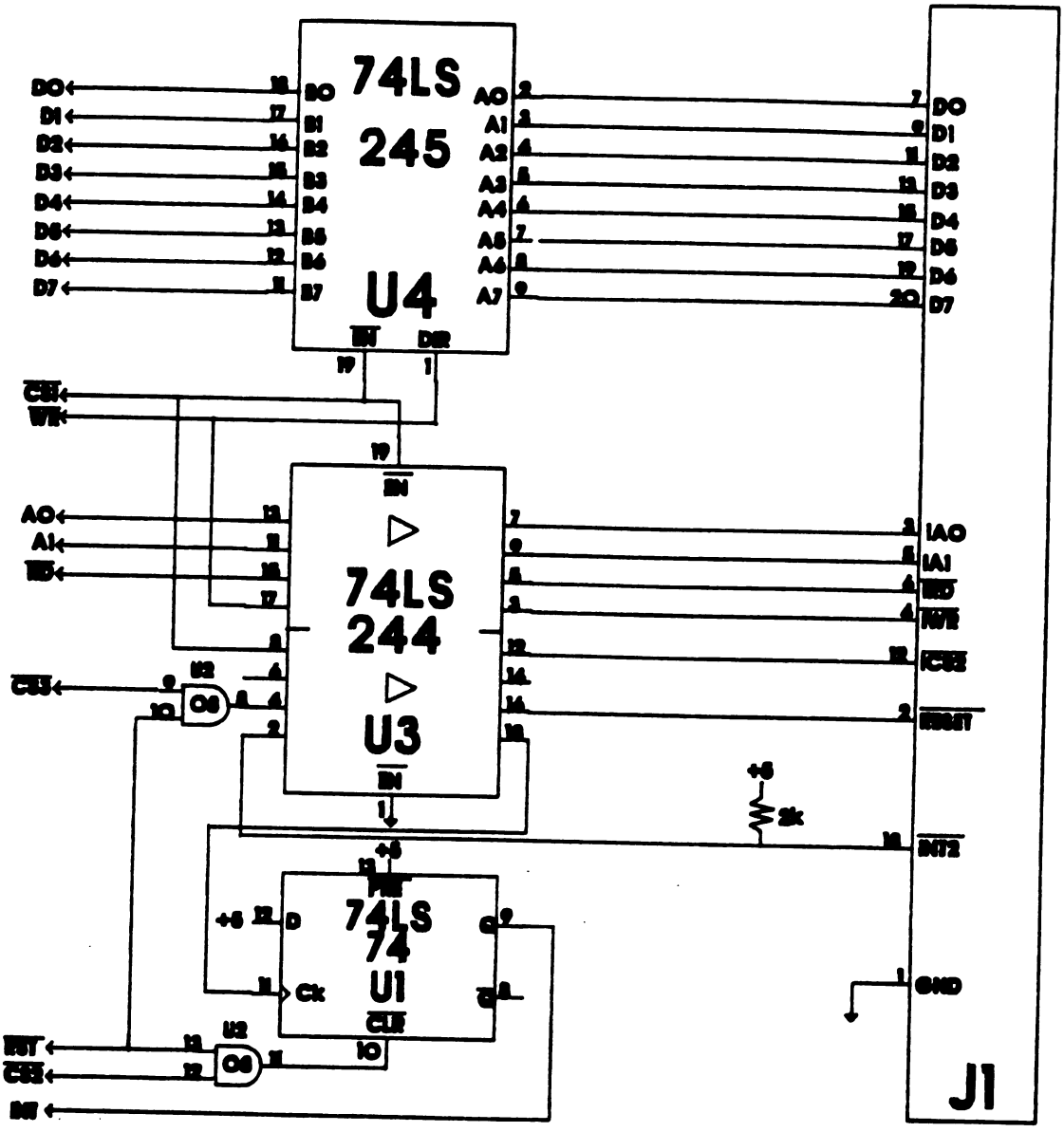


Figure 2.5 Schematic of the interface to the intelligent data acquisition circuit.

The NC4000 is provided with an interface, shown in Figure 2.6, to the hardware peak-finding circuit, which is more fully described in Chapter 4. Basically, this circuit allows rapid, parallel processing of intensity data to determine peak positions and peak heights. This circuit can operate in parallel with NC4000, like the data acquisition circuit, through its status byte and interrupt feature. This parallelism allows the NC4000 to calculate the next ion path value while the peak-finding circuit is processing the current datum.

The NC4000 has access to an AMU-timer circuit in the Mass Spectrometer Interface. This AMU-timer allows the computer to synchronize each point in the scan with the user-selected scan rate. The computer cannot acquire the next point until the AMU-timer interrupt has been set. This feature is important not only for keeping the scan rate constant throughout a scan, but also to correlate scan number with time in a sequence of scans. The schematic for the AMU-timer is shown in Figure 2.7.

The NC4000 triple quadrupole mass spectrometer interface board has an interface for "softknobs," optical rotary encoders which can be configured in software to control any digitally controlled device with variable resolution. Softknobs are used for manual variation of digitally controlled devices, as would normally be performed to tune the instrument. The schematic for the softknobs interface is shown in Figure 2.8.

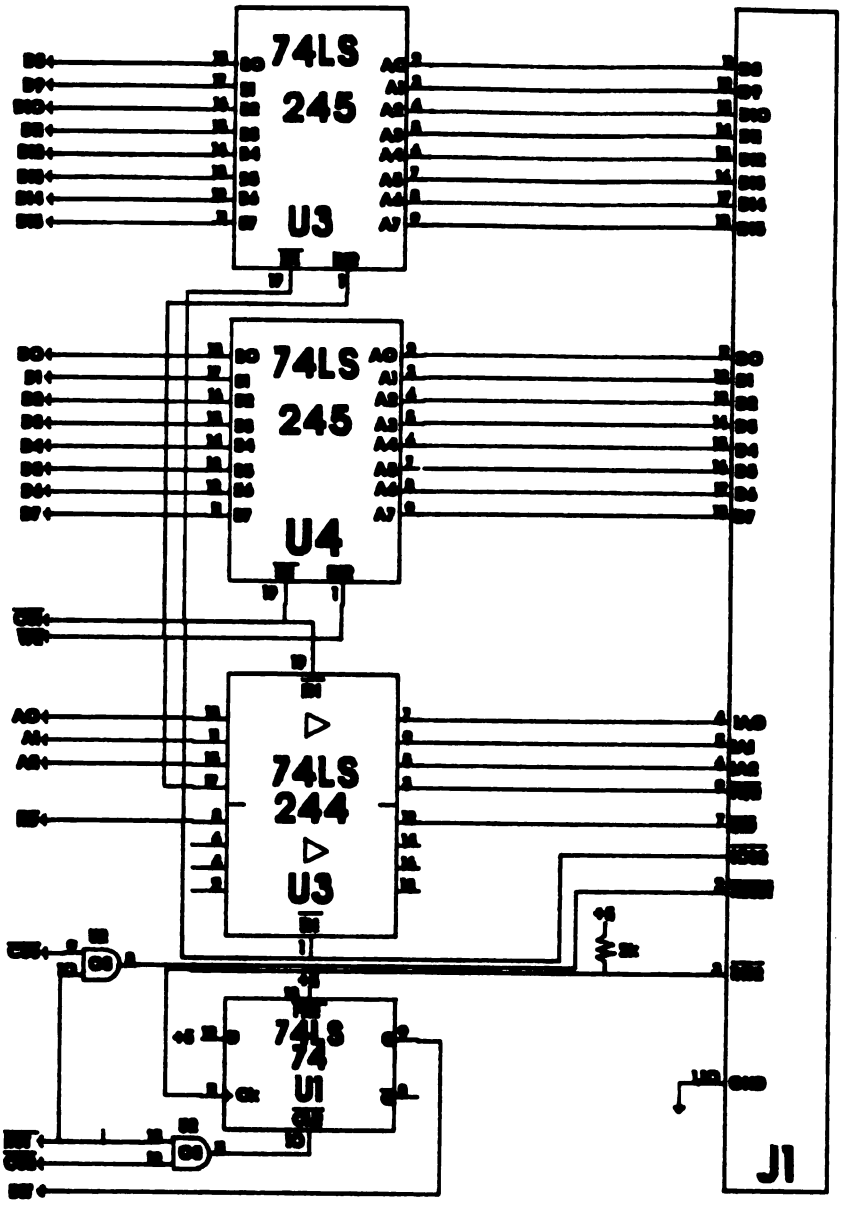


Figure 2.6 Schematic of the interface to the hardware peak-finder circuit described in Chapter 4.

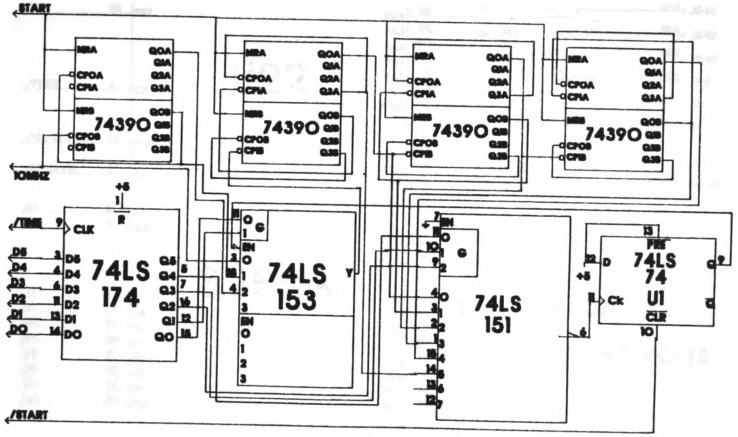


Figure 2.7 Schematic of the rate synchronization circuit.

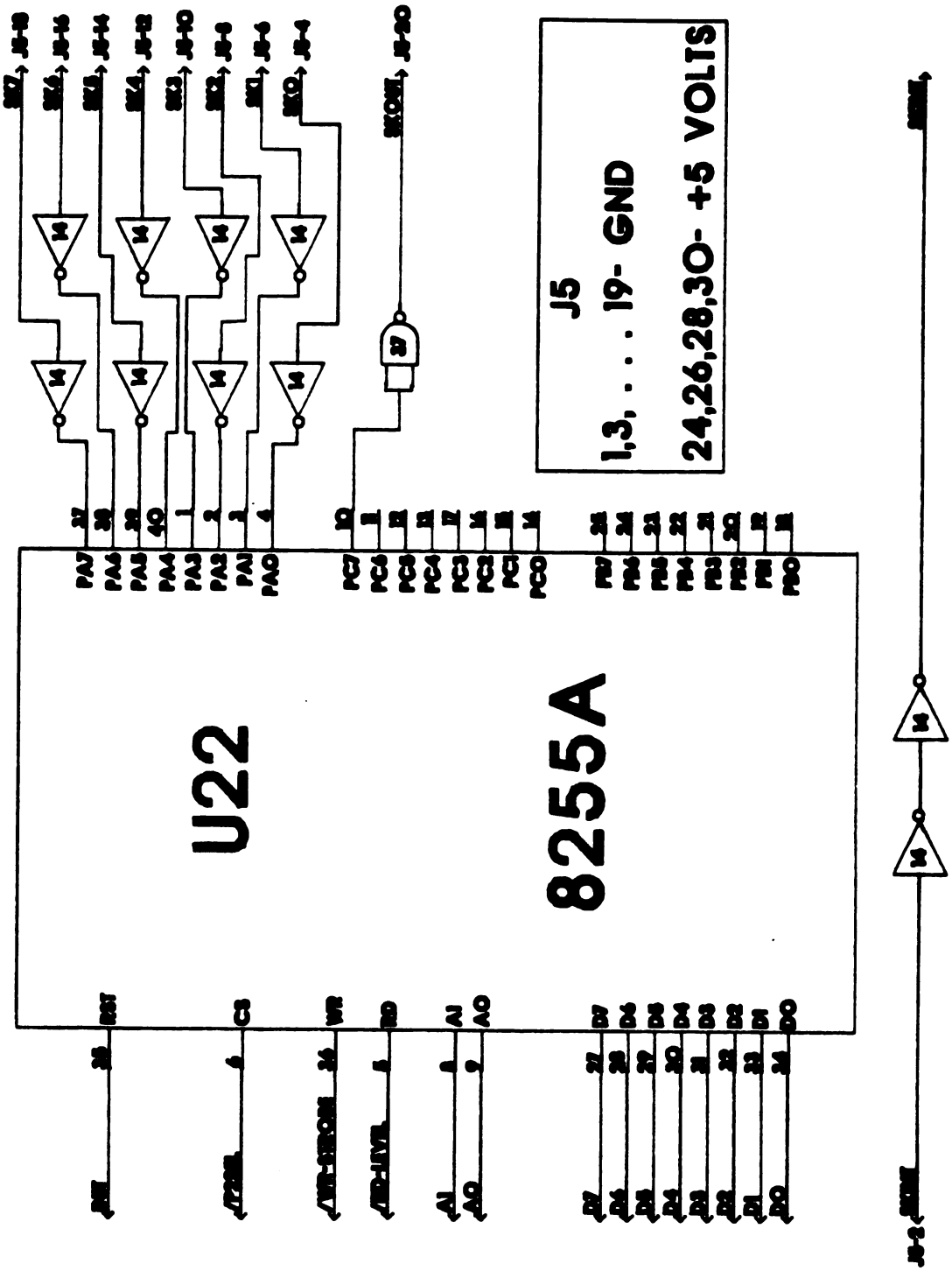


Figure 2.8 Schematic of the softknobs interface.

The mass spectrometer interface has a parallel port, whose schematic is shown in Figure 2.9, dedicated to transferring data to the host PDP 11-23 (10) minicomputer for further processing and eventual archival storage. A MM52167 real-time clock (11) is provided, which has a battery back-up. This allows the correct time to be stored with all experimental data without resetting the clock after every power-up. Finally, the AMD 9513 (12) pulse-counter circuit, whose schematic is shown in Figure 2.10, is provided for ion counting. Thus, the NC4000 control system has the capability for dual-mode detection, which is fully described in Chapter 5. The 9513 counter/timer further provides the capability for higher resolution timing than the AMU-timer provides.

The most important part of the Novix TQMS Control Board, though, is the actual interface between the Novix NC4000 and all of the separate interfaces described above. The essential problem is to enable the fast timing of the NC4000 to operate compatibly with the slower timing of the peripheral devices. For instance, the NC4000 is provided with fast (35 nS) memory, because the NC4000 places a memory address on the bus during the high portion of the clock cycle and expects to read or write the corresponding datum on the next rising edge. Most of the peripheral devices on the interface board cannot meet this timing requirement. The Novix-mass spectrometer interface (Figure 2.11) seeks to minimize the generation of control signals through software

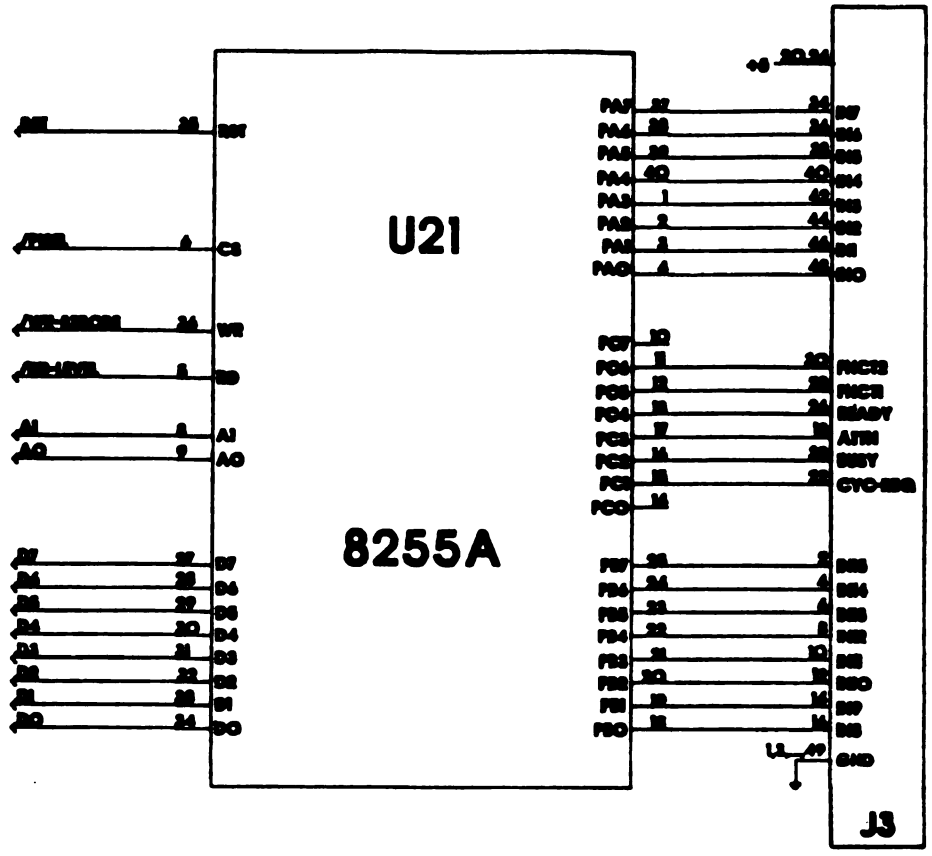


Figure 2.9 Schematic of the parallel port for uploading data to a host minicomputer.

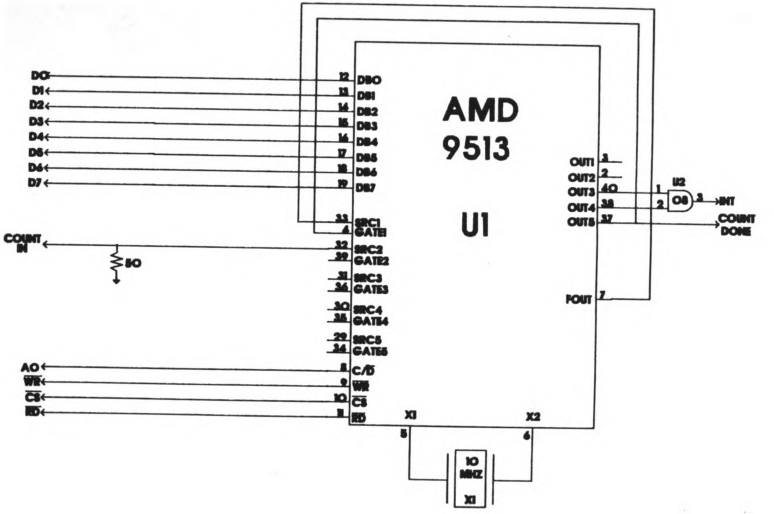
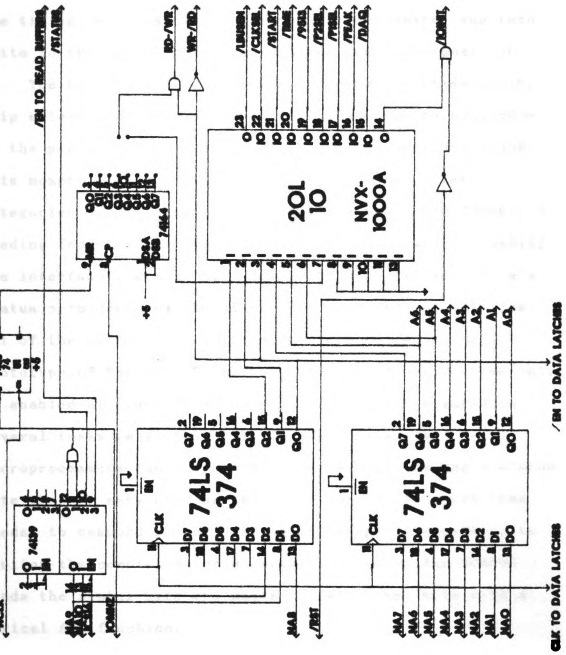


Figure 2.10 Schematic of the pulse-counting circuit.

Figure 2.11 Schematic of the interface between the NC4000 and the rest of the mass spectrometer control circuits.


/EN TO READ BUFFER
/STATUS
RD-/WR
WR-/RD
/LBUSEN
/CLKSEL
/START
/TIME
/DAQ
2OL
10
NVX-
1000A
74164
+5
74LS
374
74LS
374
74139
A6
A5
A4
A3
A2
A1
A0
/EN TO DATA LATCHES
CLK TO DATA LATCHES
NA7
NA6
NA5
NA4
NA3
NA2
NA1
NA0
NA8
/RST

manipulation. For instance, the counter and serial ports provided with the Novix BetaBoard are addressed through a series of latches, in which you write to an address to send a control signal high, wait an amount of time appropriate to the timing requirements of the peripheral device, and then write to the same address to send the control signal low.

The Novix-mass spectrometer interface utilizes JCS16, a chip select generated on the Novix BetaBoard and available on the peripheral bus, which selects addresses C000-C799H. This memory selection is further divided into three categories: 1) writing to the interface bus (C000-C399H), 2) reading from the interface bus (C600-C799H), and 3) reading the interface's status byte (C400-C599H). The interface's status byte contains the status bit or interrupt bit from all of the appropriate interfaces. One error in the prototype of the NC4000 is that interrupt functions can only be enabled at times when the processor is not engaged in several tasks (multitasking) In many instances, microprocessors can respond more quickly by reading a status byte than by servicing an interrupt due to the extra time needed to respond properly to an interrupt. Therefore, to monitor the completion of a certain activity, the NC4000 reads the status byte and masks off all other bits with a logical AND function.

In order to write to a peripheral device, the NC4000 accesses an address in the range of C000-C399H, causing the corresponding data bits 0-15 and address bits 0-10 to be

latched on the Novix TQMS Control Board. In this way, the appropriate data and address are temporarily stored for the peripheral devices to process at their own rate. Selection signals (chip selects) for the various interfaces are decoded by a PAL20L10 (13), the specifications for which are shown in Appendix C. The chip selects and the write signal (/WR) itself are only valid for a period of time specified by enable signals. These enable signals (active low) are selected by a jumper from any one of eight pulse widths from a 74LS164 shift register. The length of the pulse width determines the time for which the chip selects or the write strobe are valid. The falling edge of the enable pulse starts one clock cycle after writing to the mass spectrometer interface and lasts from 100-800 nS (typically 500 nS for a chip select; 400 nS for /WR). The enable signal for /WR should be selected as at least 100 nS less than the enable for the chip selects, since many chips latch the data on the rising edge of the write strobe. Obviously, the NC4000 cannot write to the interface within 600 nS of the last write or conflicting signals will be generated. So far, this has not been a problem.

The NC4000 reads from the mass spectrometer interface board in two steps. First, it writes the selected address on the interface board, except that bit 8 (RD-/WR) is high. This translates to the original write address plus 100H. Since bit 8 is high, the PAL recognizes that a read operation is in progress and holds the given chip select low

indefinitely. Inverting the value of bit 8 provides a read signal (/RD). The selected peripheral device responds to its chip select and /RD and places the appropriate datum on the interface data bus. Second, the NC4000 reads from address C600H, which transfers the contents of the interface data bus onto the Novix's data bus.

The Novix-mass spectrometer interface circuit keeps the Novix convention of word addresses. The NC4000 can write bytes to a peripheral's address with no problem, because the upper byte will simply be ignored. However, when reading a byte from a peripheral address, the upper byte must be set to zero to read the correct value.

Software

The software for the Novix control system is essentially that which Dr. Carl Myerholtz (14,15) wrote for the Extrel 400/3 control systems except for the following four essential categories of modifications.

First, all of the existing control system software which was originally written in 8086/8088 assembly language had to be rewritten in high-level FORTH. This was fairly easy, certainly much more easy than writing high-level FORTH software in assembly language.

Second, the original control system software had been written in polyFORTH I, whereas the NC4000 FORTH kernel was consistent with polyFORTH II. Thus, all inconsistencies between polyFORTH I and II had to be resolved. Some

differences were only in nomenclature; for instance, the word END was renamed UNTIL to comply more fully with FORTH-79 standards. Other changes were more subtle. Software which took advantage of certain known structures in polyFORTH I (16) had to be rewritten when those structures were changed in polyFORTH II (16).

Third, the NC4000 is a full 16-bit microprocessor, with no provisions for byte addressing in hardware. Therefore, byte operations take place in software at a considerable time disadvantage. Byte operations take about twice as long as word operations on the NC4000. Furthermore, only the first 32 Kbytes of memory can be addressed as bytes. Each word of memory has two addresses: a word address and two byte addresses (twice the word address and twice the word address plus one). The words CELL and BYTE convert from one address form to the other. Therefore, all byte operations which could easily be word operations were rewritten as such, costing the user one unused pseudo-byte of memory. Other operations, such as string operations, had to be rewritten to convert word addresses to byte addresses and use existing character operations on the NC4000.

Fourth, new routines were written for the data acquisition circuit and the hardware peak-finder, which did not exist on the Extrel 400/3 control system. Routines were also written to control the AMD 9513 pulse-counting circuit.

All of the revised code described above resides on the Novix control system hard disk, as well as on floppy disk back-up copies and hardcopy output.

4. Results

The Novix NC4000 Forth computer is now capable of fully controlling a triple quadrupole mass spectrometer. Tables 2.2, 2.3, and 2.4 show the scanning speed and the amount of averaging available for each speed with the old Multibus I control system (Table 2.2), the multimicroprocessor system described in Chapter 1 (Table 2.3), and the new Novix control system (Table 2.4) without the hardware peak-finder. The Novix control system not only operates at faster scan speeds than the Multibus control system but also provides 64 times the averaging for equivalent scan speeds. The new data acquisition circuit is responsible for some of this increase over the Multibus system. This data acquisition circuit allows the user to select a number of averages which are factors of two (1, 2, 4, 8, etc.). Although this leads to a significant increase in the amount of averaging for the Novix system over the Multibus system, it is comparable to the multimicroprocessor system for most scan speeds. The Novix control system would require twice as much time for data acquisition as the multimicroprocessor system in order to perform more averaging. This condition was met only for the 50 and 100 amu/S scan rates.

Table 2.2
Multibus-I Scanning Functions

Rate	Pts/S	Amu/S	Time/Pt	# Averages/Pt
2	2500	250	400 uS	1
3	1000	100	1 mS	4
4	500	50	2	16
5	250	25	4	32
6	100	10	10	128
7	50	5	20	256
8	25	2.5	40	512
9	10	1	100	1024
10	5	0.5	200	2048
11	2.5	0.25	400	4096
12	1	0.10	1 S	4096
13	0.5	0.05	2	4096
14	0.25	0.025	4	4096

Table 2.3
Multimicroprocessor Scanning Functions

Rate	Pts/S	Amu/S	Time/Pt	# Averages/Pt
0	10000	1000	100 uS	16
1	5000	500	200	32
2	2500	250	400	64
3	1000	100	1 mS	128
4	500	50	2	256
5	250	25	4	1024
6	100	10	10	2048
7	50	5	20	4096
8	25	2.5	40	4096
9	10	1	100	4096
10	5	0.5	200	4096
11	2.5	0.25	400	4096
12	1	0.10	1 S	4096

Table 2.4
Novix Control System Scanning Functions

Rate	Pts/S	Amu/S	Time/Pt	# Averages/Pt
2	20000	2000	50 uS	1
3	10000	1000	100	16
4	5000	500	200	32
5	2500	250	400	64
6	1000	100	1 mS	256
7	500	50	2	512
8	250	25	4	1024
9	100	10	10	2048
10	50	5	20	4096
11	25	2.5	40	4096
12	10	1	100	4096
13	5	0.5	200	4096
14	2.5	0.25	400	4096

The ability of the control system to scan the triple quadrupole mass spectrometer as fast as the rates listed in Table 2.4 does not suggest the quality of the data at those scan rates. Several problems can occur at fast scan rates. First, the quadrupole controllers may not be able to change the voltages on the quadrupole rods as quickly as the control system changes the mass command input. Second, the transit time of the ions through the quadrupole fields will limit scan speeds. At lower velocities (low kinetic energies) the ions will more slowly traverse a given quadrupole, during which time the electric fields will be changing. This may possibly lead to reduced transmission and/or degradation of peak shape. Of course, at higher ion velocities, the velocity effects will be reduced but each ion will spend less time in the quadrupole fields. The mass spectral resolution in a quadrupole increases with the square of the number of RF frequency cycles that the ions

experience in the quadrupole. Thus, higher velocities will lead to decreased resolution of the peaks in the mass spectrum. Third, the bandpass of the amplifiers in the system (the preamplifier and the voltage amplifiers of the multiamp circuit) may limit scan speeds if it is insufficient to avoid reducing peak intensity and degrading peak profiles. This effect can be corrected by using higher bandpass amplifiers; however, they generally provide lower gains and pass noise of higher frequencies than lower bandpass amplifiers. Therefore, after determination of the scan rates possible with the Novix control system, it was important to verify the characteristics of the spectra obtained with each scan rate.

Figures 2.12 through 2.15 show the peak profiles and intensities available with the present multiamp system for scan rates 2 through 5 respectively. Each figure shows the peak profiles for ions of different mass-to-charge ratios (m/z 28 from nitrogen (N_2^+) and m/z 69, 131, and 331 from perfluorokerosene (PFK)) with ion energies of 5 eV in the scanning quadrupole (quad 1). Ion energies of 5-15 eV are common for the first quadrupole in a triple quadrupole mass spectrometer operating at 20 eV collision energy. The data for each scan are unaveraged, thus, any change in the peak profiles are due to scan speed, not increased averaging at the lower scan rates. Several features are clearly evident. First, the peak profiles taken at faster scan rates show lower resolution (increased peak widths) compared to mass

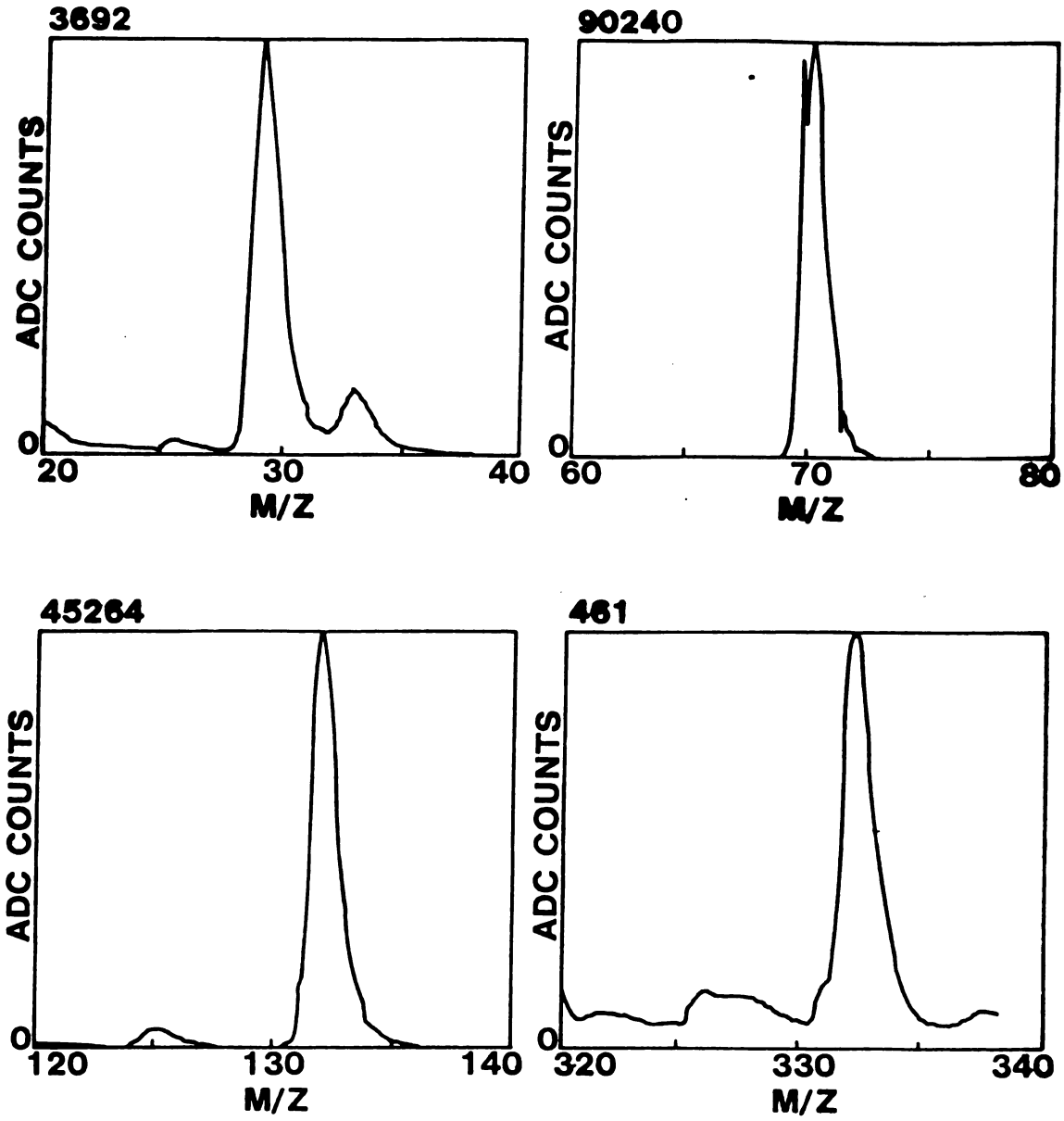


Figure 2.12 Unaveraged mass sweeps of m/z 28 (N_2^+) and m/z 69, 131, and 331 from PFK taken at 2000 AMU/S with the multiamp circuit.

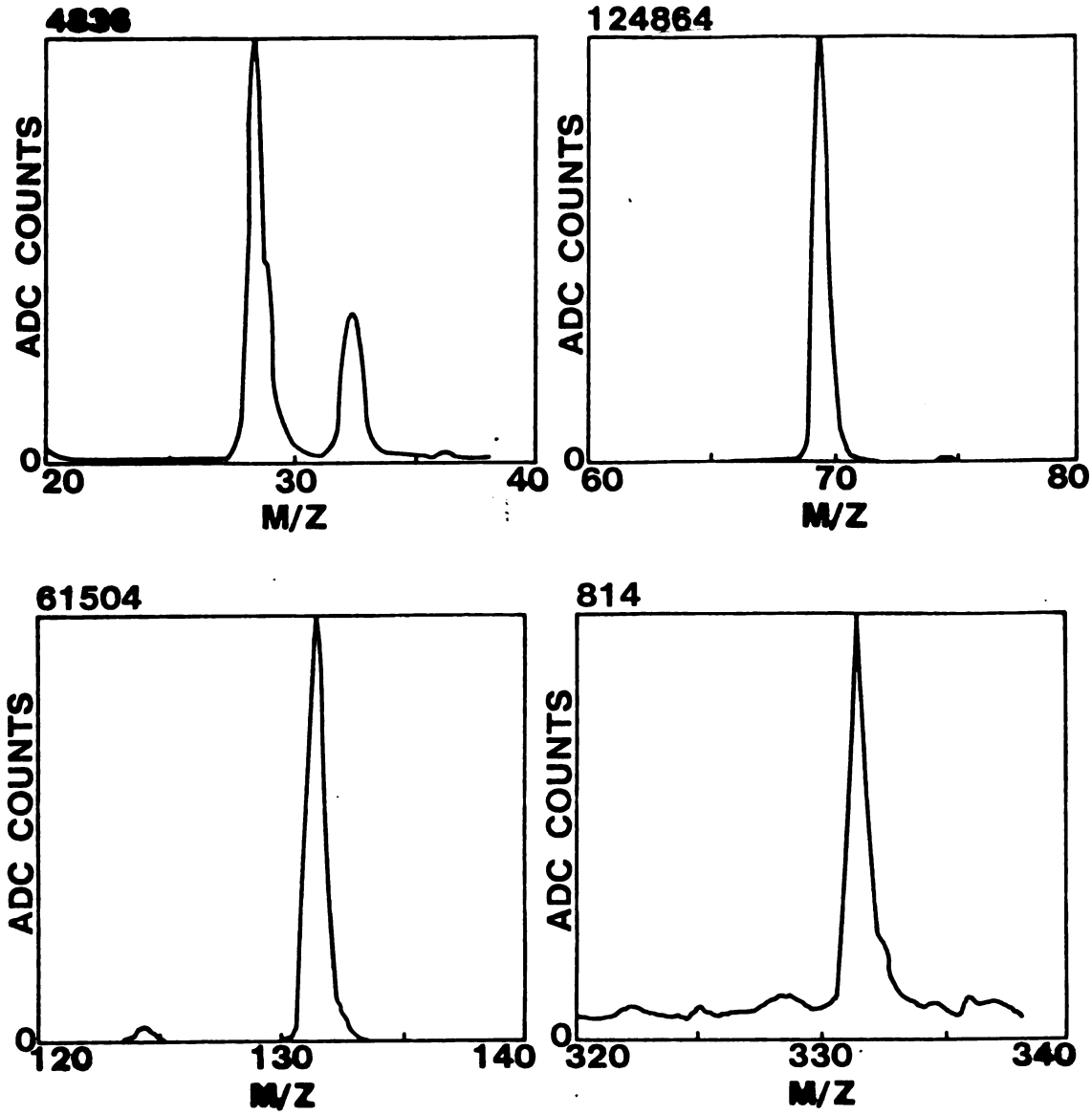


Figure 2.13 Unaveraged mass sweeps of m/z 28 (N_2^+) and m/z 69, 131, and 331 from PFK taken at 1000 AMU/S with the multiamp circuit.

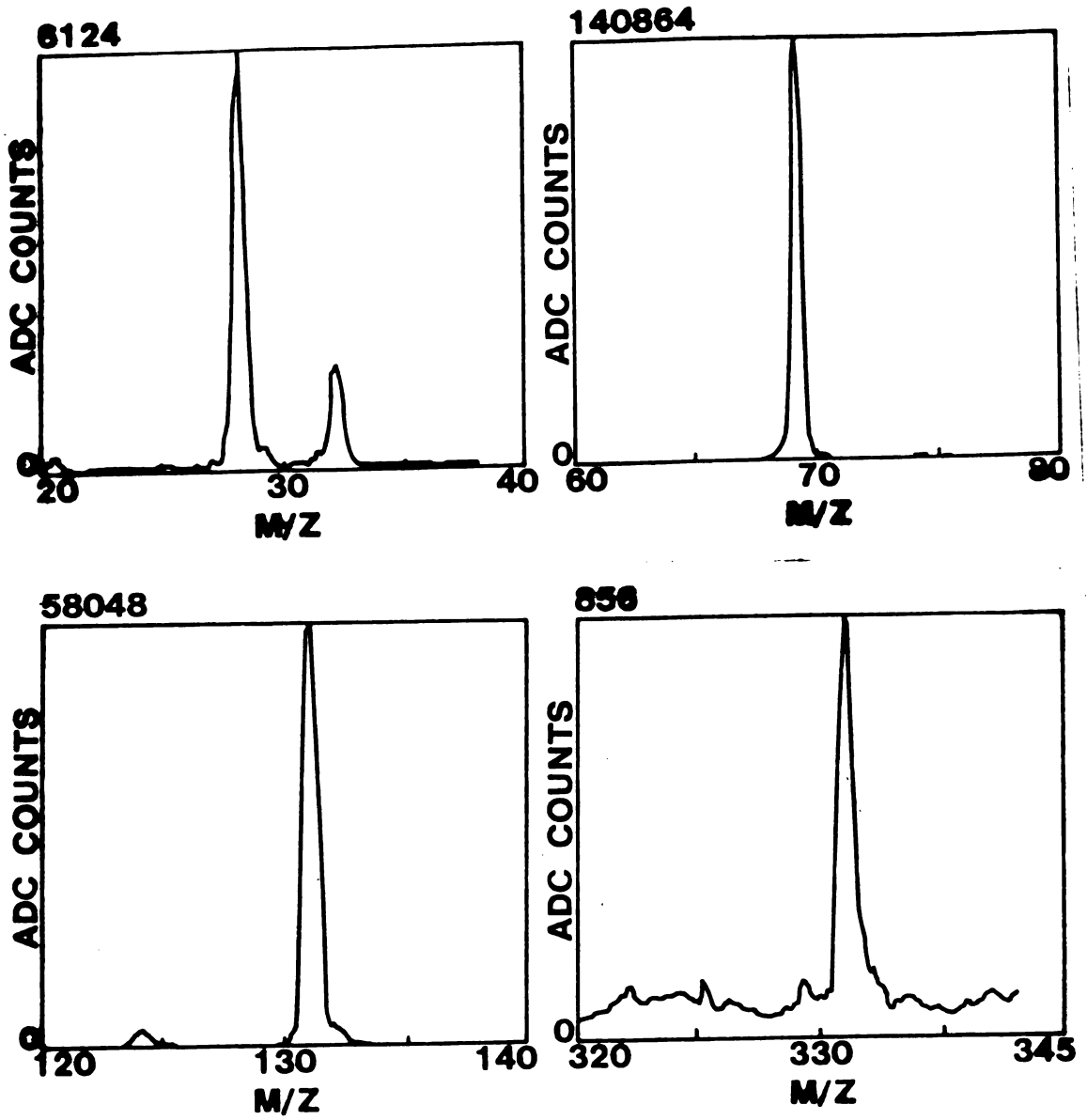


Figure 2.14 Unaveraged mass sweeps of m/z 28 (N_2^+) and m/z 69, 131, and 331 from PFK taken at 500 AMU/S with the multiamp circuit.

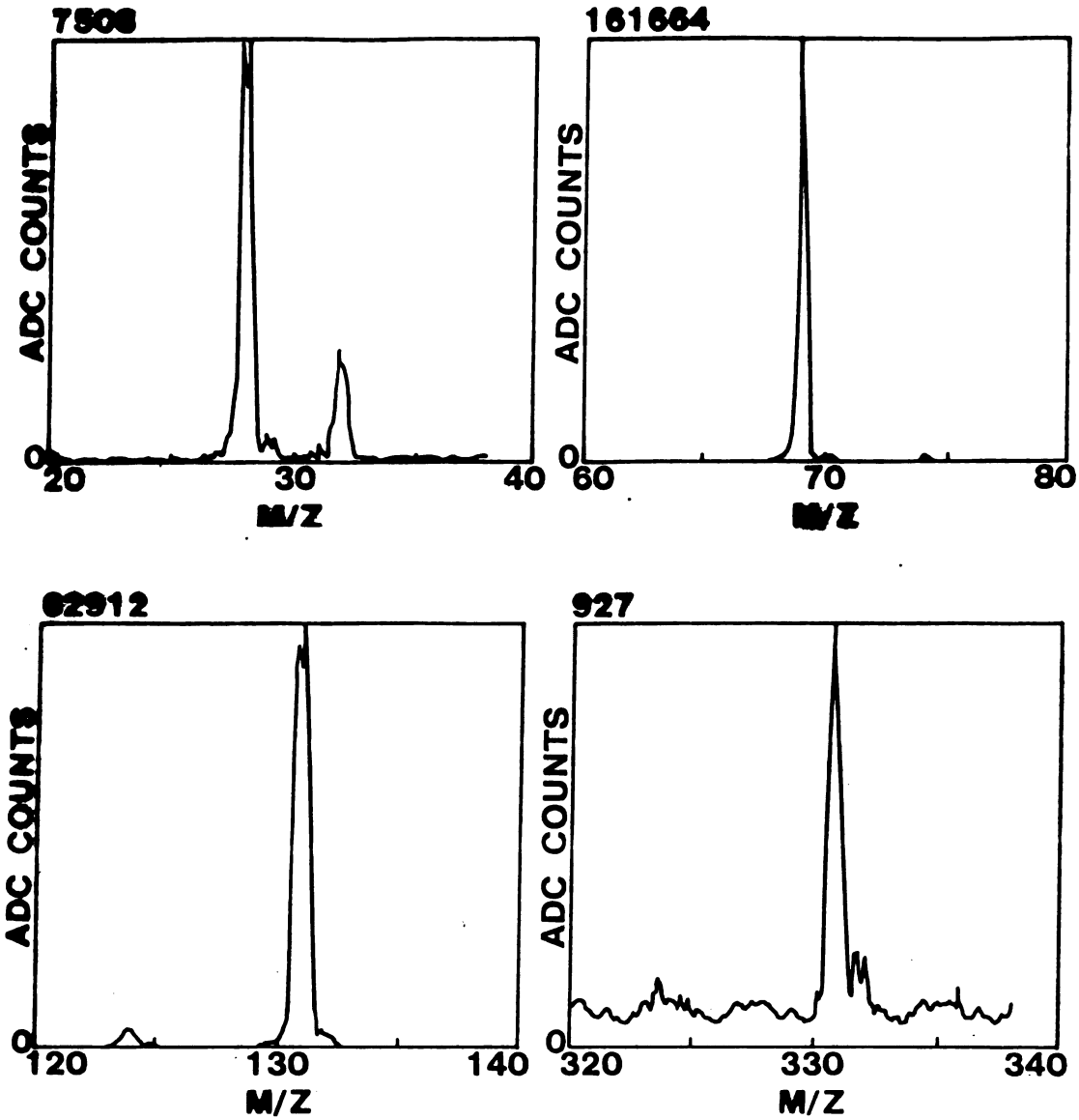


Figure 2.15 Unaveraged mass sweeps of m/z 28 (N_2^+) and m/z 69, 131, and 331 from PFK taken at 250 AMU/S with the multiamp circuit.

profiles taken at slower scan rates. Figure 2.16 shows the relationship between resolution and scan speed for each ion. Second, peak profiles taken at faster scan rates show slightly decreased ion intensities than those taken at slower scan rates. Figure 2.17 shows the relationship between ion intensity versus scan rate for each ion. Third, the peak maxima appear at progressively greater mass-to-charge ratios as the scan speed is increased. Fourth, the data for m/z 331, which is a lower intensity peak, clearly show that faster scan rates better reject low frequency synchronous noise than slower scan rates.

The decreased resolution and intensity of the mass spectral peaks, as well as the peak-shift described above, could be caused by either insufficient amplifier bandpass or by the effects of ion transit times. However, reducing the ion transit time by increasing the ion energy did not cure the peak-shift, but merely decreased the resolution further, as expected from the decreased number of RF cycles experienced by the ions. Increasing the bandwidth of the amplifier, first by eliminating the multiamp circuit (bandwidth 1 kHz) and then by replacing the present preamplifier (bandwidth approximately 3 kHz) with a wideband preamplifier (nominal bandwidth 100 MHz), established that limited bandwidth causes both the decreased resolution and much of the shifting of peak positions.

Figure 2.18 shows data obtained with the wideband amplification system at scan rate 2 (2000 amu/S), which can

Figure 2.16 Plot of the resolution of each peak versus scan speed for m/z 28 (N_2^+) and m/z 69, 131, and 331 from PFK.

Figure 2.16

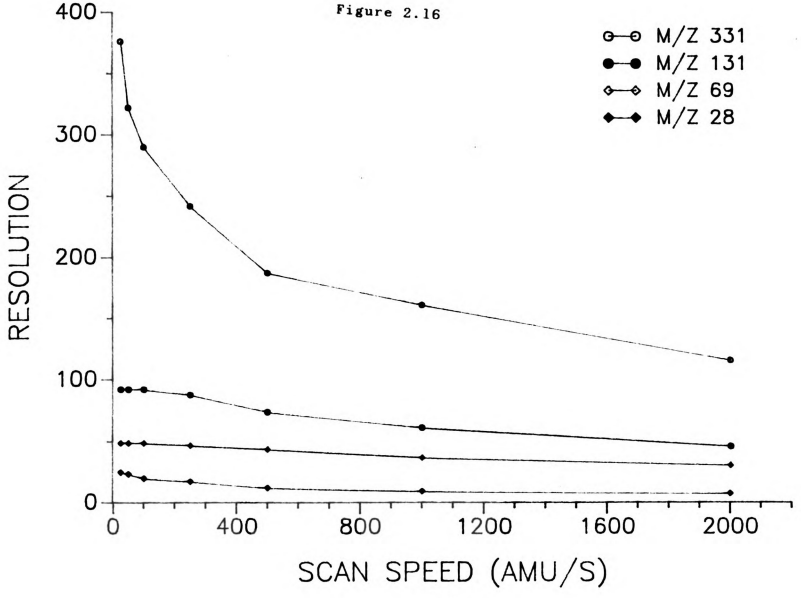

M/Z 331
M/Z 131
M/Z 69
M/Z 28
RESOLUTION
400
300
200
100
0
0
400
800
1200
1600
2000
SCAN SPEED (AMU/S)

Figure 2.17 Plot of the intensity of each peak versus scan speed for m/z 28 (N_2^+) and m/z 69, 131, and 331 from PFK.

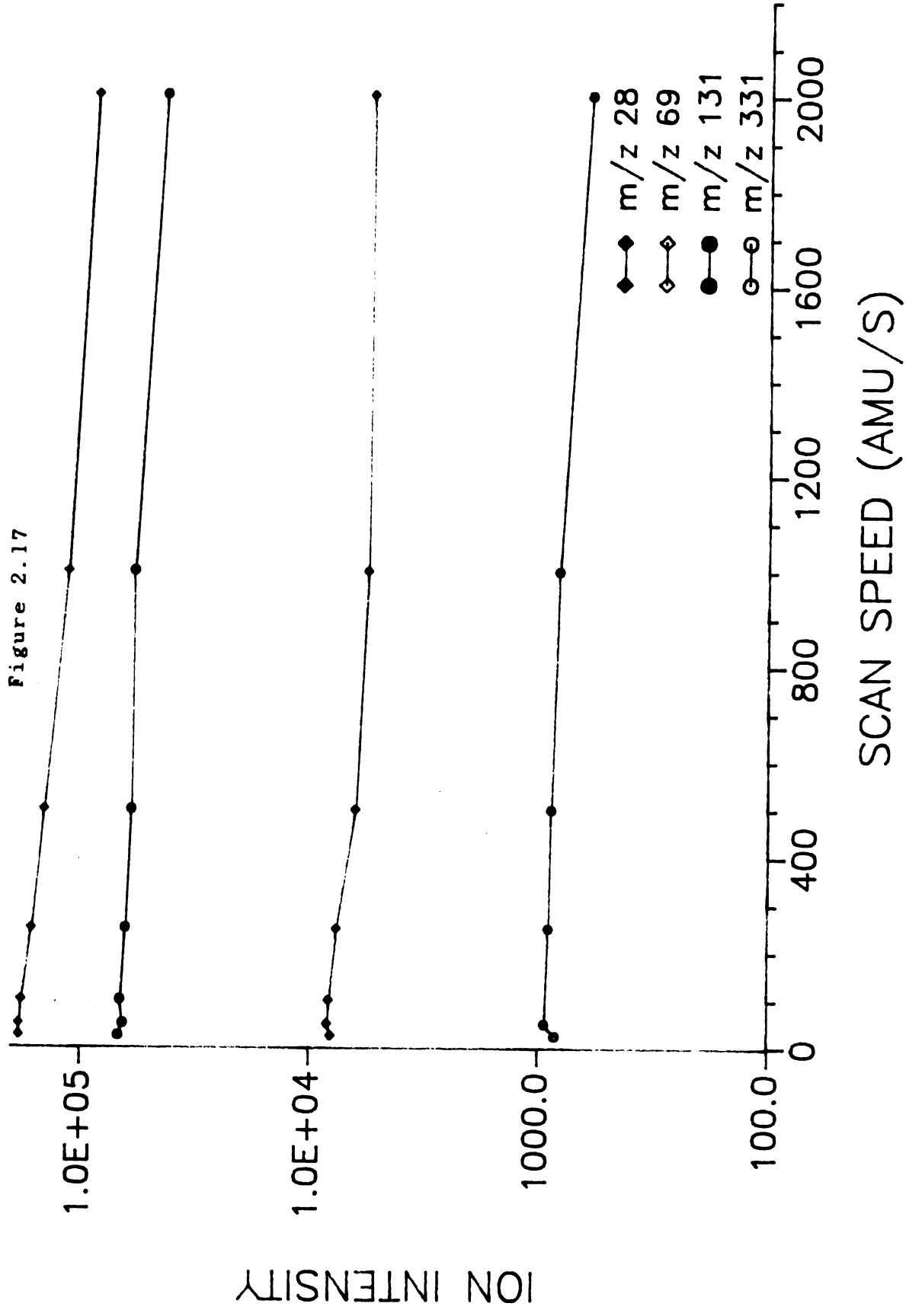

ION INTENSITY
1.0E+05
1.0E+04
1000.0
100.0
0
400
800
1200
1600
2000
SCAN SPEED (AMU/S)
m/z 28
m/z 69
m/z 131
m/z 331

Figure 2.17

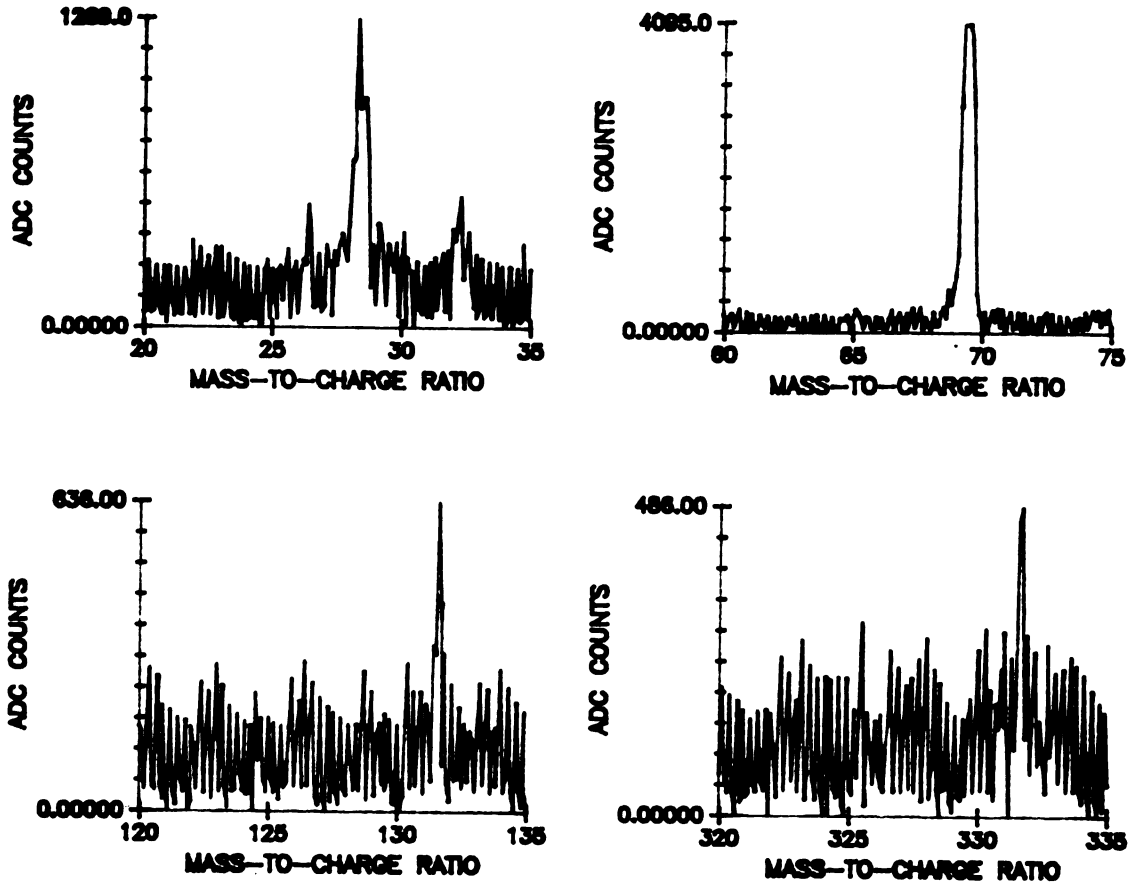


Figure 2.18 Unaveraged mass sweeps of m/z 28 (N_2^+) and m/z 69, 131, and 331 from PFK taken at 2000 AMU/S with a high bandwidth (100 MHz) preamplifier system.

be compared with Figure 2.15. This amplification system consists of a 100 MHz noninverting preamplifier with a transresistance of 100 kV/A in tandem with a 5 MHz inverting amplifier with a gain of 30. Figure 2.18 also shows that higher frequency noise increases greatly with the increased bandwidth. The intensity of the peaks in Figure 2.18 are comparable to those in Figures 2.12-2.15, since the intensity values lack multiamp amplification (256x). This data also demonstrates a slight mass-dependent peak shift, caused by the finite time required for the ions to reach the ion multiplier from the scanning quadrupole. At the faster scan speeds, the control system will be sampling the ion current of previous mass values delayed by the transit time through quadrupoles 2 and 3. This flight time is a function of mass, ion energy, and path length (whether quadrupole 1 or 3 is being scanned) and ranges between 40 to 150 uS at 10 eV for ions from mass 28 to 331 (1 to 3 mass steps). If desired, the control system can calculate corrections for this peak shift or can recalibrate at fast scan speeds. These experiments show that the TQMS instrument can provide usable data even at the fastest scan rates of the Novix control system.

The only instance where the speed of the quadrupole controllers appears to be a problem is in rapid switching of mass values. For example, if the current mass value of a quadrupole is much different from the starting value of a scan, then the software allows a settling time of 10 mS for

that first value. Otherwise, ghost peaks appear as the quadrupole controller tries to change the voltages to the starting value throughout the early part of the scan.

Figure 2.19 shows the timing relationship for a neutral loss scan (NSCAN) on the Novix control system. The unusually large amount of time (18 uS)spent acquiring an unaveraged datum (expected time is 4 uS) from the byte input/output of the data acquisition board. The Novix computer must acquire the 3 bytes of the 20-bit datum as three 16-bit words, mask off the high byte, and combine the lower two bytes into one sixteen bit word. The data acquisition and peak-finding together consume 70 % of the scan cycle. Chapter 4 will describe how the hardware peak finder cuts this time drastically.

5. Conclusion

The Novix NC4000 FORTH computer has been used as the basis for an extremely fast and efficient TQMS control system. The control system allows scan rates heretofore unobtainable on the EL 400/3 triple quadrupole mass spectrometer. Furthermore, at slower scan rates, the increased speed of the control system allows greater amounts of signal averaging, or, alternatively, the implementation of intelligent data collection algorithms. This control system was relatively inexpensive to build, yet provides superior speed (and, thus, flexibility for new experiments)

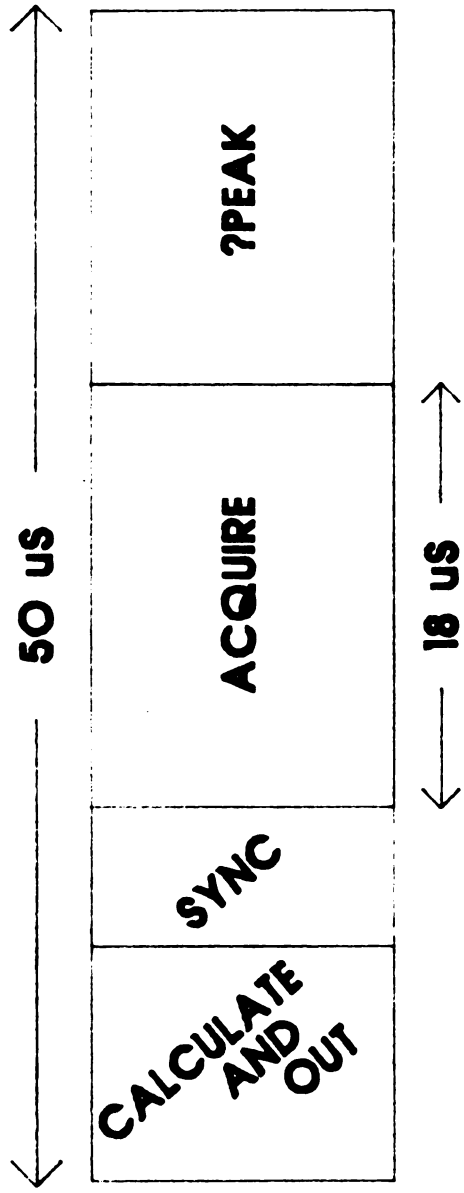


Figure 2.19 Timing relationships for an NSCAN for the Novix TQMS control system.

and inherent ease of repair to multiprocessor systems like the one described in Chapter 1.

REFERENCES

1. Novix Inc., Cupertino, CA.

2. J. H. Golden, C. H. Moore, and L. Brodie, Electronic Design, March 21, 1985.

3. International Business Machines Inc., Boca Raton, FLA.

4. P. Wallich, IEEE Spectrum, 22 (1985) 38.

5. E. Basart, Computer Design, July 1985.

6. J. Mick and J. Brick, *Bit-Slice Microprocessor Design* (St. Louis: McGraw-Hill Book Co., 1980).

7. MetaForth Computers Ltd., Hull, England.

8. Data Technology Corporation, Santa Clara, CA.

9. B. H. Newcome, personal communication.

10. Digital Equipment Corporation, Marlboro, MA.

11. National Semiconductor Corp., Santa Clara, CA.

12. Advanced Micro Devices, Sunnyvale, CA.

13. Monolithic Memories Inc., Sunnyvale, CA.

14. C. A. Myerholtz, M. S. Thesis, Michigan State University, 1984.

15. C. A. Myerholtz, A. J. Schubert, M. J. Kristo, and C. G. Enke, Instruments and Computers, 3 (1985) 11.

16. Forth Inc., Hermosa Beach, CA.

CHAPTER THREE

A FAST, RELIABLE SOFTWARE PEAK-FINDING ROUTINE FOR MS/MS

1. Peak-Finding

A typical scientific experiment, called a "scan," consists of sequentially changing the value of one variable in a system, while recording the intensity of another, dependent variable. Under computer control, the experiment consists of repetitively setting the value of the independent variable, usually a physical device controlled by a digital-to-analog converter, and recording the value of the intensity of the dependent variable. The dependent variable is usually a physical phenomenon, which is first converted into the analog domain (a voltage or current) by a transducer and then into the digital domain by an analog-to-digital converter (ADC). These repetitive measurements continue until the computer has scanned the device over a range specified by the user. The information contained in a scan normally consists of the increases and decreases (peaks and valleys) in the intensity versus the independent variable. Chromatograms from liquid or gas chromatography, absorption and emission spectra in optical spectroscopy or mass spectra in mass spectrometry typify data obtained from scans.

Many times, however, what the scientist really uses is not the full array of device-value/intensity pairs (the full

peak profiles), but rather the maximum intensity of the peaks in the scan and the device values at which they occurred. Thus, he would rather see the raw sequential data reduced to a more convenient and manageable form. This gives rise to the necessity of formulating algorithms for the computer to perform "peak-finding." Besides merely extracting the most useful information in the raw data for the user, the peak-finding algorithm greatly reduces the amount of data that must be stored. This can be used to both increase storage speed and decrease storage space.

Peak-finding is, by no means, a trivial task. There are several conditions which can cause even experts in an instrumental technique to disagree on the existence or nonexistence of a peak in a given scan. First, lack of intensity can cause peaks to fade into background. Second, resolution between adjacent peaks often can be very poor, that is, one has extremely small valleys for given peak heights. The variable resolution in quadrupole mass spectrometry is shown by the two mass sweeps in Figure 3.1. Third, electronic noise can create misleading signals. Fourth, instrumental conditions can create bizarre peak shapes, which can, for example, suggest the presence of multiple peaks where really only one exists.

Figures 3.2 and 3.3 give commonly-found peak shapes on our instrument, a triple quadrupole mass spectrometer (1). Such peak-shapes are usually indicative of bad "tuning" of the instrument, but finding the perfect tune can be elusive.

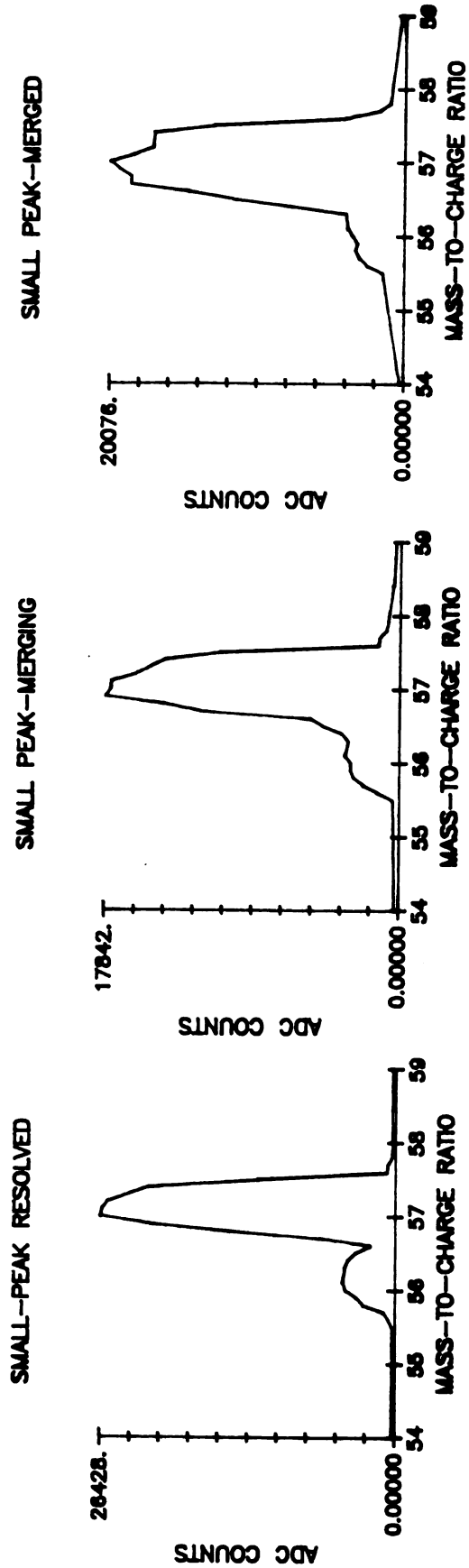


Figure 3.1 Three mass sweeps showing the variable resolution between peaks in TQMS.

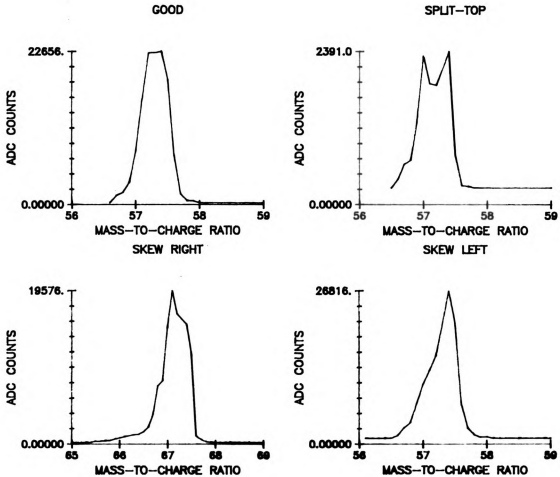


Figure 3.2 Common peak profiles found in TQMS.

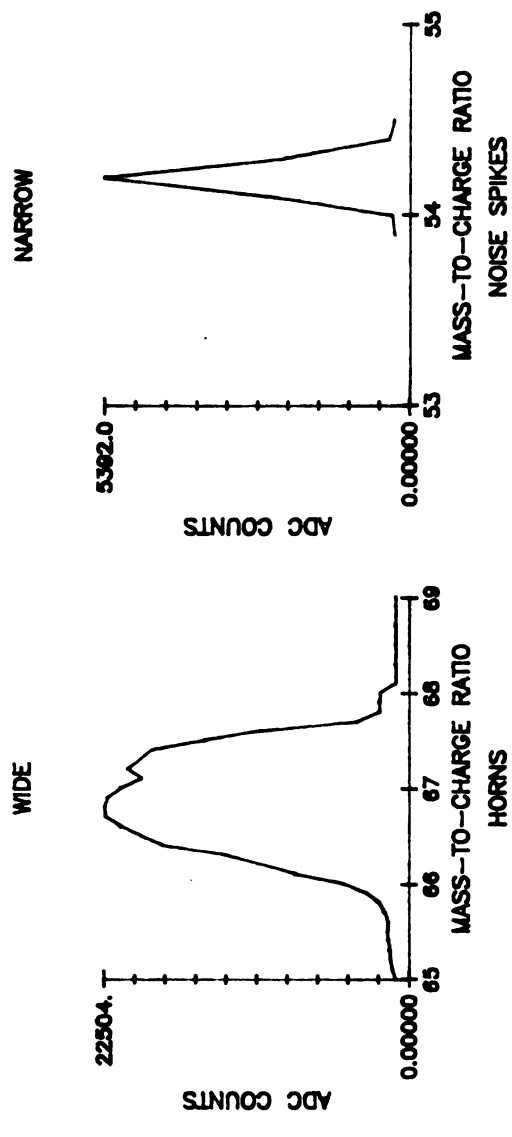


Figure 3.3 Common peak profiles and an example of noise spikes found in TQMS.

Further experiments, in which usually static parameters such as the resolution control are varied, can help clarify the existence of peaks. However, before undertaking further experiments, it is necessary to know that a problem does indeed exist. In other words, access to the raw data which is being reduced is also needed. However, the algorithm operates as a black box to the user, into which raw data disappears and from which only the processed results appear. There are very few clues in the reduced data on which to base an opinion of whether extraneous or missing peaks are real or an artifact of the algorithm. Users can be quite demanding in their expectations of the peak-finding algorithm which causes the algorithm to become an easy scapegoat. A practical definition of a good peak-finding algorithm is: a sequence of mathematical or logical steps which processes raw data and indicates a peak in every instance that the user would, if he were to examine the raw data, and in no other.

The programmer must first decide whether the peak height or the peak area is a more appropriate measure of the peak intensity. This is, of course, dependent on the application. For many applications, the peak height is the appropriate measure of signal intensity. Even in applications such as chromatography where peak area is more characteristic of intensity, peak height can often give more reproducible results, depending on the method of area calculation (2). Peak height is one appropriate measure for

mass spectrometric intensity and so my algorithm searches for the peak maximum, rather than calculate the peak area, although it could be easily modified to do so. Finding the peak height is also faster and easier to program. A related problem is whether to calculate the peak position as the device value at which the maximum intensity occurs or whether to make a centroidal calculation. The essential issue here is which measurement of peak position gives the more repeatable value scan to scan, since the mass scale is calibrated on a frequent basis. No conclusive proof has been offered, so the programmer generally chooses the method of calculating peak position which complements his method of finding peak intensity. For peak heights, the device value at the peak height is the logical way to calculate peak position.

The programmer of a peak-finding algorithm has further to choose between real-time processing or post facto processing. Post facto processing involves gathering all of the raw data in a scan and then performing peak-finding. This has the advantages of random access to every piece of data, allowing more sophisticated algorithms such as correlation functions (3,4) to be employed, and relief from overly critical timing requirements. My experiments show that correlation functions have limited utility in triple quadrupole mass spectrometry, since correlation functions measure the correspondence of the current peak profile to a reference peak profile yet triple quadrupole peaks can vary

widely in shape. In addition, correlation functions do not deal readily with signal offsets without some sort of high-pass digital filter.

The time that it takes to perform peak-finding in post facto processing will not influence the rate of scanning, but it will influence the cycle time from scan to scan. The disadvantage of post facto peak-finding is that large data buffers must be used to store all data points in a scan. As an example, our data system requires 8 bytes of memory to store one datum point: 2 bytes for device value, 2 bytes for peak width and flags, and 4 bytes for intensity. For storing a full peak profile with equally spaced device steps, only 4 bytes would be really needed, but would require calculation of the device value for each intensity value before listing or displaying the data. There are typically 10 points per mass unit and a mass range of 1000. Thus, at least 40,000 bytes of memory would be needed to store the raw data for a single scan. Since we use Intel hardware (5), this buffer would be in extended memory and so would be awkward and slow to address.

Real-time peak-finding has complementary advantages and disadvantages. It reduces greatly the need for large memory buffers. For instance, one would expect, at most, 1000 peaks in the typical mass spectrum. Thus one requires 8000 bytes for storage of a processed spectrum instead of at least 40,000 bytes for storage of a spectrum prior to processing. Main memory can usually support a buffer this large,

depending on the size of the application code. Since one has access only to the current piece of data and, depending on the algorithm, certain selected pieces of previous data, less sophisticated algorithms must be used. The magnitude of the noise cannot be easily calculated, for instance. Background levels rarely change greatly on a day-to-day basis for a given scan type, however, so the user can enter a threshold value before the scan, below which peaks will not be found.

The time spent in performing peak-finding will obviously affect the fastest scan rate possible, since peak-finding is performed after the acquisition of each datum. Thus, one desires the peak-finding algorithm to execute as fast as possible. However, one would expect faster scan-to-scan cycle times for real-time algorithms, since they merely move the peak data to permanent storage after the scan. Post facto algorithms must perform all peak-finding after the scan and then move the data to permanent storage.

2. Instrumentation and Software

The algorithm described in this Chapter was written for application in the control systems of the two triple quadrupole mass spectrometry (TQMS) instruments in our laboratory. One control system, controlling an Extrel 400/3 triple quadrupole mass spectrometer (6), has a Matrox 8086 CPU card in a Multibus I bus system. The other control system is the multimicroprocessor system described in

Chapter 1 with one master and three slave 8088 microprocessors in a homebuilt system of Newcome-Enke design (7). The algorithm resides in the Reduction Slave whose task is to perform real-time peak-finding and to display graphics. Both systems run the 8086/8088 polyFORTH I kernel. The multimicroprocessor runs a distributed-processing FORTH, modified from polyFORTH (8), and both systems run a FORTH-based software control system for the TQMS instrument (9). The algorithm was written in high-level polyFORTH I, but syntax appropriate to the FORTH-79 and FORTH-83 standards, with double-length extensions, was used.

Recently, the algorithm was also installed on a control system featuring the Novix NC4000 (10) running polyFORTH II (11) by merely resolving the differences between the byte-addressing of the 8086/88 FORTH and the word-addressing of the NC4000 FORTH. This control system was described fully in Chapter 2.

3. The Algorithm

Some applications use post facto peak-finding and several such algorithms are available (3,4,12,13). I chose to use real-time peak-finding in the control of our triple quadrupole mass spectrometer, but there are fewer reports in the literature about algorithms of this kind (14,15,16). The three most important decisions in writing a peak-finding algorithm are: 1) the conditions that must be met before the

algorithm starts looking for a new peak, 2) the conditions that must be met in order to update the maximum intensity for the current peak and 3) the conditions that must be met in order to indicate that a peak is present (peak termination criteria). These three aspects should be implemented with as little dependence as possible on peak shape or day-to-day variations in peak widths or heights. Also, some immunity to noise spikes should be included. The functions in my real-time peak-finding algorithm which address these needs are summarized in the flow chart of Figure 3.4 and are discussed below.

Looking for a New Peak

Most algorithms start looking for the maximum intensity of a new peak immediately after finding the old one. When looking for a new peak, however, the previous peak must be allowed to reach threshold or the valley between the two peaks. If the previous peak has not terminated before the peak-finding algorithm starts looking for a new one, the tail of a wide peak may be incorrectly interpreted as another peak. This leads to the phenomenon known as peak-splitting.

My algorithm does not monitor the absolute intensity alone, but also the difference between the current and previous intensities (the increases and decreases in intensity). Thus, the algorithm first compares the current and previous intensities to determine whether the intensity

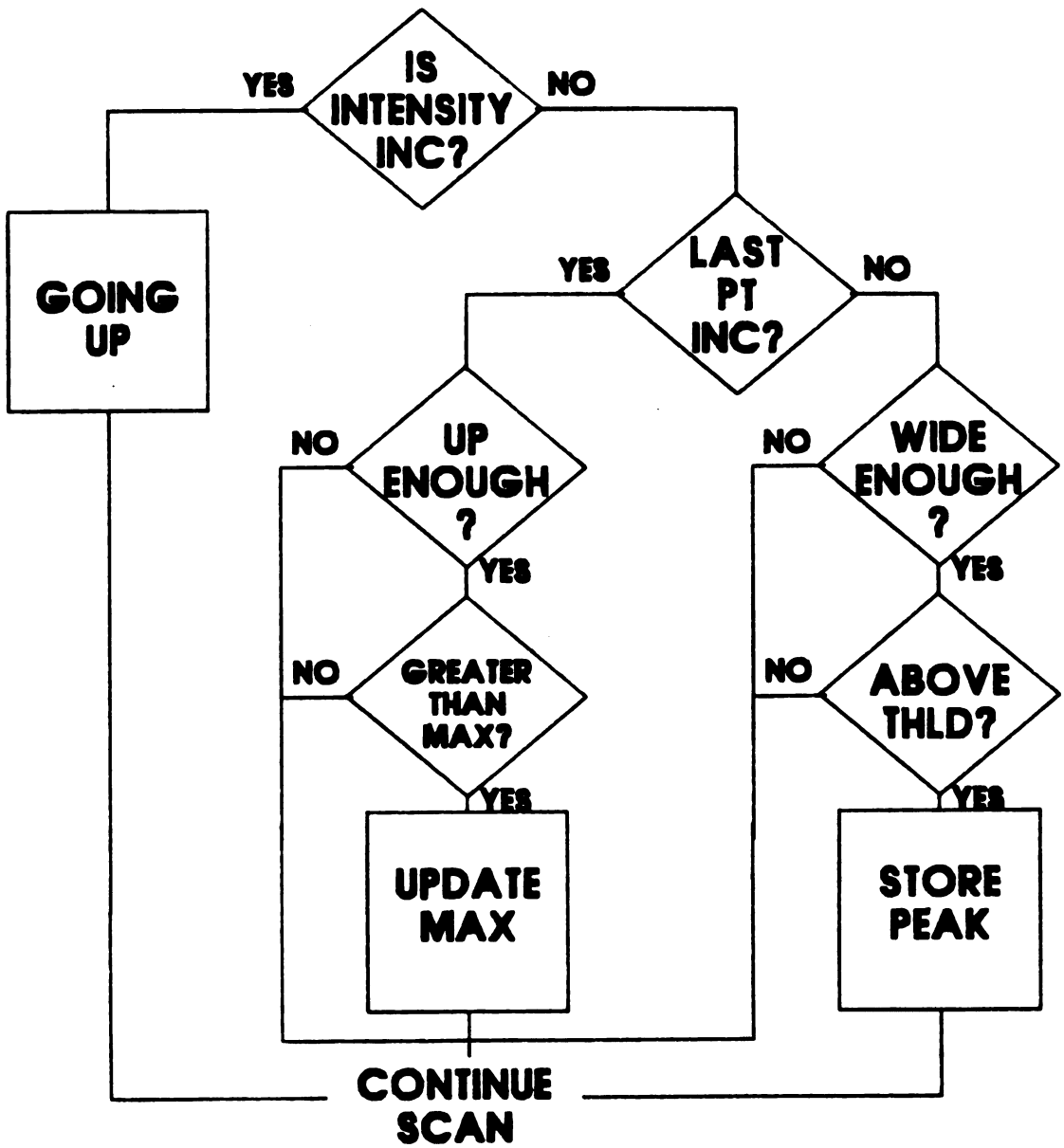


Figure 3.4 The flow of decision processes in the current peak-finding algorithm.

is increasing or not (labeled "IS INTENSITY INC?" in Figure 3.4). At least two increases in intensity (not consecutive) must occur before the algorithm starts to search for the new maximum, i.e., a minimum of 3 consecutive or 4 nonconsecutive points with an upward trend. After a peak is found, therefore, the intensity will necessarily reach threshold or its minimum intensity before a new peak search begins. Again, this is especially important when a large peak is followed by a much smaller one, since it avoids splitting the large peak's tail, possibly overwhelming the signal of a smaller peak in the same region. The algorithm's requirement for two increases (labeled "UP ENOUGH?" in Figure 3.4) instead of just one, attempts to avoid false triggering by noise fluctuations on the downside of the previous peak. Technically, this is accomplished by requiring that the variable UPFLAG, which measures the number of intensity increases for the current peak, be greater than one in order to update the maximum again.

Updating the Maximum

Generally, a peak-finding algorithm will test each point to find the point of maximum intensity in a peak. The exception to this in my algorithm, as mentioned previously, is allowing the intensity to reach threshold or the valley between two peaks before starting to test for the new maximum, since the intensity of the tail of the previous peak may be more intense than the maximum of the following

peak. If the intensity of the peak is currently decreasing, but was increasing on the previous datum (the variable +LAST was set), then the previous point was a local maximum, by definition. If such a local maximum has a value greater than the current value for the maximum intensity of the peak (stored in the variable MAX-PEAK), then the value of the previous point becomes the new value for the maximum intensity. Therefore, if a peak is currently being monitored (UPFLAG greater than 1) and the previous point was a local maximum with an intensity greater than the current value of MAX-PEAK (labeled "GREATER THAN MAX?" in Figure 3.4), then my algorithm updates the maximum (labeled "UPDATE MAX" in Figure 3.4). (MAX-PEAK then becomes the intensity of the previous point and XMAX, the associated device value, becomes the device value for the previous point.)

Peak Termination

When terminating a peak, the peak-finding algorithm should be sure of three things. First, the algorithm should ascertain that the maximum intensity of the current peak has been reached. This could be accomplished by comparing the current intensity to some fraction of the maximum intensity or to a user-set threshold, although the algorithm may become more sensitive to negative noise spikes. Alternatively, the algorithm could verify that the basic trend of the peak is downward, that is, that the intensity has been decreasing over the last few points. Second, the

algorithm should determine that the peak is wide enough to constitute a real peak and not a noise spike. "Wide enough" is a relative term which will vary from application to application and may vary from experiment to experiment. Third, the algorithm should make sure that the intensity is significant enough. For a real-time algorithm, the peak intensity should be compared to a user-set threshold. This threshold should approximate the background offset plus the background noise in the system, or should be large enough to reject small peaks in which the user is not interested.

Therefore, in my algorithm, the peak is terminated if the following three conditions are satisfied: 1) there have been at least two consecutive decreases (the peak is currently decreasing and the variable +LAST is cleared-labeled "LAST PT INC?" in Figure 3.4), 2) the current width of the peak (kept track of in the variable POINTS) exceeds the user-set minimum contained in the variable PWIDTH (labeled "WIDE ENOUGH?" in Figure 3.4), and 3) the maximum intensity (the variable MAX-PEAK) is above the threshold (labeled "ABOVE THLD?" in Figure 3.4). The minimum practical value for PWIDTH is 4 device steps, since there must be at least two increases to start looking for a peak and at least two decreases to terminate the peak. Thus, setting PWIDTH at 3 or less does not increase the sensitivity of the algorithm to narrower peaks. If a peak is found, the maximum intensity and its associated device

value, along with the peak width and any flags, are moved to the data buffer and all peak-finding variables are zeroed.

Further Notes on the Algorithm

The flow of the software which implements the peak-finding algorithm is shown in Figure 3.5. Besides the implementation of the three important functions discussed above, the software takes care of the details discussed in the following paragraphs.

When the intensity is increasing, the algorithm merely notes the fact by increasing UPFLAG, setting +LAST and proceeding (labeled as "GOING UP" in Figure 3.4 as well). Thus, while intensity is increasing, very little time is spent on peak-finding.

The first increase in intensity also starts the width counter (the variable POINTS). Otherwise, when POINTS is equal to zero, a peak has been found recently and the criterion for looking for a new peak has not been satisfied, so POINTS is not increased.

The algorithm concludes by updating YPREV, the previous intensity value. Using THRESHOLD as a screen for new data (data below THRESHOLD are not processed) would speed up the algorithm quite a bit, but would interfere with the tracking of the increases and decreases of intensity. This is especially important when a peak starts below THRESHOLD and ends above it.

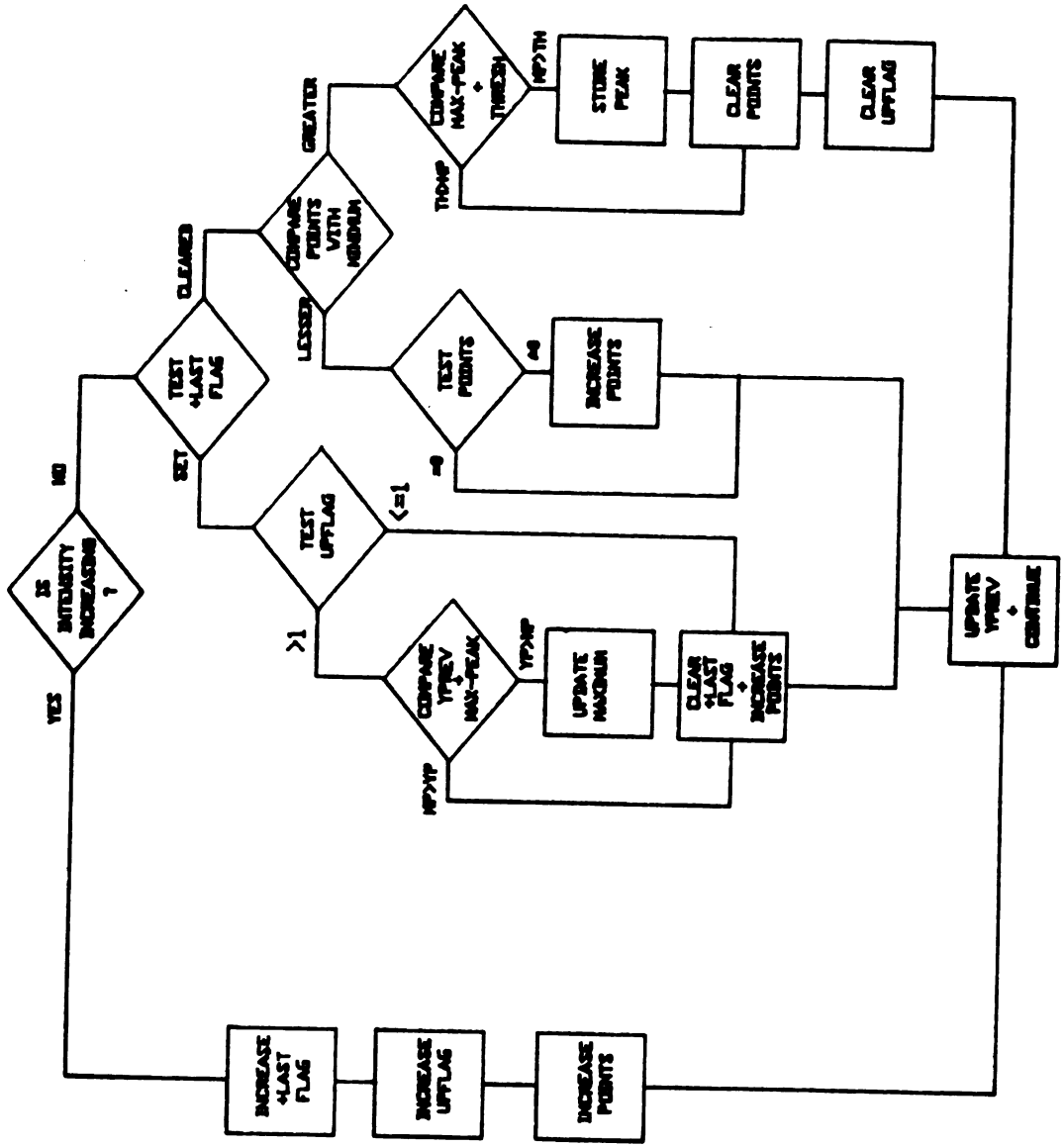


Figure 3.5 The flow of the software implementing the current peak-finding algorithm.

The FORTH code which implements this algorithm can be found in Appendix D. For increased speed, I have implemented the algorithm in FORTH Assembler for our 8086 and 8088 microprocessors. For Intel enthusiasts and other interested parties, this code is also given in Appendix D. In some respects, the algorithm was developed for final implementation in assembly language. Thus, instead of using the stack, my peak-finding algorithm uses VARIABLEs for storing values, making the code easier to understand. Using variables makes the algorithm execute more slowly in high-level FORTH, but not in FORTH Assembler. Assembly language routines access the stack and other memory locations with equal speed.

Comparison With Other Real-Time Algorithms

Few descriptions of real-time peak-finding algorithms can be found in the literature. Most of the work in this area has been done by scientific instrument manufacturers who consider their algorithms proprietary. The actual code for their algorithms can be hard to obtain, although the user's manual for the instrument itself can hint at some of the features of the algorithm in its discussion of user-controlled variables. The few algorithms that can be found in the literature do not tend to be the major focus of the paper, but merely a sidelight. Hopefully, the increasing popularity of digital signal processing and chemometrics

will help to focus attention on this most important form of signal processing and data reduction.

The algorithms described in the literature approach real-time peak-finding in different ways. One common approach to peak-finding updates the maximum if it exceeds the current maximum and indicates a peak if the intensity falls below either a user-set threshold or a certain fraction of the maximum intensity (e. g., 1/2 or 3/4) (14,15). The tests for a new peak usually begin immediately, because single intensity values (rather than the difference in successive intensities) provide no easy means to determine the end of the previous peak.

In these algorithms, spurious, "fragmented" peaks are often falsely indicated on the shoulders of real peaks, since the bottom of a peak is not sought. By post-processing the peak data, one can guess which of the indicated peaks are merely fragments, especially if one knows how frequently a peak is expected (as in low-resolution mass spectrometry, where they rarely occur closer than 1 mass unit apart). In order to reduce this peak fragmentation and reduce the susceptibility to large noise spikes, a minimum width function is is often used. However, the minimum width function reduces fragmentation only if the fragment is indicated as a peak but is narrower than the minimum width, in which case it is disregarded. Many times, however, the algorithm cannot distinguish the fragment from the following peak and, if the fragment is larger than the

following peak (as is often the case with isotope peaks), the position and intensity of the fragment will be indicated as the position and intensity of the following peak. In this case, the width of the fragment plus the width of the following peak is sufficiently large to exceed the minimum width requirement. Furthermore, if the spectrum contains both wide and narrow peaks, the high minimum width needed to reject the indication of the shoulders of the wide peaks may also reject the indication of the narrow peaks. This can be a big problem in magnetic sector mass spectrometers in which the peak width varies with mass.

Another approach to peak-finding specifies a maximum width. If the intensity has not fallen below threshold and the peak width has reached this maximum, then the peak is indicated. This approach leads to problems with clusters of poorly resolved peaks. If the intensity does not fall below threshold between the peaks, then a peak will be indicated each time the maximum width is reached. If one specifies a maximum width of 15 steps, for example, then two peaks will be found every 3 mass units (10 steps per mass unit), missing one peak.

In order to perform accurate peak-finding and eliminate the problems described above, an algorithm must use all of the criteria implemented in my algorithm: examining the difference between successive intensities as well as the magnitude of the current intensity, waiting for the termination of the previous peak before searching for the

next peak, comparing the current width with a minimum width, and comparing the peak intensity with a user-set threshold. Ignoring any of these criteria in an algorithm can lead to problems under appropriate circumstances. Thus, reliable peak-finding is time-consuming and can easily limit scan rates. Furthermore, the difficulties in peak-finding described above become worse with increasing dynamic range. Since mass spectrometry/mass spectrometry (MS/MS) provides a dynamic range of 10^9, peak-finding is especially difficult. Therefore, for MS/MS, one must use the best algorithm possible for reliable peak-finding, yet find ways to implement the algorithm quickly enough to minimize the limitation of scan rates.

4. Algorithm Testing

The acid test for any algorithm is how it behaves in actual use. This algorithm and the MSSIN algorithm (15) were applied to a series of 16 peak profiles to test their performance. The profiles were chosen to represent a wide range of peak-finding problems. The profiles were gathered from real, raw data and were normalized to prevent any one example from becoming harder or easier for the algorithm to handle, based solely on its intensity. Since all of the algorithms are basically software state machines, how they handle a given peak depends not only on the peak profile itself, but also on the the value of the algorithm's state variables (variables whose value influences the course of

action of the algorithm) and user-defined variables (threshold, minimum width, etc.) upon encountering the profile. The state variables provide the only influence that the data in the preceding part of the spectrum have on the processing of the current datum. Each algorithm was run through the profiles with all mathematically possible starting states. Obviously, some combinations of starting states and peak profiles are either physically impossible or highly unlikely. Therefore, one must temper the success or failure of the algorithm for a given state-profile combination with an intuitive knowledge of the likelihood of its occurrence.

Unfortunately, each algorithm has a different number of state and user-defined variables (Table 3.1), so it is difficult to compare algorithms quantitatively. It is also difficult to quantitate the likelihood of the experimental occurrence of a state-profile combination. Nevertheless, this quantitation is necessary as an absolute measure of the success of the algorithm. So, testing the algorithms for all possible starting states provides, at best, only a qualitative view of their strengths and weaknesses. However, this scheme can provide information about situations in which the algorithm's performance may be suspect, so that such situations can be avoided. Also, the testing scheme immediately exposes most errors in an algorithm. The testing code that I developed can be found in Appendix E.

Table 3.1
Variables for Kristo Algorithm
and MSSIN

State Variables

Kristo Variable	MSSIN Variable	Algorithmic Function
MAX-PEAK	MI	Maximum Intensity
POINTS	NP	Current Width
UPFLAG	UF	Intensity Increases
+LAST		Last Pt Increasing?
YPREV		Previous Intensity

User-Defined Variables

Kristo Variable	MSSIN Variable	Algorithmic Function
THRESHOLD	D3	Intensity Threshold
PWIDTH	NP	Minimum Width

5. Algorithm Performance

Table 3.2 compares the performance of the present algorithm and that of another popular algorithm, MSSIN, given in reference 15 for one set of starting conditions. The values of each algorithm's variables indicate that they are not currently monitoring a peak (maximum intensity is zero), that the previous point was decreasing (+LAST=0), but that the next point will be increasing (previous intensity=0). This means that any previous peak has been indicated and has either reached threshold or the valley between peaks. Both algorithms were run through the peak profiles with values of the minimum width ranging from 3 to 6. For reference, depending on the value of the resolution

control, quadrupole mass spectrometric peaks can range from 4 to 15 points wide.

The results show that the MSSIN algorithm fragments peaks at low values of the minimum width. It does not fragment them at higher values, but then narrower peaks are not found. My algorithm also does not find the narrower peaks at high values of the minimum width, but does not have problems with fragmenting at lower values. Also, note that my algorithm found a peak for noise spike 2 at a minimum width of three. This particular noise spike resembles a real peak in that it has a typical profile with a width of four points. Thus, it is not surprising that my algorithm indicated a peak under those conditions. The user must compromise between sensitivity and noise immunity. The lower minimum width is sensitive to very narrow peaks, but also to fairly wide noise spikes. Of course, in general, noise spikes have a lower intensity than do peaks, so that they may be rejected at an appropriate threshold value. Unfortunately, this is not always the case.

TABLE 3.2

COMPARISON OF TWO REAL-TIME
PEAK-FINDING ALGORITHMS
Starting from Threshold

PEAK PROFILE	MSSIN (15) MIN. WIDTH VALUE				PRESENT WORK MIN. WIDTH VALUE			
	3	4	5	6	3	4	5	6
Good	F							
Skew Right	F							
Skew Left	F							
Wide	F	F	F					
Narrow			M	M			M	M
Horns	F	F						
Splits								
Isotope-Coalesced	FM	FM	FM	FM	M	M	M	M
Isotope-Merges	F	F	F	F				
Isotope-Resolved								
Small-Coalesced	FM	M	M	M	M	M	M	M
Small Merges	FM	M	M	M	M	M	M	M
Small-Resolved	F							
Noise No. 1	0	0	0	0	0	0	0	0
Noise No. 2	0	0	0	0	1	0	0	0

KEY: M - MISSED A PEAK
F - FRAGMENTED A PEAK
0 - IGNORED A SPIKE
1 - CALLED SPIKE A PEAK

The results shown in Table 3.3 show the performance of the algorithms when they encounter the 16 test profiles after the tail of a previous peak. In this case, the intensity of the tail is larger than the maximum intensity of the test profiles. Upon encountering the given profiles, then, the maximum intensity for MSSIN is the intensity of the tail, but is zero for my algorithm, since the maximum intensity cannot be updated before two intensity increases. The variables UPFLAG (UPCHECK) and +LAST are zero since the intensity has not increased since finding the previous peak.

The results in Table 3.3 show that my algorithm avoids increased fragmentation and sensitivity to noise spikes, by waiting for the previous peak to reach threshold or minimum intensity. Fragmentation occurs with the MSSIN algorithm because the tail is incorrectly identified as a peak. In fact, this intensity can supersede the maximum of neighboring peaks with low intensity.

TABLE 3.3

COMPARISON OF TWO REAL-TIME
PEAK-FINDING ALGORITHMS
Coming Off a Large Peak

PEAK PROFILE	MSSIN (15) MIN. WIDTH VALUE				PRESENT WORK MIN. WIDTH VALUE			
	3	4	5	6	3	4	5	6
Good	F^2	F	F	F				
Skew Right	F^2	F	F	F^2				
Skew Left	F^2	F	F	F				
Wide	F	F^2	F^2	F	F	F	F	
Narrow	F	F	F					
Horns	F^2	F^2	F	F				
Splits	F	F	F					
Isotope-Coalesced	F^2M	F^2M	F^2M	F^2M	FM	FM	FM	FM
Isotope-Merges	F^2	F^2	F^2	F^2M				
Isotope-Resolved	F	F	F	F				
Small-Coalesced	F^2M	FM	FM	FM	M	M	M	M
Small Merges	F^2M	FM	FM	FM	M	M	M	M
Small-Resolved	F^2	F	F	FM				
Noise No. 1	1	0	0	0	0	0	0	0
Noise No. 2	1	1	0	0	1	0	0	0

KEY: M - MISSED A PEAK
F - FRAGMENTED A PEAK
F^2 - FRAGMENTED A PEAK TWICE
0 - IGNORED A SPIKE
1 - CALLED SPIKE A PEAK

The results shown in Table 3.4 show the performance of the algorithms when they encounter the test profiles after encountering a large intensity increase (or spike). The intensity of the spike is larger than the intensity of the maximum of the test profiles. Once again, the maximum intensity is the intensity of the spike for MSSIN, but is zero for my algorithm, since the maximum intensity cannot be updated before two intensity increases. UPFLAG (UPCHECK) and +LAST are both equal to one since the intensity has increased once since finding the previous peak. The results in Table 3.4 again show that my algorithm avoids increased fragmentation after encountering a large intensity increase before a peak, because the maximum will not be updated without a consistent upward trend. Sharp intensity spikes do not by themselves constitute a consistent upward trend. Although the spike itself is not wide enough to be called a peak, it can cause fragmentation in the following peaks with some algorithms.

TABLE 3.4

COMPARISON OF TWO REAL-TIME
PEAK-FINDING ALGORITHMS
After a Large Intensity Spike

PEAK PROFILE	MSSIN (15) MIN. WIDTH VALUE				PRESENT WORK MIN. WIDTH VALUE			
	3	4	5	6	3	4	5	6
Good	F^2	F	F	F				
Skew Right	F2	F	F	F				
Skew Left	F^2	F	F	F				
Wide	F^2	F^2	F^2	F	F	F	F	
Narrow	F	F	F	FM				M
Horns	F^2	F^2	F	F				
Splits	F	F	F	F	F	F	F	
Isotope-Coalesced	F^2M	F^2M	F^2M	F^2M	FM	FM	FM	FM
Isotope-Merges	F^2	F^2	F^2	F^2M				
Isotope-Resolved	F	F	F	F				
Small-Coalesced	F^2M	FM	FM	FM	M	M	M	M
Small Merges	F^2M	FM	FM	FM	M	M	M	M
Small-Resolved	F^2	F	F	FM				
Noise No. 1	1	1	0	0	0	0	0	0
Noise No. 2	1	1	1	0	1	1	0	0

KEY: M - MISSED A PEAK
F - FRAGMENTED A PEAK
F^2- FRAGMENTED A PEAK TWICE
0 - IGNORED A SPIKE
1 - CALLED SPIKE A PEAK

The results shown in Table 3.5 show the performance of the algorithms when they encounter the test profiles after encountering a small intensity increase (or spike). In this case, the intensity of the spike is smaller than the intensity of the maximum of the test profiles. The conditions upon encountering the profiles is the same for Table 3.5 as in Table 3.4, except that the current value for the maximum intensity is smaller. The results in Table 3.5 again show that my algorithm avoids fragmentation after encountering a small intensity increase before a peak, again by waiting for a consistent upward trend before updating the maximum. However, even with MSSIN, there is reduced fragmentation as compared with a large spike, because the intensity is not enough to overwhelm the intensity of the following peaks.

TABLE 3.5

COMPARISON OF TWO REAL-TIME
PEAK-FINDING ALGORITHMS
After a Small Intensity Spike

PEAK PROFILE	MSSIN (15) MIN. WIDTH VALUE				CURRENT WORK MIN. WIDTH VALUE			
	3	4	5	6	3	4	5	6
Good	F							
Skew Right	F							
Skew Left	F							
Wide	F	F	F		F	F	F	
Narrow								M
Horns	F^2	F^2	F	F				
Splits					F	F	F	
Isotope-Coalesced	FM	FM	FM	FM	FM	FM	FM	FM
Isotope-Merges	F	F	F	FM				
Isotope-Resolved								
Small-Coalesced	FM	M	M	M	M	M	M	M
Small Merges	FM	M	M	M	M	M	M	M
Small-Resolved	F							
Noise No. 1	0	0	0	0	0	0	0	0
Noise No. 2	0	0	0	0	1	1	0	0

KEY: M - MISSED A PEAK
F - FRAGMENTED A PEAK
F^2 - FRAGMENTED A PEAK TWICE
0 - IGNORED A SPIKE
1 - CALLED SPIKE A PEAK

6. Algorithm Speed

Table 3.6 shows the average execution time for the peak-finding algorithm. The time for the algorithm to process 13 points which defined a basic peak shape was measured. The peak profile provides opportunity for the algorithm to exercise all the decision processes, similar to a true scan. The time required to process the profile was then divided by 13 to give the average time spent per point. The execution time also includes two FORTH literals, which place the intensity value on the data stack. Fortunately, the algorithm itself executes much more slowly than the literals, so the timing reflects fairly accurately the true execution time. Execution times are given for both the algorithm written in high-level polyFORTH I on an 8088 processor, FORTH 8088 Assembler, and polyFORTH II on an NC4000.

TABLE 3.6

Timing Information

Language	Processor	Speed
FORTH Assembler	5 MHz 8088	215 uS
High-level FORTH	5 MHz 8088	823 uS
High-level FORTH	6 MHz NC4000	16 uS

7. Conclusion

A real-time peak-finding routine has been developed which executes quickly and performs reliably even under

adverse conditions. The algorithm has some flexibility through user-adjustment of the variables PWIDTH and THRESHOLD. Background noise and small peaks can be effectively filtered by THRESHOLD. Adjustment of PWIDTH allows the user to compromise between sensitivity to narrow peaks and immunity to noise spikes. The algorithm was designed for mass spectrometric peak-finding, but should be applicable to other fields such as atomic emission spectroscopy with little or no modification.

My study of peak-finding and peak-finding algorithms has shown that it is important to examine the upward or downward trends in the intensity and not just the magnitude of the current intensity value. This feature allows the previous peak to reach threshold or its minimum intensity and a consistent upward trend to be demonstrated before a new search for the peak maximum begins, eliminating the splitting of peaks. The examination of differences in intensity also enables the establishment of a consistent downward trend before indicating a peak, rather than depending on the intensity to fall below a certain level. It is also important for a peak-finding algorithm to have a minimum width requirement to reduce sensitivity to noise spikes, although inappropriate minimum width values can also reduce the sensitivity to narrow peaks. Thresholds are also a common and useful function in peak-finding algorithms, because they eliminate problems with normal variation in background intensity as well as small peaks.

References

1. R. A. Yost and C. G. Enke, Anal. Chem., 51, 1979, 1251A.

2. Aleksander Janik, J. Chrom. Sci., 13, 1975, p. 93.

3. N. W. Bell, "Computer Detection of MS Peaks by Real Time Cross Correlation," Technique Paper No. MS-2, Hewlett Packard: Palo Alto, CA.

4. W. F. Bryant, M. Trivedi, B. Hinchman, S. Sofranko, and P. Mitacek, Anal. Chem., 52 (1980) 38.

5. Intel Corporation, Santa Clara, CA.

6. Extrel Corp., Pittsburgh, PA.

7. B. H. Newcome and C. G. Enke, Rev. Sci. Instrum., 55, (1984) 2017.

8. C. A. Myerholtz, A. J. Schubert, M. J. Kristo, and C. G. Enke, Journal of FORTH Applications and Research, Vol. 3, No. 2, 1985, p. 189.

9. C. A. Myerholtz, A. J. Schubert, M. J. Kristo, and C. G. Enke, Journal of FORTH Applications and Research, Vol. 3, No. 2, 1985, p. 193.

10. Novix Inc., Cupertino, CA.

11. Forth, Inc., Hermosa Beach, CA.

12. J. W. Cooper, Minicomputers in the Laboratory, New York: Wiley Interscience Publishers, 1983, pp. 251-7.

13. Laboratory Subroutines Programmer's Reference Manual, Digital Equipment Corporation, Marlboro, MA, 1982, Chapter 2.

14. William H. Caskey, Journal of FORTH Application and Research, 1, p. 11.

15. J. F. Holland, Computer Program "MSSIN," Michigan State University, East Lansing, MI 48824.

16. INCOS Software User's Manual, Finnigan MAT Corp., San Jose CA.

CHAPTER FOUR

HARDWARE PEAK-FINDING FOR RELIABILITY AND INCREASED SCAN RATES

1. Introduction

Reducing sequential intensity data to mass/intensity pairs for all peaks in a mass spectrum occupies much of the time during a mass scan. "Peak-finding" can thus limit the maximum mass scanning speeds to less than that possible for the mass filter employed. Also, since the processor spends so much time on this task, a substantial fraction of the ion current goes unsampled, which leads to a less than maximum signal-to-noise ratio for the data obtained. Therefore, it is desirable to reduce the amount of time necessary to perform peak-finding reliably and to perform this task in parallel with other functions.

I have developed an electronic peak-finding accelerator (PFA) for quick reduction of raw intensity versus mass data to mass-intensity pairs for all peaks in a mass spectrum. The accelerator accepts sequential intensity data from the host control system processor and returns peak positions and intensities for storage. The peak-finder achieves the speed of a dedicated hardware peripheral, yet retains programmability through software sequencing of the hardware functions.

The accelerator's peak-finding program is written in horizontal microcode (64-bits wide), which is tedious to

write by hand. Therefore, a microcode compiler, based entirely on FORTH, has been written to facilitate development of code for new peak-finding algorithms.

2. The Hardware Peak-Finder

Concept

A block diagram of the hardware peak-finder is found in Figure 4.1. Letters in parentheses in the following paragraphs refer to sections on the diagram. The heart of the hardware peak-finder is a RAM-based state machine. The desired peak-finding algorithm is written to the peak-finder's RAM (B) by the host processor. The algorithm is a control sequence program and directs the processing of incoming data. The sequence of instructions in the program is controlled by the next-address latch (C). A 16:1 multiplexer (D) provides the ability to branch in the program on any one of 16 conditions. These conditions (E) are generated by the status of the comparators and flags in the peak-finder. Incoming data are manipulated on a 32-bit bus, which connects the input latches, the output latches, latches for a threshold value, and two banks of comparators with internal registers. Additional counters and flipflops track the occurrence of certain events. Thus, many useful hardware functions can be performed which parallel functions used in software peak-finding algorithms. Furthermore, it

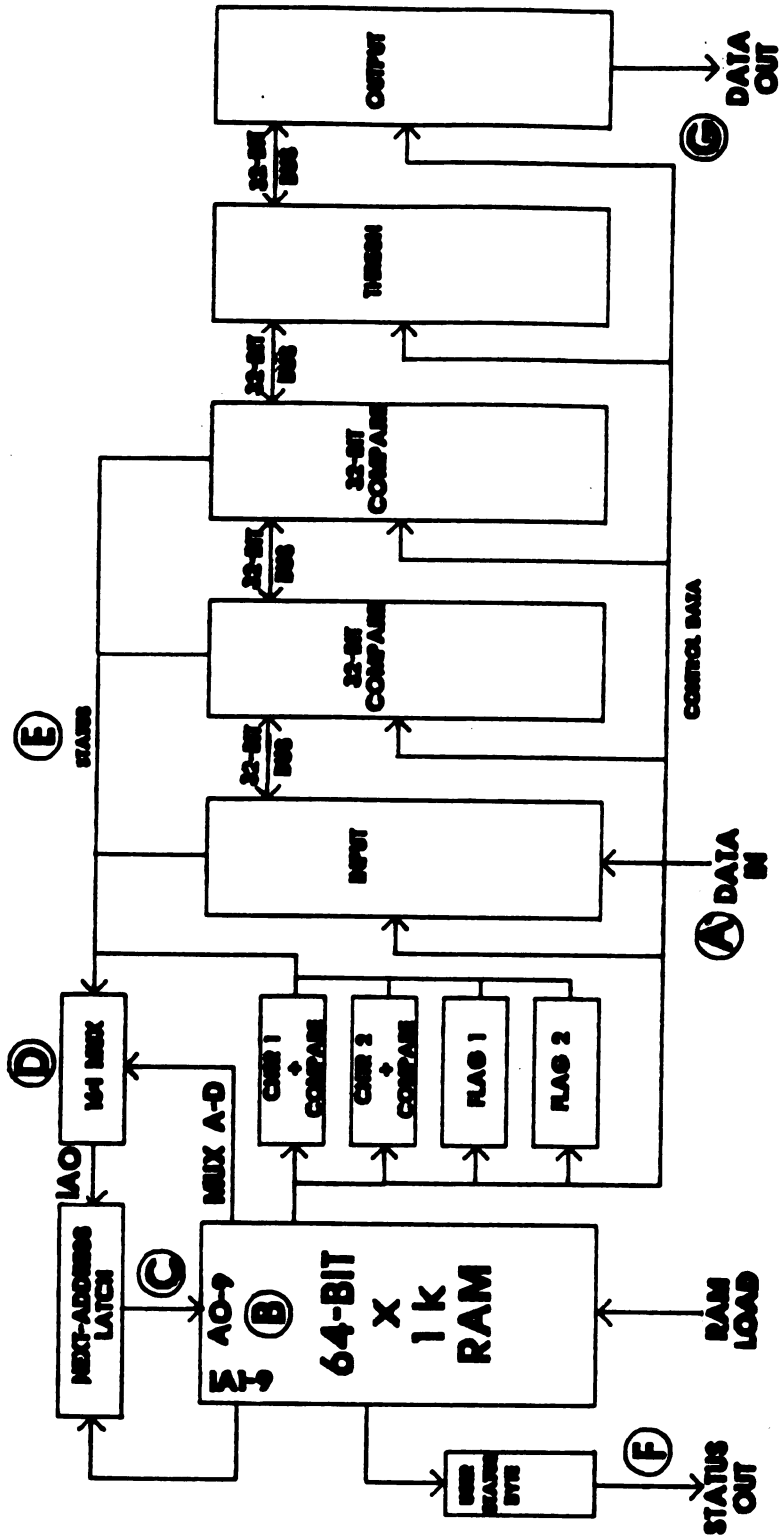


Figure 4.1 Block diagram of hardware peak-finder.

is simple to add further arithmetic elements without extensive redesign.

Acquired data are written into the input latches of the accelerator (A) and peak data are read from the output latches (G). The host processor can monitor the status of the peak-finder (whether the current datum has been processed and whether or not a peak was found) by reading the user status byte (F). With this monitoring capability, the peak-finder can process data simultaneously with the host processor, thus freeing the processor to attend to other tasks.

Interfaces have been designed for two host processors. One host processor is the multimicroprocessor control system described in Chapter One, consisting of four Intel 8088 (1) microprocessors, one master and three slaves. The Detection Slave handles the data acquisition and can also control the peak-finding accelerator. The software for the multimicroprocessor control system (2,3) and for its version of the microcode compiler are written in polyFORTH I (4). The other host processor is the Novix control system described in Chapter Two, based on the NC4000 FORTH processor (5). The software control system and the version of the microcode compiler for the Novix control system are written in polyFORTH II (4).

Hardware Design

The hardware peak-finder was divided into two separate circuits for modularity and simplicity of design. The first circuit, designated as PFA-1000A, contains all of the RAM for the microcoded algorithm, counters for such quantities as the peak width, and flipflops which serve as flags. The second circuit, designated as PFA-1000B, contains the 32-bit intensity (y-value) bus and the 16-bit mass (x-value) bus, consisting of latches and registers for storing numbers and comparators for determining the greater of two numbers. Modifications can be made to either circuit without changing the other. For instance, one can add more memory to PFA-1000A without changing PFA-1000B. Extra memory would allow longer control words (greater than 64-bits) and, thus, more control signals. The extra signals could be used to control a multiplier-accumulator (MAC) to allow real-time centroidal calculations. The two circuits transmit and receive signals to and from each other over a 50-pin ribbon cable. Separating the peak-finder into two circuits also prevented troubles with the computer-aided design (CAD) system on which they were designed. The memory-management software on the CAD computer has trouble handling extremely large numbers of parts (greater than about 40 integrated circuits).

The three processor address lines (PA0-2), the processor chip select line (/CS), and the processor read

(/RD) and write (/WR) lines are decoded in PFA-1000B to select the various chip which control the peak-finder (shown in Figure 4.2). The lower six of the possible eight addresses select the bytes for writing the input intensity and x-value data and reading the peak intensity and mass data. One of the upper two addresses allows reading/writing to the random access memory, while the other address allows either reading from the event counter or changing the mode of the peak-finder to either the load mode (writing and verifying the microcode for the peak-finding algorithm) or run mode (processing incoming data). When changing the mode of the PFA, the board can also be reset to initialize all devices and set the currently selected address to zero. Setting appropriate jumpers in PFA-1000A allows the PFA to be addressed either as bytes or words. Thus, the peak-finder can be optimized for processors with either 8-bit or 16-bit input/output.

Some of the general signals generated in PFA-1000B are further combined with memory address selection circuitry (shown in Figure 4.3) in PFA-1000A. The processor can then read or write to each byte in the microcode RAM when the PFA is in load mode. The selection of a particular byte in a particular word in the memory occurs through an autoloading feature (shown in Figure 4.4). Therefore, RAM addresses must be sequentially accessed when either reading or writing. When the PFA is in run mode, all eight bytes of memory (64 bits wide) are accessed at once, making all

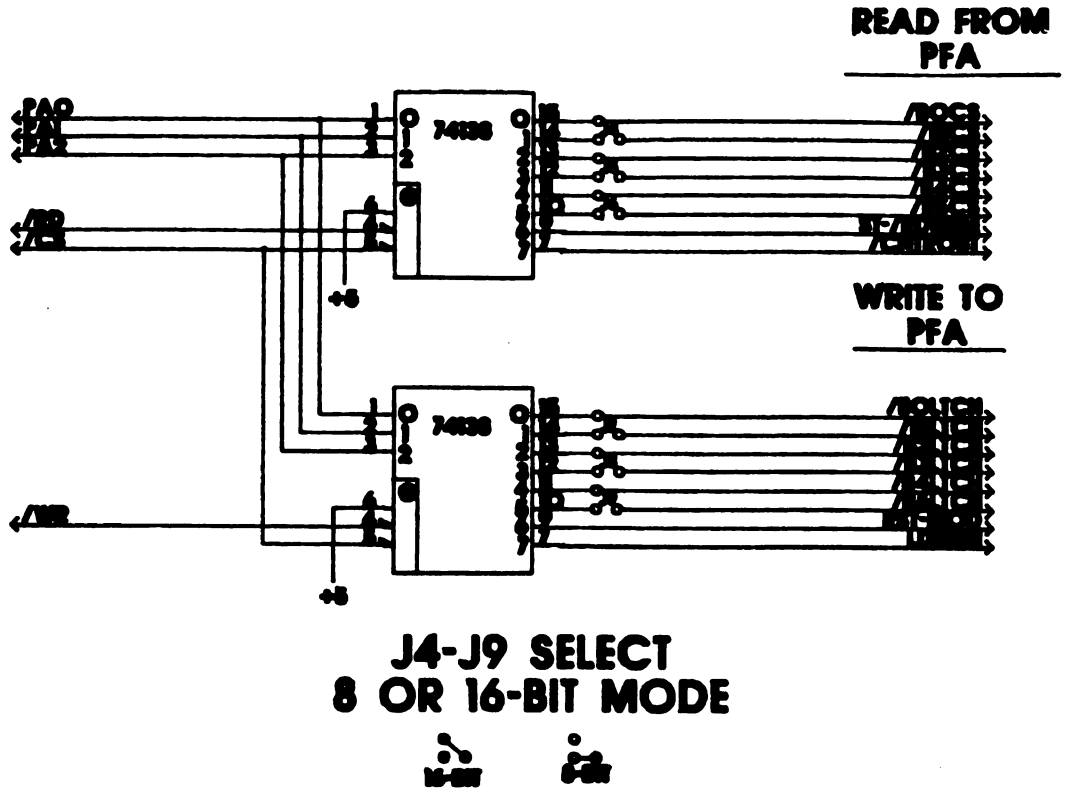


Figure 4.2 Schematic of address-decoding circuitry.

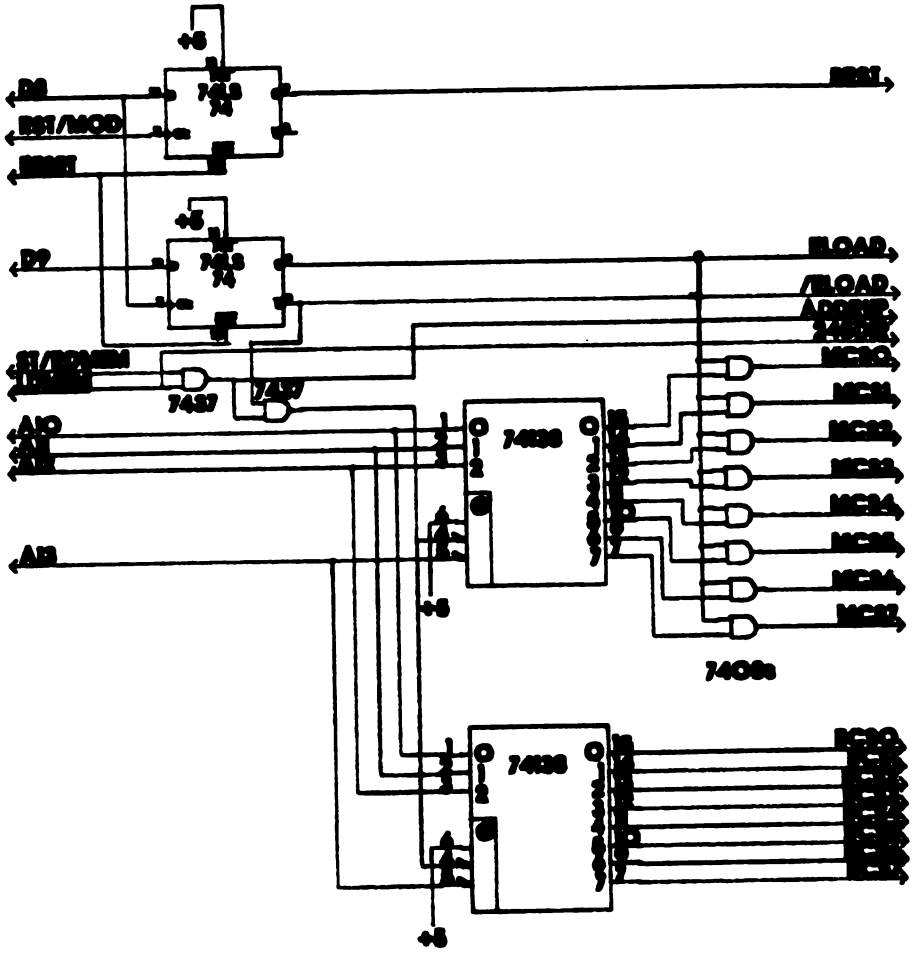


Figure 4.3 Schematic of chip-select generation for random-access memory.

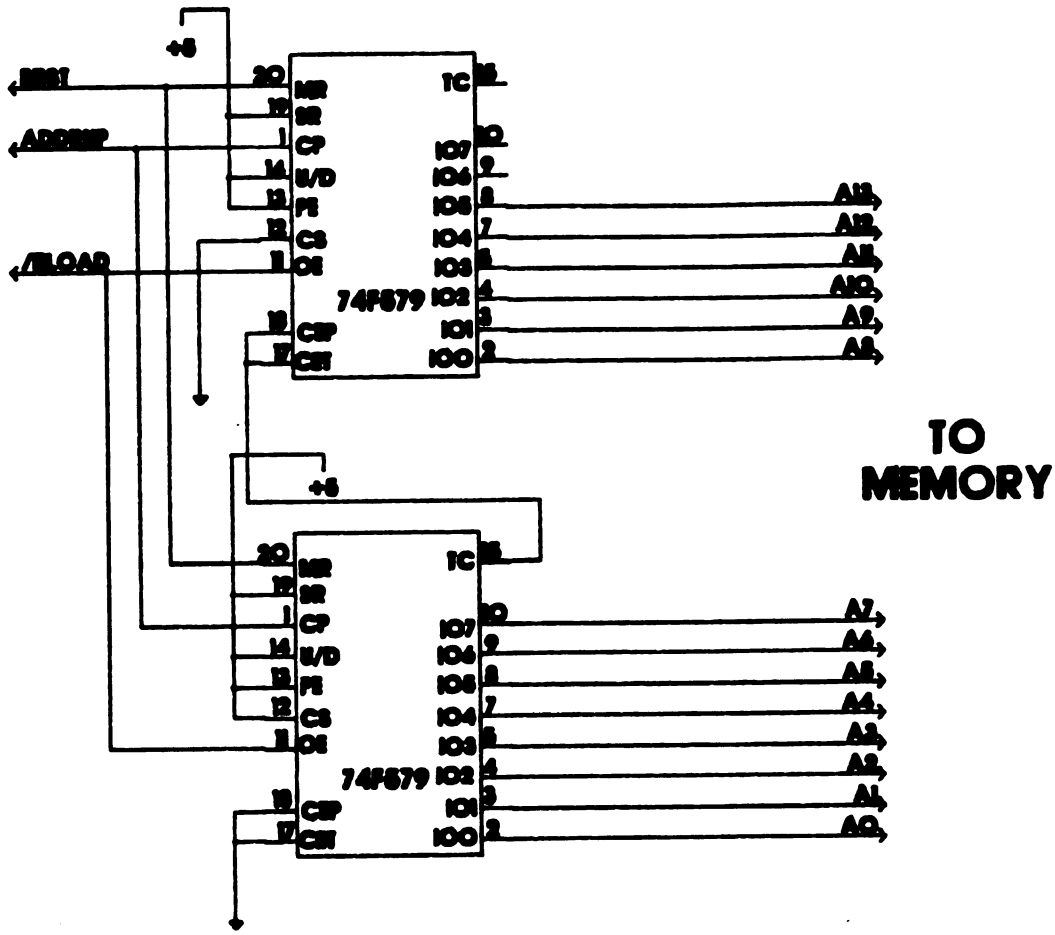


Figure 4.4 Schematic of the autoloading feature for generation of addresses when reading from or writing to the random-access memory.

control signals (bits in the control word) available simultaneously. The selection of the appropriate control word occurs through the next-address bits of the current control word in the microcode, as well as the current state of the signal selected by the 16-input multiplexer. Since the output of the multiplexer is always the least significant address bit, whether a branch is desired or not, memory must be allocated by pairs in software. If no branch is desired, then the same control word occupies both locations allowing for correct execution upon any value of the multiplexer's output. This next-address generation scheme is shown in Figure 4.5.

The design of one byte in the microcode RAM is shown in Figure 4.6. Each byte includes two 2148H random access memory chips with 4 by 1024 bit storage, a 74LS245 bidirectional transceiver for reading and verifying the bytes in each control word, a 74LS374 octal latch for latching the current control signals for each clock cycle. The 2148H was chosen for its speed (20 nS access time) and low cost. PFA-1000A contains eight such bytes, providing the 64-bit control word. Each bit in the 64-bit microcode is a direct control signal either for the next-address generation, one of the integrated circuits on the intensity or x-value buses, or one of the counters or flipflops.

The design of the intensity bus is shown in Figure 4.7. Basically, incoming data are latched by a bank of six 74LS374s (4 for the 32-bit intensity and 2 for the 16-bit x-

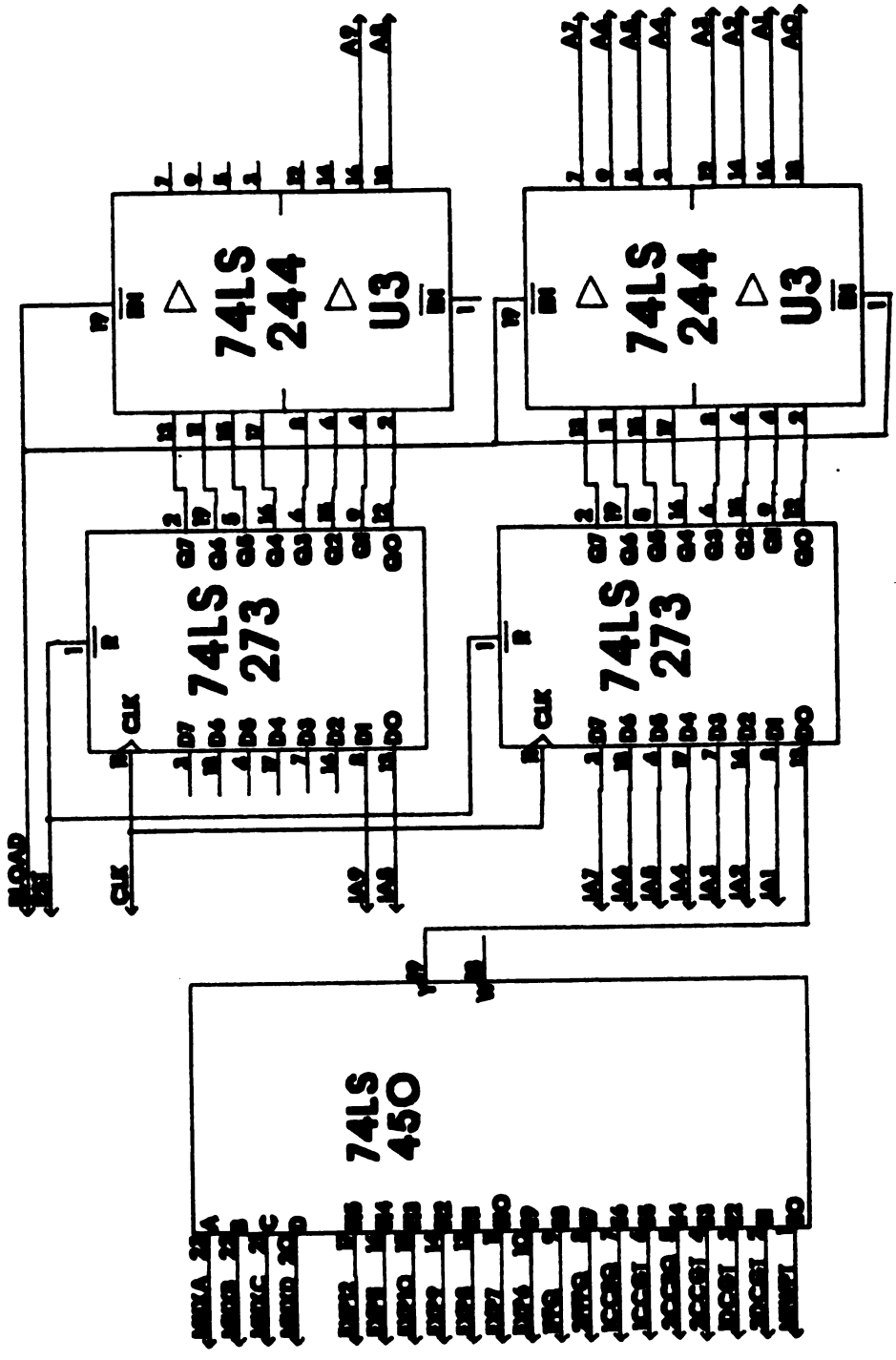


Figure 4.5 Schematic of the next-address generation circuitry when executing the peak-finding algorithm.

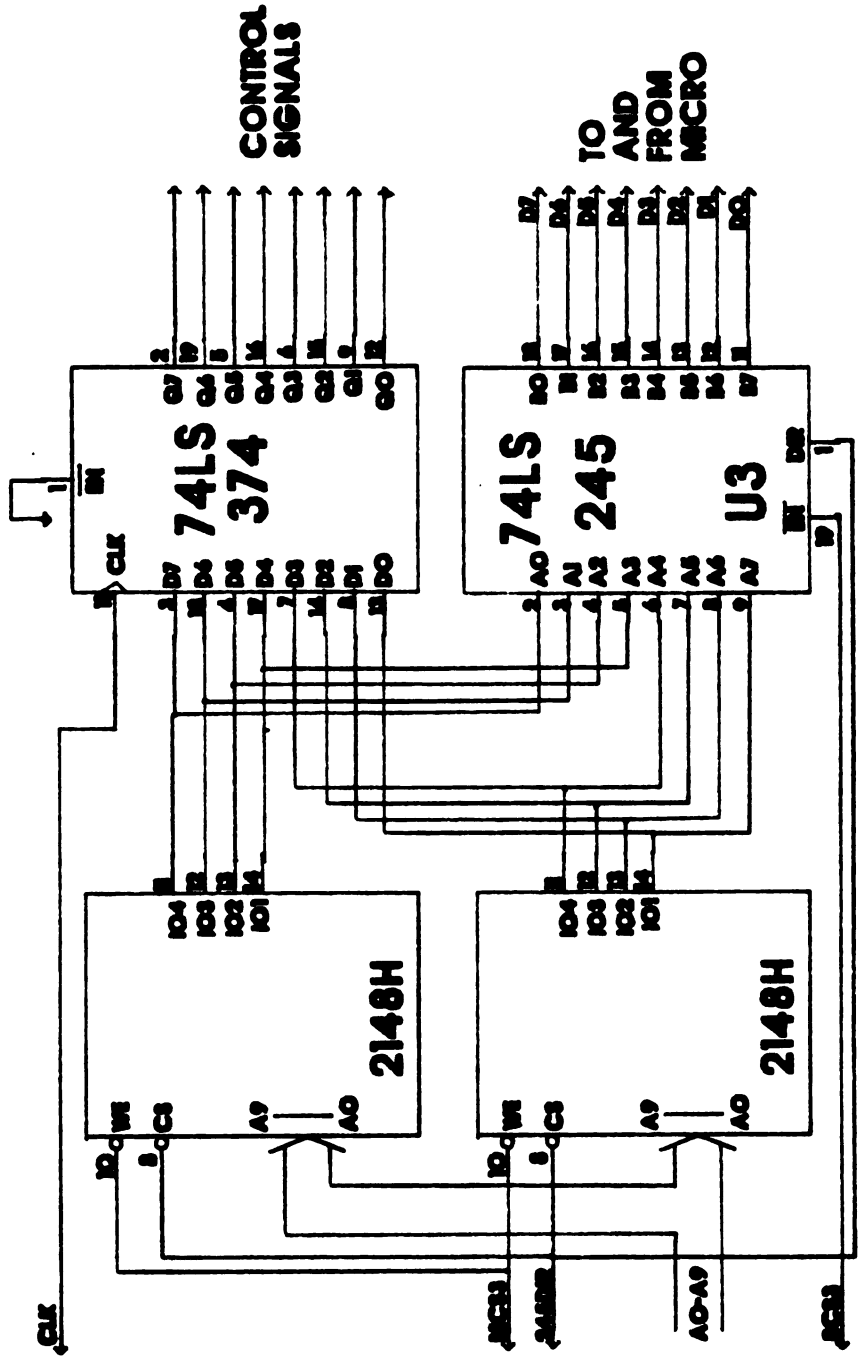


Figure 4.6 Schematic of one byte of random access memory with circuitry for access from the host computer and latching control signals for each clock cycle.

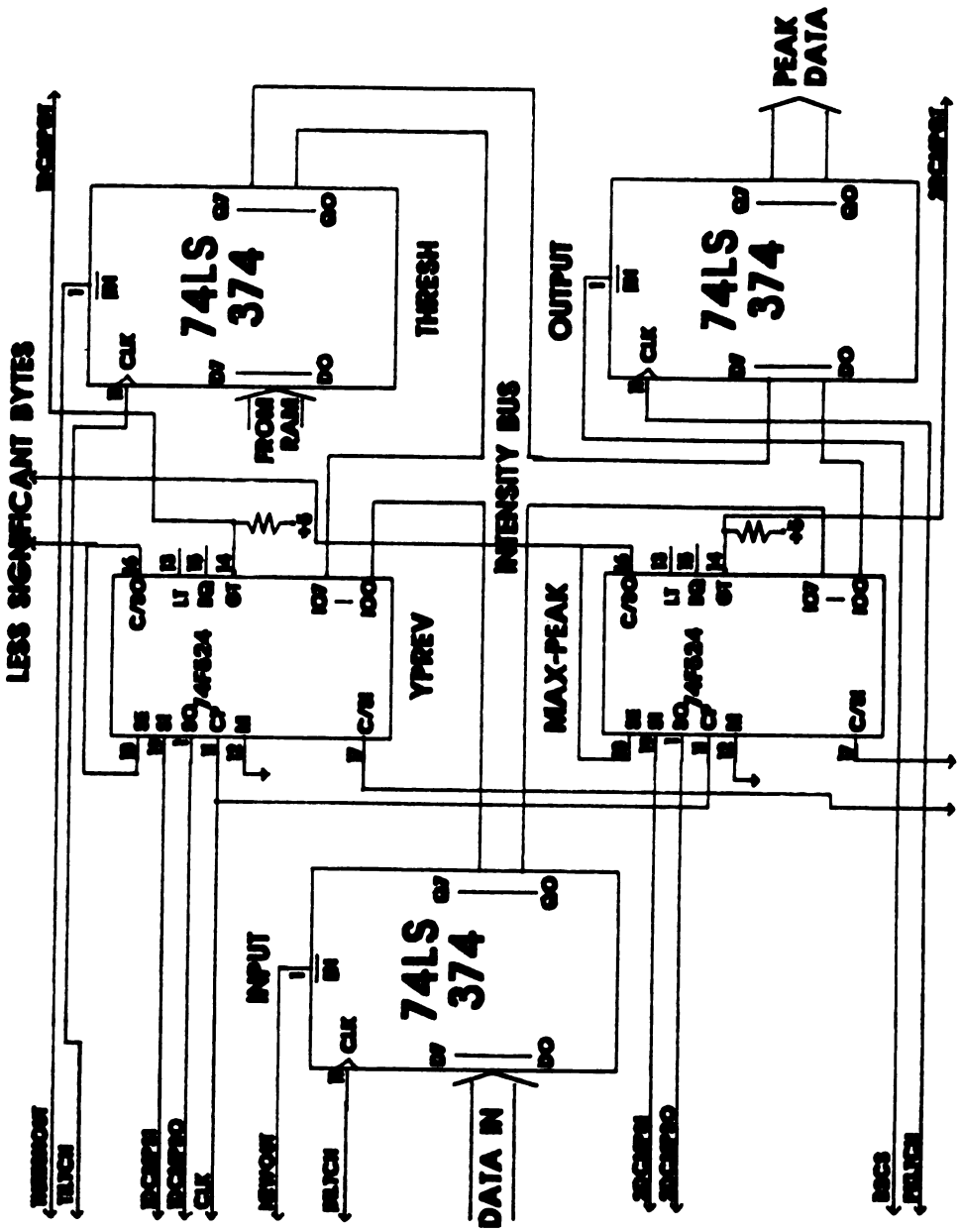


Figure 4.7 Schematic of the intensity bus.

value). There are also latches for a threshold value and output (peak) data. Two banks of four 74F524 8-bit registered comparators allow storage of the present maximum intensity and the previous intensity, as well as comparing these values with data on the intensity bus. Thus, the previous intensity value can be compared with the present intensity by driving the present intensity onto the bus from its latches and monitoring the status lines from the appropriate bank of registered comparators. In another example, the maximum intensity can be compared against threshold by transferring the threshold value onto the bus from its latches and monitoring the status lines from the 74F524s which contain the maximum intensity in their registers.

3. The Microcode Compiler

Design Goals

An algorithm microcode compiler was written to allow inexperienced programmers to write and test new algorithms for the peak-finding accelerator without extensive knowledge of either the intricacies of the peak-finder's electronics or of writing horizontal microcode. With this goal in mind, I sought to use the inherent extensibility and flexibility of FORTH to create a syntax for the compiler that was as close to FORTH itself as possible. In essence, the syntax should create a microcode compilation that is analogous to

the familiar resident FORTH compilation, including the typical control structures. Appendix F contains a full listing of the FORTH code for the microcode compiler.

I wanted the compiler to be easily modified, so further improvements to the hardware would not outdate the compiler. This flexibility in the compiler also allows it to be modified for completely different applications which require horizontal microcode. In order to achieve this ease of modification, the microcode compiler needed to be flexible in several ways.

First, the compiler needed to be flexible with respect to word length. The peak-finding accelerator described in this chapter uses a 64-bit word, but this compiler attribute could be modified for a completely different environment or for the same environment after the addition or deletion of hardware functions. This is accomplished by changing the value of the constant WORDLENGTH in block 2 to the new word length (in bits). The new value must be a multiple of 8, but is otherwise limited only by the requirement that the buffer in which the compiler constructs the microcode not exceed the amount of memory available on the host system. This restriction that WORDLENGTH be a multiple of 8 is merely a function of the organization of memory on the host processor. Most processors move data in multiples of 8 bits (e. g., 8,16,32-bit processors). Unused bits at the end of a control word need not actually exist in the hardware,

since nonexistent bits in the peak-finding RAM will not be latched or used.

Secondly, the compiler needed to be flexible with respect to grouping of the bits into control fields (groups of signals that control related devices or accomplish related tasks) and with respect to the definition of each bit in the control field. The flexibility allows the compiler to be easily modified after any hardware changes to the peak-finder or for a completely new microcode environment.

Thirdly, the compiler needed to be flexible with respect to the depth of the program space. The RAM used in the hardware peak-finder is 1024 words deep, but other environments might require more extensive program space. Again, the value for the depth of the program space is limited only by the memory available for the microcode buffer on the host system and can be changed by changing the value of the constant RAM-DEPTH in block 1.

Fourthly, flexibility with respect to the control structures would also be desired. For instance, additional signal multiplexers could add the capability make a decision (jump) based on two conditions simultaneously. One would then like the compiler to take advantage of this capability. Unfortunately, this would necessitate a departure from the common IF . . . THEN syntax employed here. A syntax similar to 0=IF, 1=IF, 2=IF, 3=IF, . . . THEN would be needed in that case to respond to the four possible conditions. The

modifications to the present compiler to allow that feature would not be too difficult, but, for now, I have chosen familiarity rather than unnecessary flexibility.

The Software

The heart of the microcode compiler is the word ENCODE found in block 5. ENCODE is a second-order defining word, that is, a word which defines new defining words. ENCODE defines words for each control field, such as MUX.SELECT and 1FLAG (also defining words), assigning to each control field a length in bits and a location in each control word. These words, in turn, define the various instructions for their designated control field. MUX.SELECT, for instance, defines instructions which will compile the appropriate bits for controlling the input to the 16-channel multiplexer into the control word currently being compiled. 1FLAG defines words which will compile the appropriate bits for one of the peak-finder's flipflops into the control word currently being compiled. Thus, 1FLAG can define 1CLR, which will compile the appropriate bits for clearing the first flipflop when the control word is executed, 1SET, which will compile the bits for setting the flipflop upon execution, and 1HLD, which compiles the bits which leave the flipflop unchanged upon execution. The activity initiated by these instructions (1CLR, 1SET, 1HLD, etc.) is to compile the appropriate bits into the microcode buffer in the designated

instruction field of the control word currently being compiled.

These primary instructions can themselves be linked by the typical FORTH colon (:) definitions into a data movement, which can represent an entire control word or a logically linked part of a control word. Furthermore, sequences of control words can be linked by colon definitions into macros. Movements and macros can be used for programming convenience or to give an instructions and control words new mnemonic content. In order to link control words into useful macros, though, one also needs control structures to direct the sequence of the compilation of the control words.

Three variables, "current," "node," and "available" control the sequence of compilation of control words into the microcode buffer. The variable "current" holds the number of the current control word being compiled and directs compilation of instruction fields into the appropriate location in the microcode buffer. The variable "available" holds the number of the next unused location in the buffer. Locations for control words are allocated two at a time: one for the control word which will be the subject of a branch on zero and one for the control word which will be the subject of a branch on one. If no branching is occurring, these two instructions are the same. The variable "node" comprises three bytes. The first two bytes are the number of the most recently compiled control

word which contained a branch instruction, while the third byte signals whether the control word currently being compiled is the subject of a branch or not. If a branch instruction occurs, the compiler first compiles the control word which will be the subject of the branch if the result is one (the selected multiplexed signal is high). The compiler does this by placing the address of this control word into the next-address field of the control word from which the branch is occurring (the node) and onto the parameter stack to provide the reference location for later compilation of the corresponding branch-on-zero instruction. If no branch instruction occurred in the control word compiled last, the address of the current instruction should be compiled into the next-address field of the previous instructions.

The variables "current," "available," and "node" are manipulated by the control structures "if," "else," "then" and "next." These control structures retain the same meaning as in FORTH itself. They are preceded by a condition test (a MUX.SELECT instruction field), which upon execution will generate either a one or a zero, which is the value of the least significant address bit. The word "if" signals that the following control word will be the subject of a branch upon a result of one; "else" signals that it will be the subject of a branch upon a result of zero. The word "next" is the control structure for linear program flow (no branching). Therefore, the control word following "next" is

not the subject of a branch. The word "next" must be used inside a colon definition; the corresponding word for interpretive mode is "<next>." All of the other control structures can be used in either compile or interpretive mode.

The word START precedes the control word that will start the repetitive part of the algorithm. Thus, one can have a set-up portion of microcode (to initialize devices) and a run-time portion (to actually find peaks). HOME forces a return to the address of START after execution of the previous control word and should, therefore, follow the last instruction in each branch of the algorithm. This command restarts the algorithm for processing the next datum. All of the other control structures can be used in either mode. The word MICROCODE initializes the compiler and should precede compilation of the algorithm.

Error-checking routines make sure that only one component is driving each bus in order to eliminate bus conflicts. If any potential conflicts are found, error messages are generated which indicate the suspect instruction and bus. The assembly level code words ?BIT and ?2BITS return the value of a single bit or two contiguous bits respectively, given a byte address and a bit number. MCHECK uses these assembly level routines to perform this error checking.

MCODE>DISK moves the entire code-buffer to a file on disk for later use. The file starts at the block designated

in CODE-FILE, a constant whose value can, of course, be changed. Block 25 in Appendix F contains some debugging and listing aids. SHOW types out a given microinstruction. CLEAR erases a given microinstruction. REV gives the current state of all control structure variables. "list" types out the first n microinstructions.

Algorithms can be compiled interactively, as long as "<next>" is used, or formulated inside a definition using "next." ALGO in block 26 in Appendix F is one such word, which defines a sample peak-finding algorithm (6). ALGO uses macros written in blocks 15-18. Executing UCODE will execute the word ALGO, thus, compiling the algorithm into the microcode buffer. If no errors are found, UCODE will also write the resulting microcode to disk.

4. Results

Processing Speed

With the current peak-finding algorithm, the hardware peak-finder is capable of processing an average of one datum per microsecond. Theoretically, then, the peak-finder could process 1 million points per second. However, the primary limitation to the speed of peak-finding is actually transferring the data from the data acquisition circuit to the hardware peak-finder. Of course, the speed of the peak-finder makes almost any operation seem slow, but there are two reasons why the transferal of intensity data is slower

than might be expected. First, the intensity values are only available as bytes. The peak-finder, on the other hand, can accept values either as 16-bit words or as bytes. Thus, writing data as bytes to the peak-finder is twice as slow as would be possible writing words. Second, reading data from the data acquisition circuit is the slowest transaction on the Novix Mass Spectrometer Interface Board (described in Chapter 2). On the other hand, when the peak-finder is operated in byte mode, it is not necessary to mask off the high byte when reading from the data acquisition circuit and to combine the two low bytes into one 16-bit word, since the high byte is simply not latched. This adds to the speed advantage of hardware peak finding over software peak-finding for the Novix Control System.

Peak-finding speed could be further enhanced by using the flexibility designed into the hardware peak-finder to create an autoloading feature which would control and accept output from the data acquisition circuit. The autoloader would generate the control signals and monitor the "acquisition-done" interrupt from the data acquisition board, reading the data at hardware speeds. The large amount of time spent on software transferal of data between these two circuits would be eliminated. The drawback to this approach is that it eliminates the possibility of other kinds of software processing between acquisition and peak-processing like dual-mode acquisition, described in Chapter 5.

Improvements in Scan Speed and Signal Averaging

The addition of hardware peak-finding to the Novix TQMS control system has made possible the scanning speeds and concomitant amounts of averaging shown in Table 4.1. This can be compared to the scan speeds and amounts of averaging before adding hardware peak-finding in Table 4.2. A new scan speed of 4000 amu/S is now possible. However, this scan speed contains no rate synchronization step and may be erratic from point to point. Furthermore, the problems with the 2000 amu/S rate described in Chapter 2 will only be exacerbated at 4000 amu/S. This scan rate has been included for completeness, but its utility is questionable. Note that the scan rates for hardware peak-finding produce only reduced spectra, so that no peak profiles can be obtained.

The lower scan rates, in which software peak-finding consumed as much as 36% of the time spent at each point, can now include much more signal averaging with hardware peak-finding. Rate 2 (2000 amu/S) has 8 times more averaging. Rate 3 (1000 amu/S) has twice the amount of averaging. By rate 6, the amounts of averaging are equal. As the scan speed decreases, more and more of the time spent at each point is spent signal averaging, whether peak-finding is performed in software or hardware. The time saved by performing peak-finding in hardware is not enough to allow twice as much averaging at these lower rates (the number of

averages must be a factor of two, as described in Chapter 2). Of course, the improvement in the number of averages performed at a given scan rate with hardware peak-finding would be much more dramatic with slower processors.

Table 4.1
Novix Control System Scanning Functions
With Hardware Peak-Finder

Rate	Pts/S	Amu/S	Time/Pt	# Averages/Pt
1	40000	4000	25 uS	1
2	20000	2000	50	8
3	10000	1000	100	32
4	5000	500	200	64
5	2500	250	400	128
6	1000	100	1 mS	256
7	500	50	2	512
8	250	25	4	1024
9	100	10	10	2048
10	50	5	20	4096
11	25	2.5	40	4096
12	10	1	100	4096
13	5	0.5	200	4096
14	2.5	0.25	400	4096

Table 4.2
Novix Control System Scanning Functions
Without Hardware Peak-Finder

Rate	Pts/S	Amu/S	Time/Pt	# Averages/Pt
2	20000	2000	50 uS	1
3	10000	1000	100	16
4	5000	500	200	32
5	2500	250	400	64
6	1000	100	1 mS	256
7	500	50	2	512
8	250	25	4	1024
9	100	10	10	2048
10	50	5	20	4096
11	25	2.5	40	4096
12	10	1	100	4096
13	5	0.5	200	4096
14	2.5	0.25	400	4096

5. Conclusion

The hardware peak-finder has made possible faster scan rates and, more importantly, greater signal averaging at fast scan rates. Because of the limitation that the number of averages be a factor of two, hardware peak-finding is only important at scan rates faster than 250 amu/S. The hardware peak-finder accomplishes this speed in peak finding and still allows flexibility in the choice of algorithms through software sequencing.

The algorithm compiler has made the electrically complex hardware peak-finding accelerator accessible to the average scientist. We have used the extensibility and interactive nature of FORTH to make the tedious chore of creating 64-bit horizontal microcode into a familiar programming environment with typical control structures and inclusive error-checking.

The hardware peak-finder allows a function once run in software at a typical speed of 100 uS/point to now be executed in hardware at 1 microsecond/point. The limitation in scanning speed is no longer reliable peak-finding, but in data acquisition and transferal to the peak-finder.

REFERENCES

1. Intel Corp., Santa Clara, CA.

2. C. A. Myerholtz, A. J. Schubert, M. J. Kristo, and C. G. Enke, Intelligent Instruments and Computers, 3, 1985, 11.

3. C. A. Myerholtz, A. J. Schubert, M. J. Kristo, and C. G. Enke, Intelligent Instruments and Computers, 3, 1985, 13.

4. Forth, Inc., Hermosa Beach, CA.

5. J. H. Golden, C. H. Moore, and L. Brodie, Electronic Design, March 21, 1985.

6. M. J. Kristo and C. G. Enke, in preparation.

CHAPTER FIVE

DUAL-MODE DETECTION FOR HIGH PERFORMANCE ION CURRENT MEASUREMENT AND EXTENDED DYNAMIC RANGE IN MS/MS

1. Introduction

Kondrat and Cooks (1) first reported the increase in useful dynamic range obtained in tandem mass spectrometry (MS/MS). This is due to the great decrease in chemical noise in the system, resulting from the addition of the fragmentation step and the second mass analyzer. However, very little work has been done since that observation to capitalize on that increased dynamic range or even to make it available in a single scan. Figure 5.1 compares the dynamic range afforded by various data acquisition systems with the dynamic range potentally available in MS/MS systems. The typical dynamic range for an MS/MS instrument extends from the limits of chemical noise in the system (usually less than 1 count per second) to the saturation point of the multiplier (greater than 10^9 counts per second). Of course, with analog measurements, the multiplier voltage or preamplifier gain can be changed to measure larger or smaller ion flux rates, but this does not change the dynamic range of ion flux measurable at any given setting.

In this work, a data acquisition system has been implemented which takes advantage of the full dynamic range of MS/MS by using the Galileo Dual Output Channeltron (2).

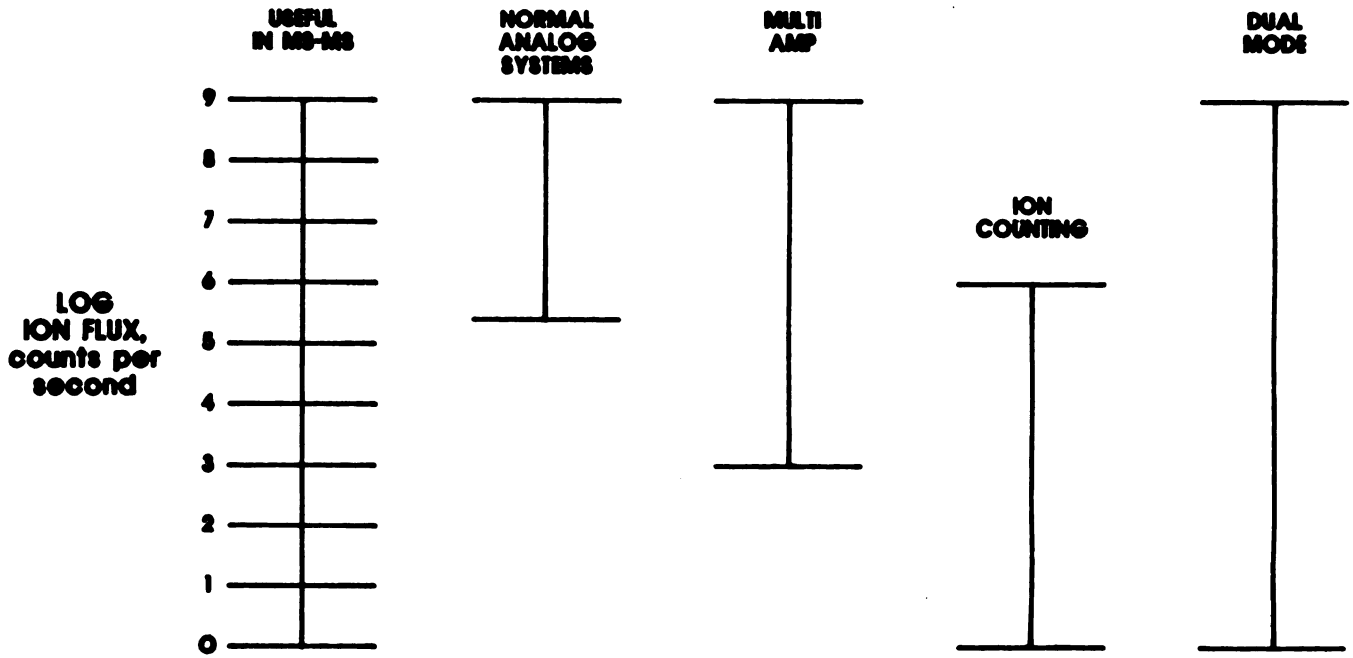


Figure 5.1 A comparison of the dynamic ranges afforded by various systems versus the dynamic range available in MS/MS: the basic analog amplifier/ADC available on most systems, the multirange analog amplifier/ADC available from the multiamp circuit, ion counting circuits, and the dual-mode control system.

Kurz and Roy first described the dual-mode detector, which is capable supporting simultaneous analog measurements and pulse-counting, to the mass spectrometric community in 1979 (3). Subsequent applications using the dual-mode detector were described by Schoen (4) and Matthews (5). Basically, the dual-mode detector (see Figure 5.2) consists of two continuous dynode multipliers in series, separated by the analog anode, ground isolation grid, and protection grid. Ions incident upon the detector or its conversion dynode release secondary electrons. These electrons undergo further amplification in the low-gain section to a total gain of 10^4 (with the recommended -1400 V applied). The analog anode then collects 90% of the electrons to provide the analog signal. The remaining 10% of the electrons are free to enter the high-gain section of the dual-mode detector if the protection grid is held at common potential. This electron flux will then undergo further amplification to a overall gain of 10^8 (with +1900 V on the rear section and -1400 V on the forward section). Pulse-counting can then be performed with the signal at the anode of the high-gain section. However, a voltage of -350 V can be applied to the protection grid to prevent the electrons from entering the high-gain section. This grid protects the high-gain section under conditions of high input ion flux.

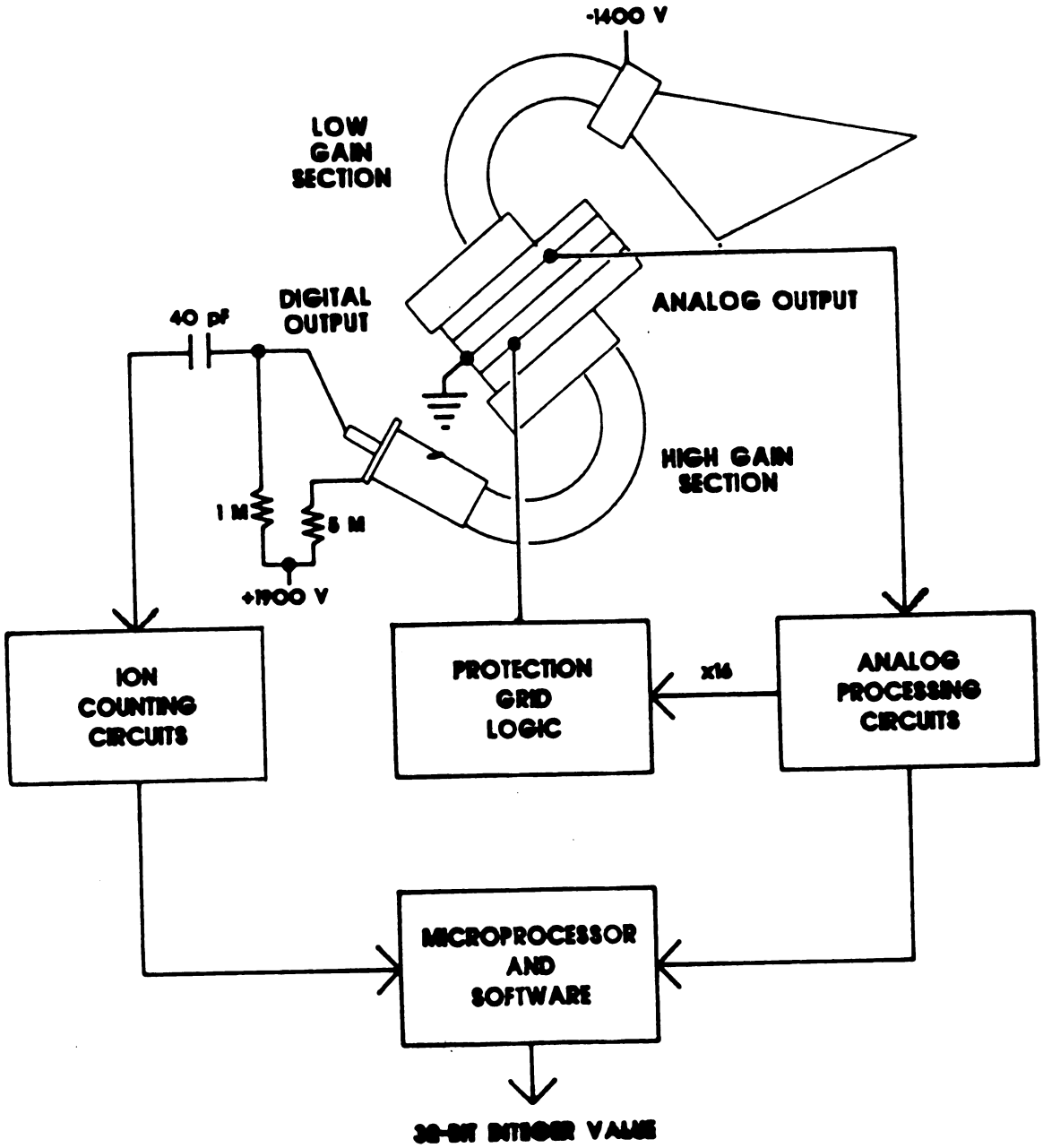


Figure 5.2 Schematic diagram of the Galileo Dual Output Channeltron control system.

2. System Description

To take advantage of all the dual-mode detector's capabilities, the control system must include analog signal processing, pulse-counting, logic for controlling the protection grid, and normalization of the analog and pulse-counting data into a single number proportional to the absolute count rate. The interconnection of the various subsystems involved in these operations is shown in Figure 5.2.

Analog Processing

For analog processing, I have chosen the multiamp analog measurement scheme shown in Figure 5.3 (6). In this scheme, current from the analog anode is converted to a voltage by a preamplifier with a transresistance of 10^7 V/A. A multiamp circuit amplifies this signal at five different levels of gain simultaneously. In the selection logic circuit, the outputs of all of these amplifiers are individually compared with the full-scale voltages of the ADC. The largest signal which does not exceed the ADC range is selected. The selection logic circuit also provides three range bits which indicate the binary value of the log to the base two of the gain for the selected signal. A data acquisition circuit combines the range signal with the output of the 12-bit analog-to-digital converter to form a 20-bit integer intensity value. The data acquisition

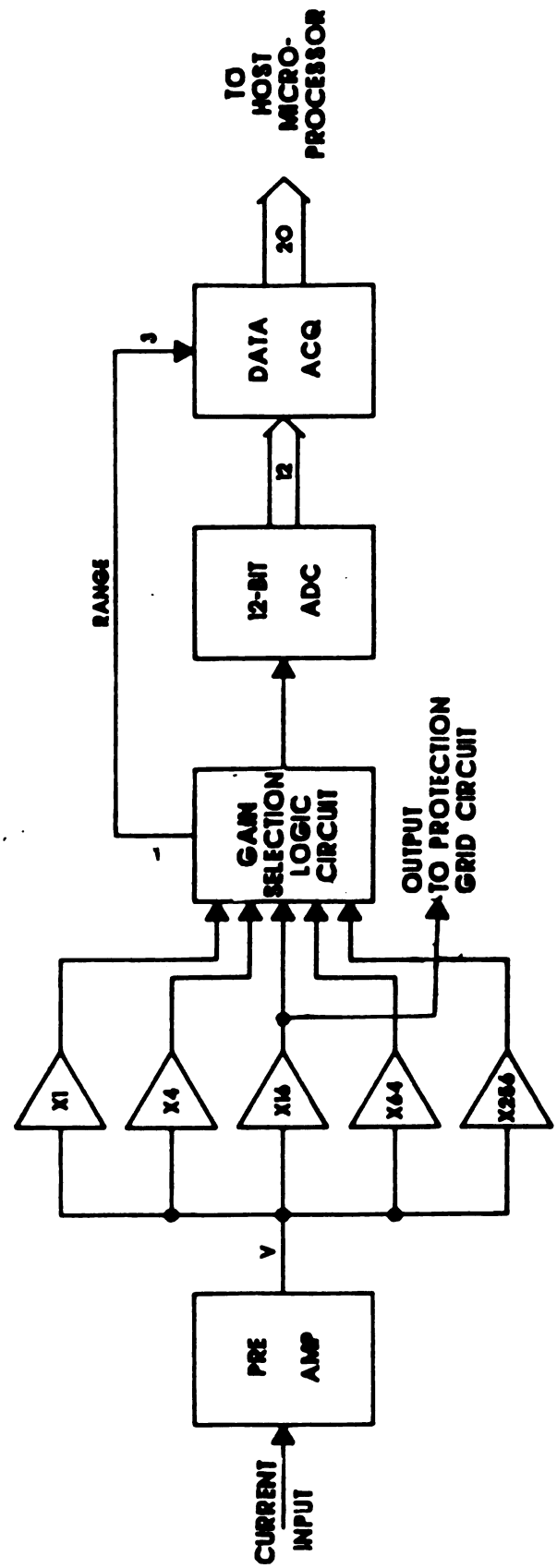


Figure 5.3 Multiamp analog measurement scheme.

circuit also sums a user-specified number of successive conversions and sends a 20-bit average to the host processor. The maximum data rate for the 20-bit average is 4 uS per conversion (e.g., for 16 conversions per datum, the data acquisition board takes 64 uS.)

Ion Counting

Current pulses from the pulse-counting anode are converted to TTL logic pulses by a pulse amplifier discriminator (Princeton Applied Research Model 1182) (7). The pulse amplifier discriminator (PAD) has user-adjustable output pulse widths from 10 to 75 nS (which affects the maximum possible count rate) and threshold levels from 150 to 500 uV, which can optimize the PAD sensitivity and immunity to noise for a given gain in the detector. The PAD can operate in current input as well as voltage input mode.

The pulses from the PAD are then counted by a versatile counter/timer circuit based on the American Micro Devices 9513 Counter/Timer Chip (8). The AMD 9513 chip contains five 16-bit counters which can be concatenated and individually configured. The pulse counter/timer circuit can measure pulse rates in either constant time (constant scan rate) or constant pulse count (constant precision) mode. Although the counting rate specification for the AMD9513 is only 7 MHz, experience in our laboratory has shown most chips capable of counting uniform pulse rates over 9 MHz with an external 10 MHz clock. The circuit also

senses when the pulse rate is insignificant (user-defined) and interrupts the processor, so the scan can continue. This is important for two reasons: 1) if the system is in constant count mode, an input pulse rate of 0 counts per second will force the system to wait indefinitely, and 2) one study has shown that up to 70% of the time spent in data acquisition is spent acquiring data at points which contain only negative ("nothing there") information (9). This interrupt feature senses the absence of useful ion current and notifies the processor to continue the scan. Thus, time spent acquiring data can be more efficiently utilized.

Figure 5.4 shows how the AMD 9513 counters are configured by the software. Counter 5 provides the counting window for the counters 1 and 2 which are concatenated into one 32-bit counter. If the counter is operating in constant time mode, then counter 5 counts clock pulses. The clock pulses are provided by any one of 5 internal frequencies, derived by division of the external 10-MHz frequency. The output of counter 5, which is only low during the count, provides a time window (gate) for counters 1 and 2 to count ion pulses. If the counter is operating in constant count mode, then counter 5 counts the ion pulse train, which can be appropriately divided when selecting a number of ion counts for the counting window larger than 16 bits. Counters 3 and 4 count down from user-set numbers, toggling their output when the count reaches zero. Counter 3 counts ion pulses and counter 4 counts clock pulses. The logical

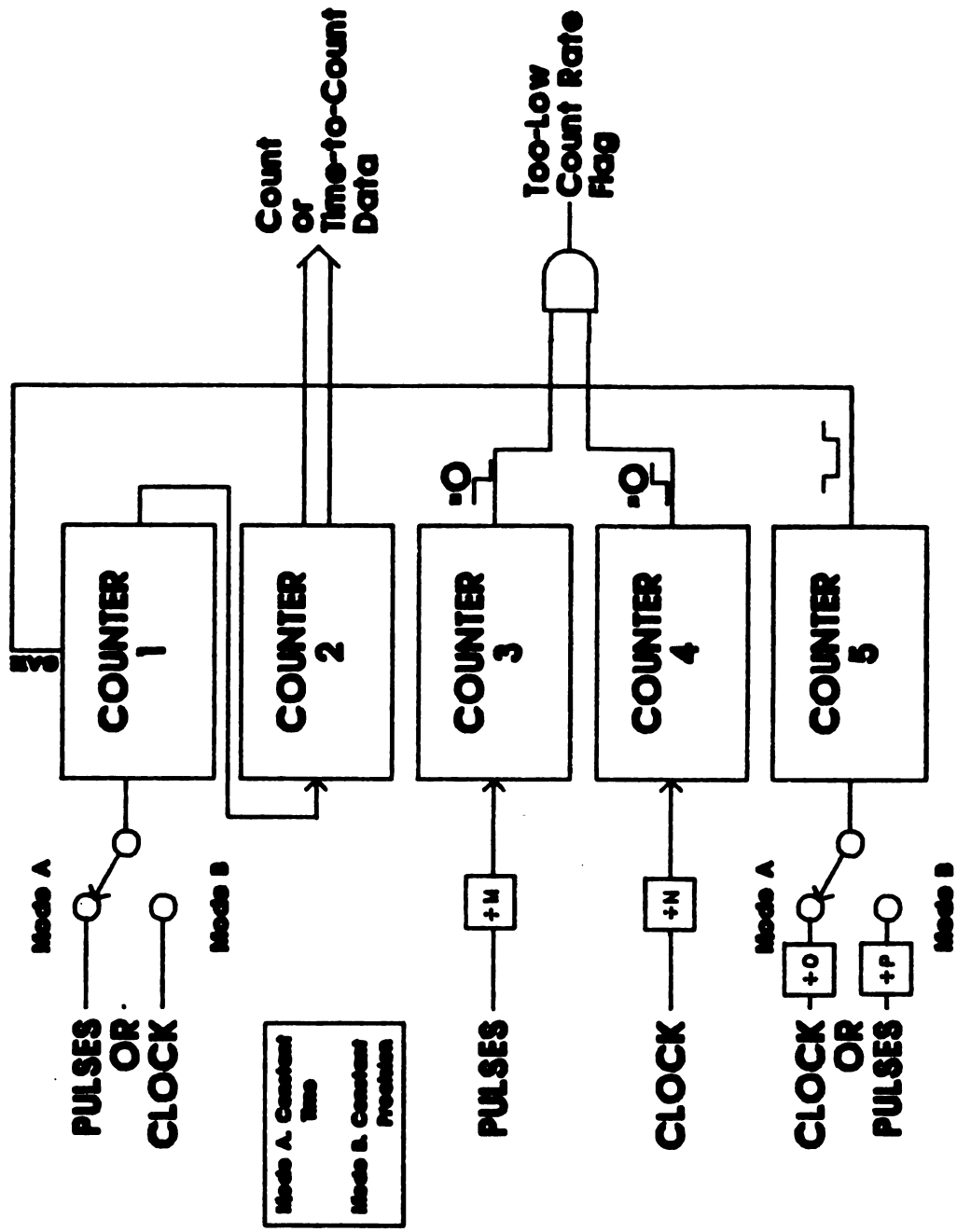


Figure 5.4 Configuration of the AMD9513 pulse counter.

AND function of the two outputs provides a signal which goes high if the time interval provided by counter 4 has elapsed without counter 3 having counted the required number of pulses. The software needed to configure the 9513 in this way can be found in Appendix G.

Protection Grid Logic

Figure 5.5 shows the schematic of the protection grid logic (10). This circuit provides two important functions. First, if the 16X line of the multiamp board exceeds a user-set voltage (currently +1.6 V, which corresponds to an ion current of 10 nA at the analog anode or approximately 3 million ions per second), the protection grid is held at -350 V. Otherwise, the protection grid is held at common potential. Secondly, when the protection grid is on, it holds the Q output of the 74LS74 flipflop low. Thus, the processor can set, and then read this flipflop to determine whether or not the protection grid is enabled and thus, whether to acquire analog or pulse-counting data. The asynchronous nature of this part of the circuit ensures that, if there is any doubt about the protection status, analog data (which is continuously available) will be acquired. It takes the protection grid circuitry about 2 uS to turn on the grid and 30 uS to turn it off. During data acquisition in most systems, no additional "dead time" for settling of the protection grid voltage is needed since additional processor tasks take at least 30 uS. For very

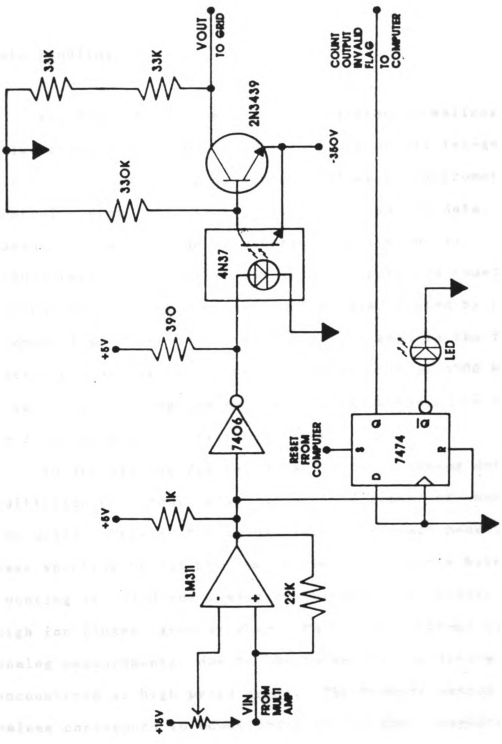


Figure 5.5 Schematic of the protection grid logic.

high performance systems, however, such time considerations may become significant.

Data Handling

Finally, the dual-mode control system normalizes the pulse-counting and analog data into one 32-bit integer value for further processing by the overall mass spectrometric control system. Therefore, all pulse-counting data, whether taken in constant time or constant count mode, is transformed into a 32-bit integer flux rate (in counts per second) by dividing the number of pulses counted by the elapsed time. Although the 32 bits assigned to the flux rate could accommodate a count rate as high as 4300 MHz (32 bits), the counting system only counts random pulse rates up to 3 MHz as will be described later.

To achieve the desired normalization, analog data are multiplied by a conversion factor (the number of counts per ADC unit). This factor is determined for each peak in the mass spectrum by calibration in the region where both ion-counting is valid and analog measurements are significant. High ion fluxes (greater than 3 MHz in our system) require analog measurements, due to the pulse overlap errors encountered at high pulse rates. The highest analog output values correspond to count rates of 1-2 GHz, depending on the analog gains in the system (analog multiplier gain and preamplifier gain). Thus, ion-counting data require about

22 bits to be represented digitally, while representing the data in integer form from both outputs requires 29-32 bits, again depending on the analog gains. Simplicity of software requires that values be stored in multiples of 8 bits, so a 32-bit integer intensity value is used.

Dual-mode Scanning

In our system, under actual operating conditions, the instrument control computer can acquire ion intensity data using the dual-mode control system hardware and software in a dual-mode scan sequence. After the quadrupole mass filter and lens voltages have been updated, the dual-mode control system sets, and then checks the protection grid flipflop. If the flipflop is cleared, it acquires and formats an analog value. Otherwise, ion-counting data are acquired and corrected for ion pulse overlap. If pulse-counting is valid and the analog intensity is significant, then the analog calibration factor (ion counts per ADC unit) is calculated for the current peak in the mass spectrum. Then the resulting 32-bit intensity can be stored or processed further, for example to obtain peak heights, areas, or positions. The next set of quadrupole and lens voltages can be applied and the scan continues.

Dual-Mode Software

The user can invoke dual-mode acquisition at any time with the command DUAL-MODE. This command brings up a menu

which steps the user through the various choices available in dual-mode acquisition. First, the user chooses between counting ions in constant time or constant count mode. Then, the user can select the appropriate time or count interval from a menu. The user is further prompted to enter a time window (in microseconds) and a threshold for the number of counts to occur in that window to be considered a datum with ion count information significantly above the background count rate. The software then transfers this information to the Detection slave, where the counter/timer circuit is configured and the dual-mode sequence becomes the operative algorithm for data acquisition. The user can switch back to pure analog acquisition at any time by typing ANALOG. The dual-mode sequence software is found in Appendix H.

3. Pulse Overlap Correction and Analog Calibration

One would expect a straightforward correlation between the equations which describe the numerical output from the two sections of the dual-mode detector. If the counting system is fast enough to keep up with the incident flux, i. e., each count is a single ion event, the number obtained from the pulse-counting section will be equal to the actual incident ion flux (Φ_J)of those ions that convert into 1 electron or more. The current obtained from the analog

section is given by Equation 1, neglecting collection efficiencies;

$$i = e \Phi_j m(t,V) n(v,j) \qquad [1]$$

where e is the charge on an electron and m is the multiplier gain of the first section, which depends on the high voltage applied across the analog section of the multiplier and the history of the multiplier, and n is the average number of electrons emitted per incident ion, which depends on the ion velocity and the particular species striking the multiplier. The number obtained from the analog measurement, though, is the analog-to-digital conversion of the analog voltage output of the multiamp circuit. This voltage is a product of the input current, the transresistance of the preamplifier, and the gain of the selected voltage amplifier. The analog output is accompanied by a digital code for the amplifier gain. This leads to the generalized equation for the analog intensity given by Equations 2 and 3:

$$\text{ADC output} = iC/e \qquad [2]$$

$$\text{ADC output} = C \Phi_j m(t,V) n(v,j) \qquad [3]$$

where C is a constant that includes the preamplifier transresistance and the amplifier voltage per least significant bit, which are known.

As indicated in Equations 1 and 3, the gain of the analog section of the dual-mode detector (m) is a function

of the species striking the detector. Many studies have shown that the gain of an ion multiplier varies with the mass and the velocity of the incident particle, the number of constituent atoms in the incident particle, and the chemical nature of the constituent atoms (11-21). This is due principally to a variation in the average number of electrons produced when the ion collides with the first surface in the detector system. The use of conversion dynodes can reduce this species-dependence to some degree, but variations still occur (22-24). In order to know the absolute ion flux, then, frequent measurements would be necessary to maintain an up-to-date value for Cm(t,V)n(v,j). For each peak in the mass spectrum, at an ion flux in the region where both ion current and ion count outputs are active, the control system simultaneously measures the ion current (in ADC units) and the ion count (in incident ions per second). Since both values are obtained at the same incident ion flux, they allow the determination of the ion current produced by a given ion flux for the specific incident ion beam. Normalization between the two outputs (division of the ADC output by Cm(t,V)n(v,j)) takes place in software, but relies extensively on an Intel 8087 Numeric Coprocessor for timely calculations (25). All software was written in the programming language FORTH (26).

4. Results

The outputs from both sections of the dual-mode detector are plotted against each other for a wide range of ion fluxes for air (predominately a mixture of nitrogen and oxygen ions) in Figure 5.6. This curve represents a convolution of the current output of the detector and the counting characteristics of the pulse-counting system. Due to the Poisson distribution of arrival times for the incoming ions and the fixed resolution of the counting system, an increasing percentage of the incident ions are not counted as the ion flux increases. This is because multiple ions are counted as one event at the higher rates (27).

The percentage of pulses lost for a given, random input flux is well known from Poisson statistics ($PR_{exp} = PR_{act} * \exp(-PR_{act} * Res)$), where PR_{exp} is the experimentally measured ion count rate, PR_{act} is the actual ion count rate, and Res is the resolution window for the counting system. The resolution window can thus be shown to be $0.368/PR_{exp,max}$, where $PR_{exp,max}$ is the maximum experimentally measured count rate. The Poisson equation can be solved in reverse by successive approximations; however a simplified form of the equation, which is easier to solve in reverse, is given in Equation 3.

$$\% \text{ Pulses Lost} = 100 * \text{Pulse Rate} * \text{Resolution} \quad [3]$$

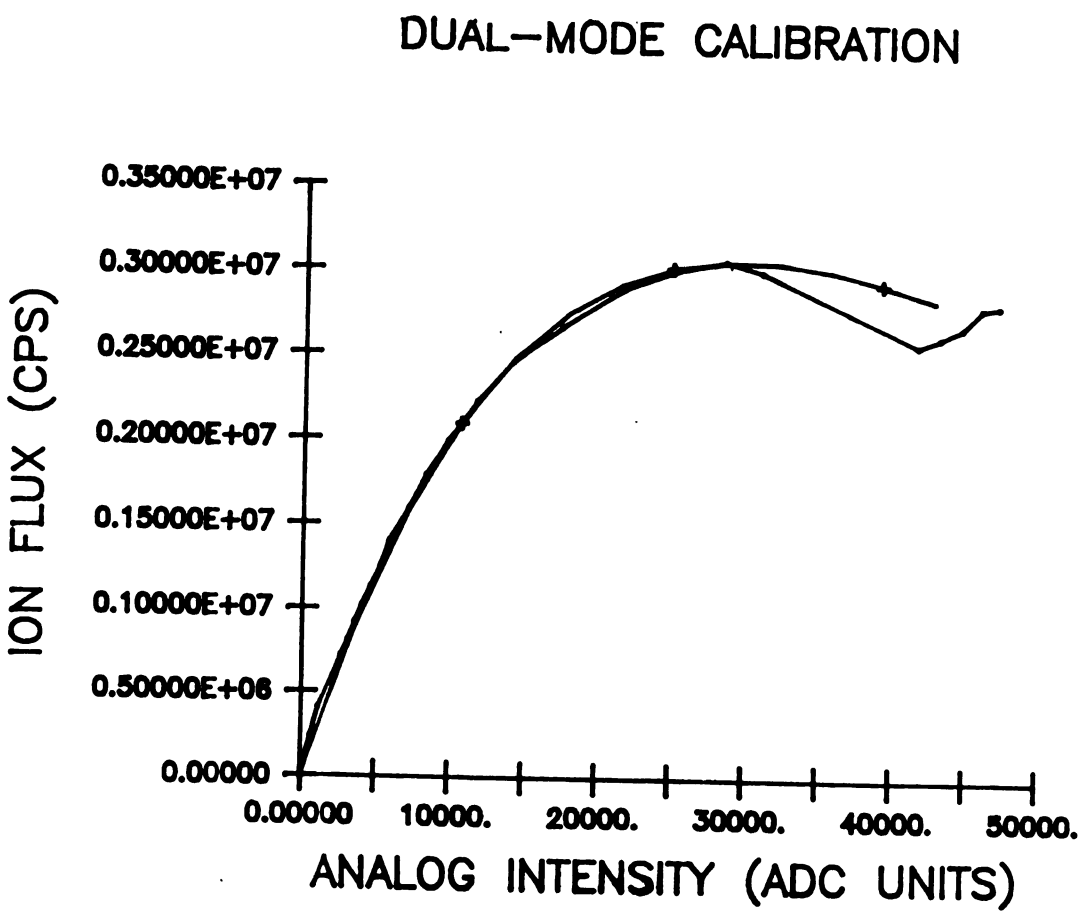


Figure 5.6 Raw ion current versus the corresponding analog intensity from the multiamp circuit (for air) and the theoretical curve (+) expected from the Poisson resolution window (120 nS) calculated from the experimental curve.

This simplified equation deviates significantly from the exponential equation at higher count rates, but is a good approximation at low count rates. In terms of the actual input pulse rate, the simplified equation deviates from the exact Poisson expression by less than 1% up to 1.2 MHz and by less than 10% up to 3.3 MHz. The value of 120 nS used as the resolution window for the theoretical curve in Figure 5.6 was found from the maximum count rate as shown above and was verified by successive approximations to create the best fit. Obviously, from the calculated value for the resolution window, the maximum count rate for the AMD9513 counter (9 MHz uniform pulse rate) is the main contributor to pulse overlap.

After calculating the value for the resolution window, the actual input count rate can be determined by solving in reverse the exact exponential equation using successive approximations. However, on the half of the parabola in Figure 5.6 representing low count rates, the actual input count rate can also be found from the experimentally determined count rate using Equation 4, which is derived from Equation 3.

$$PR_{act} = (1 - \sqrt{1 - 4 * PR_{exp} * Res}\ ) / 2 * Res \qquad [4]$$

The raw pulse-counting data, after having been reverse-fit to obtain the actual input pulse rate, PR_{act}, is linear with

respect to analog intensity to around 6 MHz (PR_{exp}=2.9 MHz) and provides a calibration factor of 210+/-4 ion counts per second/least significant bit of the multiamp output for the data curve (air) shown. Actually, only input ion count rates less than 3 MHz (PR_{exp}=2.1 MHz) are counted and reverse-fit during the dual-mode scan sequence in order to insure the precision of the reverse-fitting process. This linearity provides a readily obtained correlation between the analog and pulse-counting data obtained. Thus, in our control system all pulse-counting data are reverse-fit to obtain a corrected count. Analog data are then multiplied by the ion count/ion current calibration factor, which is equivalent to the slope of the calibration curve, to yield a value which corresponds to the input ion flux that would give such an analog reading (if such a measurement were possible).

Figures 5.7 through 5.10 show that real-time calculation of the count-current calibration factor is possible on a peak-by-peak basis. Figure 5.7 shows the mass spectral peak profiles for the nitrogen molecular ion (m/z=28) obtained from both the analog intensity (5.7A) and the corrected ion flux (5.7B). Plotting the corrected ion fluxes versus the analog intensity acquired at the same mass-to-charge ratio yields a straight line with a slope of 40 ions per ADC count (Figure 5.8). Figure 5.9 shows the mass spectral peak profiles for CF_3^+ (m/z=69) in the spectrum of perfluorokerosene (PFK). The plot of corrected

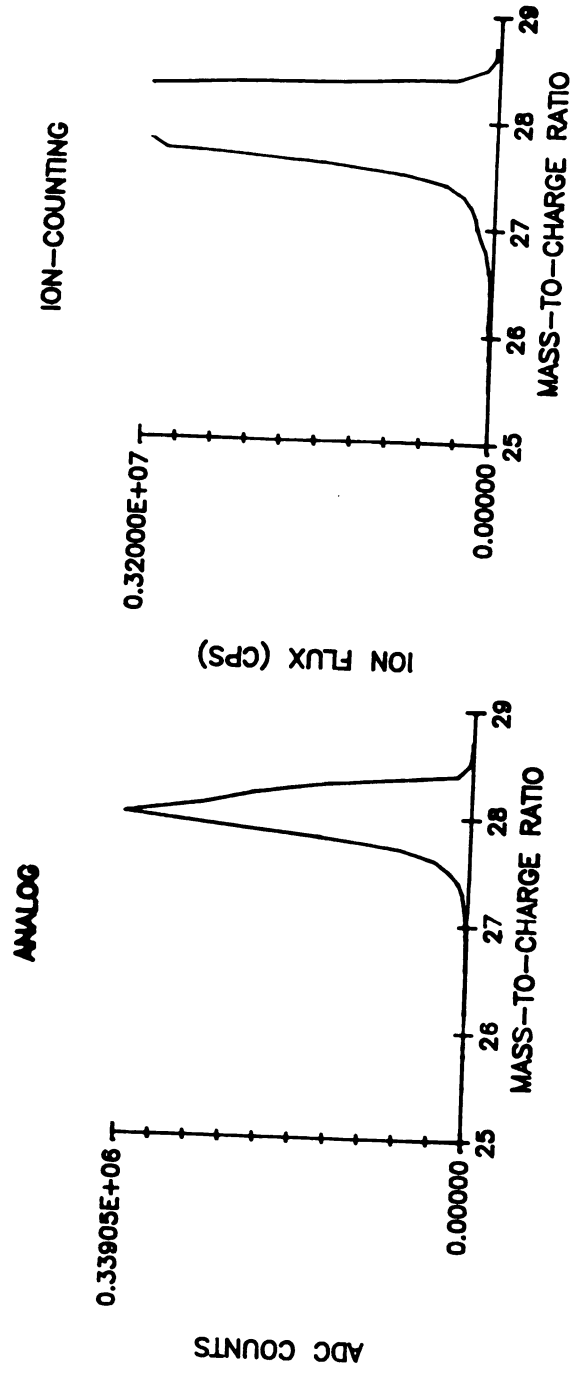


Figure 5.7 A mass sweep of the peak representing the nitrogen molecular ion (m/z=28), showing the resulting analog intensity and the measured ion flux.

Figure 5.8 Plot of ion flux versus analog intensity for the mass sweep of the nitrogen molecular ion in Figure 5.7.

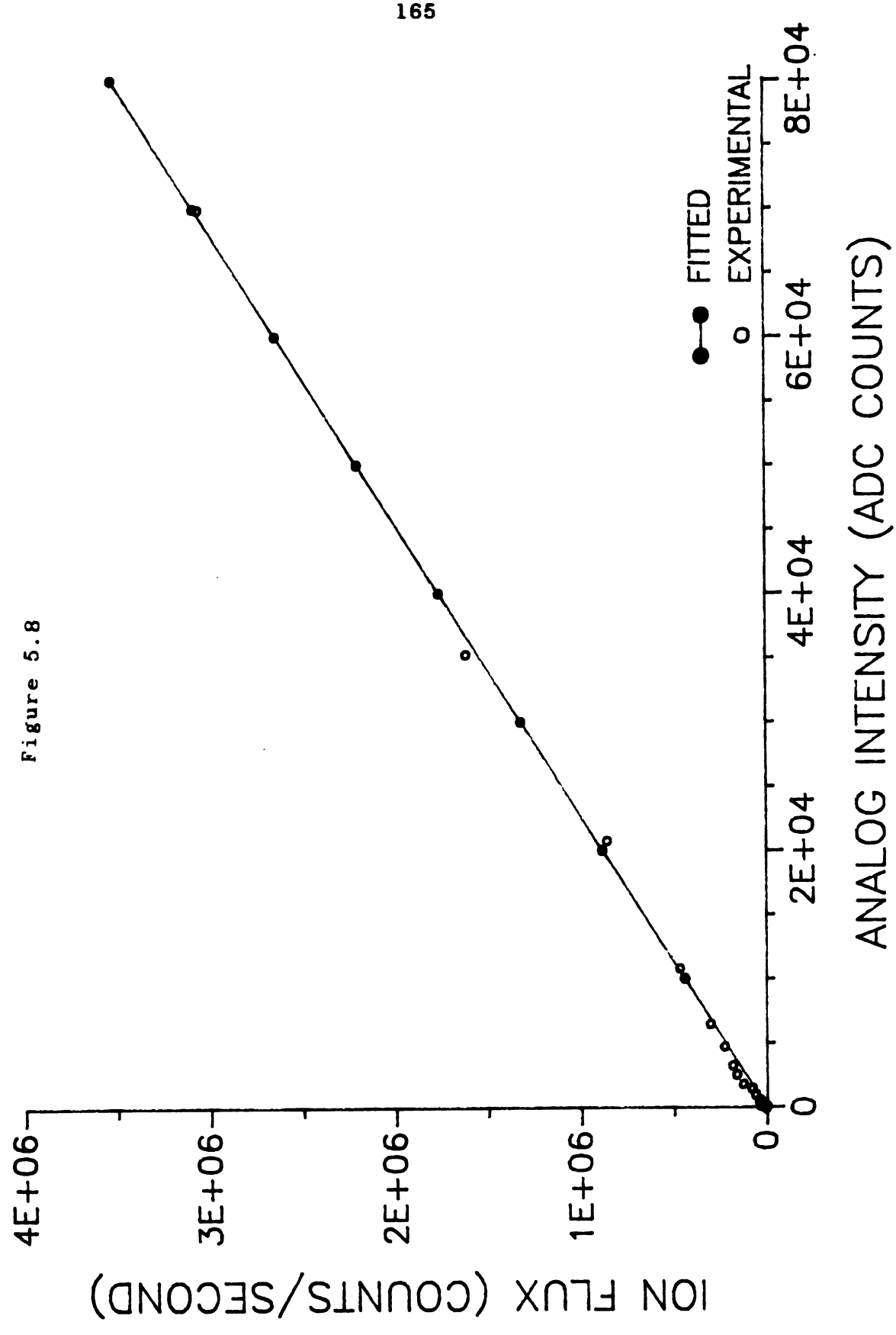

ION FLUX (COUNTS/SECOND)
4E+06
3E+06
2E+06
1E+06
0
0
2E+04
4E+04
6E+04
8E+04
ANALOG INTENSITY (ADC COUNTS)
FITTED
EXPERIMENTAL

Figure 5.8

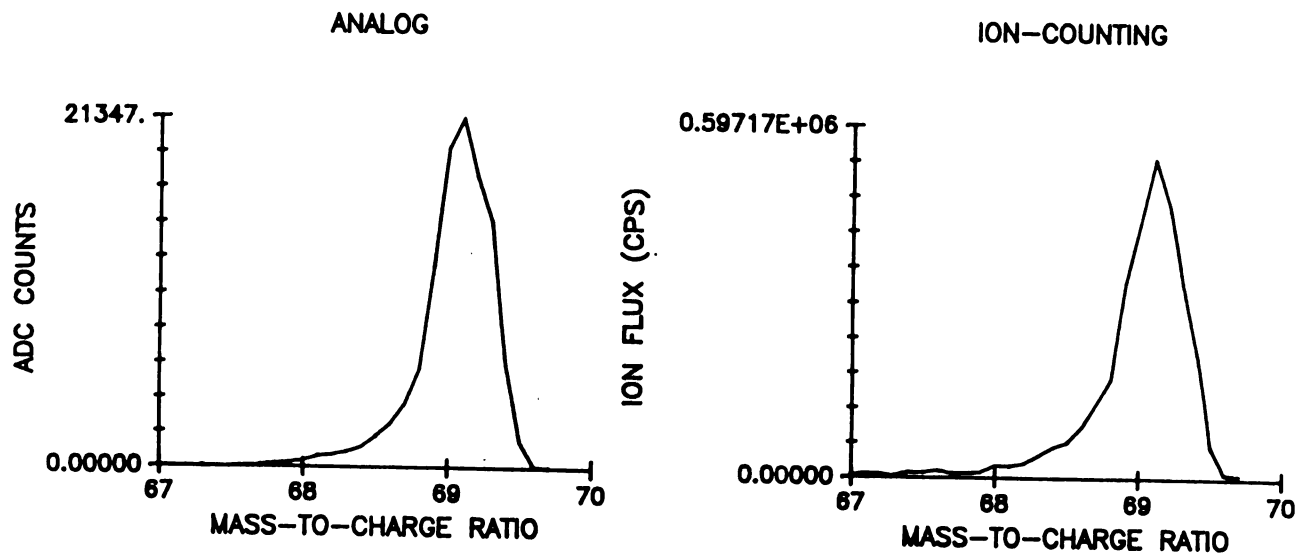


Figure 5.9 A mass sweep of the peak representing CF_3^+ (m/z=69) from PFK, showing the resulting analog intensity and measured ion flux.

Figure 5.10 Plot of ion flux versus analog intensity for the mass sweep of CF_3^+ in Figure 5.9.

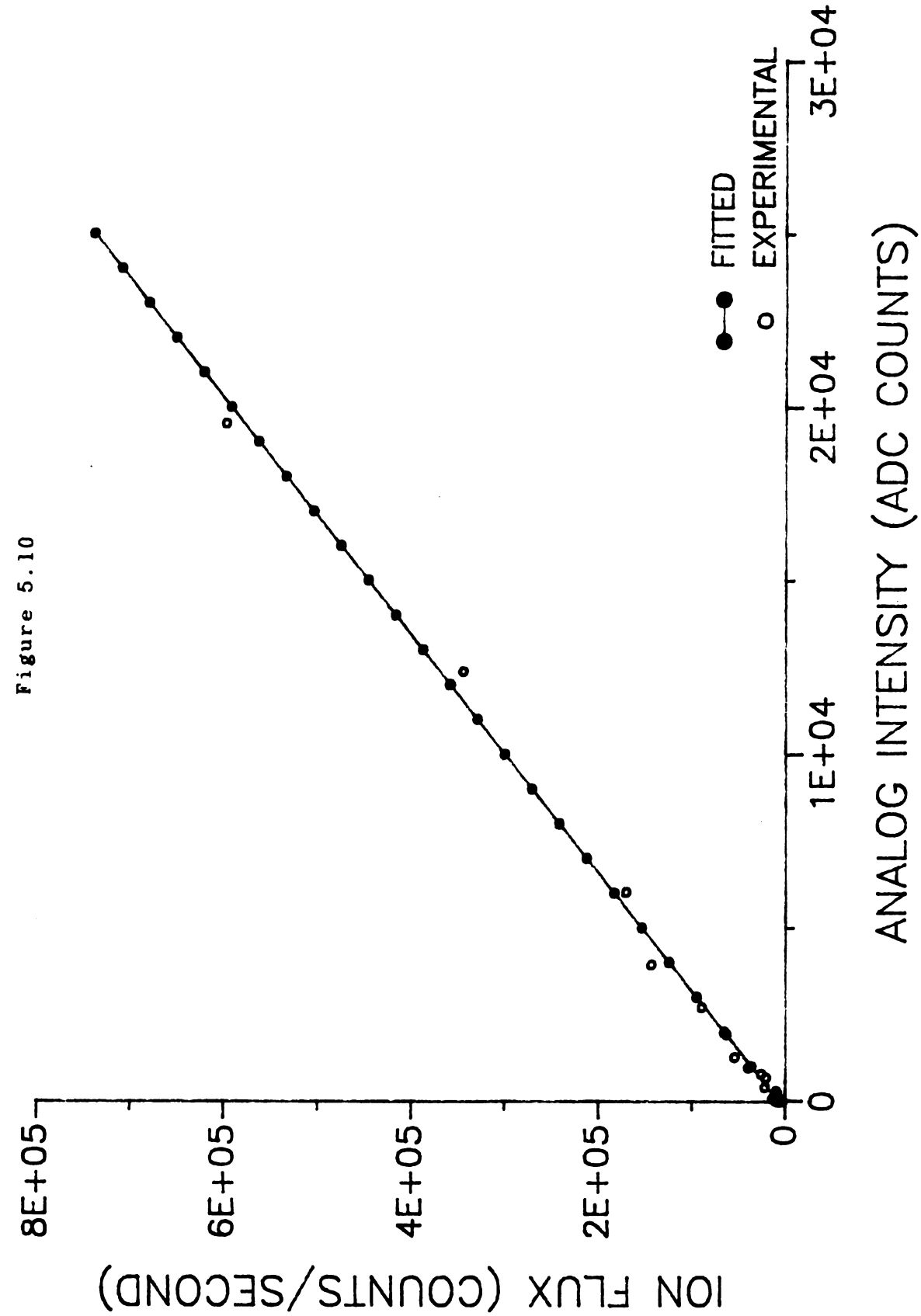

ION FLUX (COUNTS/SECOND)
8E+05
6E+05
4E+05
2E+05
0
0
1E+04
2E+04
3E+04
ANALOG INTENSITY (ADC COUNTS)
FITTED
EXPERIMENTAL

Figure 5.10

ion flux versus analog intensity yields a straight line with a slope of 32 ions per ADC count (Figure 5.10). The data for 5.7 and 5.9 were taken at a higher analog gain (lower anode voltage) than the data of Figure 5.6, yet the ion intensities for the three experiments can be directly compared since the output of the dual-mode system, after reverse-fitting and count/current calibration, is the absolute ion intensity in ion counts/second. Comparison of the count/current calibration factors for N_2^+ and CF_3^+ indicates that CF_3^+ ejects more electrons at the first surface of the detector on the average than does N_2^+. Based upon velocity alone, one would expect N_2^+ to eject more electrons upon striking the detector than CF_3^+, however the increased number of constituent atoms for CF_3^+ and the decreased strength of the carbon-fluorine bond versus the nitrogen-nitrogen bond provide the possibility of fragmentation upon impact with each fragment ejecting primary electrons.

Having dual outputs is advantageous for the reverse-fitting process in three ways. First, the resolution window for the counting system can be empirically determined either from the maximum experimentally obtained count rate or by successive approximation until the theoretical curve matches the experimental curve. This measurement can be made as often and as many times as needed. Second, the recognition and control of the operative portion of the parabolic, raw calibration curve can be achieved, merely by adjusting the

threshold voltage for the protection grid. Then the system will switch to analog measurements when the reverse-fitting is no longer useful or valid. Third, it is unnecessary to adjust the multiplier voltage to obtain the full dynamic range. Changing the multiplier voltage would result in a change in the analog calibration and, thus, the digital correction.

5. Conclusion

In conclusion, the overlap region where analog and pulse-counting outputs are both valid provides points for absolute calibration of the analog output and thus achieves the full dynamic range for MS/MS. The intensity value obtained from the control system is the actual ion flux rate in counts per second. Calibration of the factors used to correct for pulse pile-up and to convert ADC units to count rate can be automatic and frequent using data from normal operation.

The dual-mode detection system also extends the useful dynamic range available in tandem mass spectrometry. This means that the least abundant daughter ion of the least abundant parent ion can be detected as well as the most abundant unfragmented parent ion without adjusting the multiplier voltage. Furthermore, automatic over-current protection for the pulse-counting section of the multiplier makes pulse-counting information available whenever feasible

without damaging the multiplier in the event of an intense peak, such as the parent peak.

REFERENCES

1. R. W. Kondrat and R. G. Cooks, Anal. Chem., 50, 81A-92A (1978).

2. Galileo ElectroOptics Corporation, Sturbridge, MA 01518.

3. E. A. Kurz and R. L. Roy, 27th Annual Conference on Mass Spectrometry and Allied Topics, 1979, p. 479.

4. A. E. Schoen, Ph. D. Dissertation, Purdue University, 1981.

5. R. S. Matthews, M. S. Thesis, Michigan State University, 1982.

6. B. H. Newcome, private communication.

7. EG&G Princeton Applied Research, Princeton, NJ 08540.

8. American Micro Devices, Sunnyvale, CA 94086.

9. E. J. Darland, G. E. Leroi, and C. G. Enke, Anal. Chem., 52, 714 (1980).

10. G. Ratzlaff, N. Penix, M. Rabb, M. Kristo, private communications.

11. C. La Lau, *Advances in Analytical Chemistry and Instrumentation*, Vol. 8, 93-120, A. L. Burlingame, ed., John Wiley, 1970.

12. Udo Fehn, Int. J. Mass Spectrom. Ion Phys., 15, 391 (1974).

13. R. C. Lao, R. Sander, and R. F. Pottie, Int. J. Mass Spectrom. Ion Phys., 10, 309 (1972).

14. R. F. Pottie, D. L. Cocke, and K. A. Gingerich, Int. J. Mass Spectrom. Ion Phys., 11, 41 (1973).

15. C. N. Burrous, A. J. Lieber, and V. T. Zaviantseff, Rev. Sci. Instrum., 38, 1477 (1967).

16. W. E. Potter and K. Mauersberger, Rev. Sci. Instrum., 43, 1327 (1972).

17. J. Dimeff, A. J. Lieber, and C. N. Burrous, Rev. Sci. Instrum., 37, 1562 (1966).

18. B. L. Schram, A. J. H. Boerboom, W. Kleine, and J. Kistemaker, Physica, 32, 749 (1966).

19. L. A. Dietz and J. C. Sheffield, J. Appl. Phys., 46, 4361 (1975).

20. R. J. Beuhler and L. Friedman, Int. J. Mass Spectrom. Ion. Phys., 23, 81 (1977).

21. R. J. Beuhler and L. Friedman, J. Appl. Phys., 48, 3928 (1977).

22. J. L Holmes and J. E. Szulejko, Org. Mass Spectrom., 18, 273 (1983).

23. D. Hunt, W. C. Brumely, G. C. Stafford, and F. K. Botz, Practical Spectroscopy, Vol. 3, 327-8, 373-5.

24. G. C. Stafford, J. R. Reeher, and R. S. Story, Practical Spectroscopy, Vol. 3., 359-63, 373-5.

25. Intel Corp., Santa Clara, CA 95052.

26. FORTH Inc., Hermosa Beach, CA 90254.

27. Malmstadt, H. V., Enke, C. G., and Crouch, S. R., Electronics and Instrumentation for Scientists, The Benjamin/Cummings Publishing Co., Inc., 1981.

CHAPTER SIX

A TRAP-AND-PULSE ALGORITHM FOR DETECTION OF COLLISIONALLY ASSISTED REACTION PRODUCTS

1. Introduction

Programmable control systems for MS/MS allow the creation of new types of experiments by adding new software routines. One example of the power and flexibility of such a control system is the "trap-and-pulse" algorithm for enhancing the detection of ionic products from collisionally assisted reactions (CAR).

Most often, collisionally activated dissociation (CAD) is the method used for modifying the parent ion introduced into the central quadrupole of a TQMS (1). However, other methods of modification or reaction are possible. In fact, the TQMS was originally developed to study laser photodissociation reactions (2,3). Moreover, there is a growing interest in using CAR where the method of modification is a reaction between the parent ion and the collision gas to form an adduct, to study ion/molecule reactions in the central quadrupole (4-12). CAR can provide different, and sometimes more selective, information about analyte ions (7).

CAR has two main drawbacks. First, the yield of CAR products is generally very poor. Optimum conditions for forming CAR complexes, namely high collision gas pressures and low axial energies for the parent ion, are poor conditions for detection of the product ions. Because the

CAR adducts are formed at high pressures and low kinetic energies for the parent ions, the product ions also have little kinetic energy and, therefore, little tendency to exit the collision region towards the detector. Often the interquad lens between the second and third quadrupoles is used as a "drawout" lens to extract ions near the exit of the central quadrupole by placing a large attractive potential on the lens (negative potential for positive ions and positive potential for negative ions). The second drawback to CAR is that primarily only the CAR adducts are formed with few, if any, fragment ions. Thus, CAR provides little or no structural information about the adduct or analyte ion. Sometimes such structural information would be useful.

The trap-and-pulse technique (13) described in this chapter allows the mass spectrometrist to vary the average residence time of ions inside the central quadrupole, increasing the reaction yield for stable product ions. This technique improves the signal-to-background noise ratio (S/BN) for stable CAR product peaks more than averaging the ion current at the detector for equivalent scan rates. Also, trap-and-pulse produces many product ions which have been previously unseen in conventional CAR. These new ions are primarily fragments of the initial adduct and their presence provides additional structural information about the adduct. Furthermore, the trap-and-pulse algorithm can

be applied to any TQMS with interquad lenses and modifiable or extensible software without any instrumental changes.

2. Experimental

Instrumentation

All experiments were conducted on an Extrel 400/3 triple quadrupole mass spectrometer. The instrument has a dual EI/CI source, which was modified to allow fast atom bombardment (FAB) ionization (14). The FAB gun is a capillaritron probe gun from Phrasor Scientific. All samples were introduced either through a volatile liquids inlet or a direct probe. Collision gases were admitted through a stainless-steel vacuum line. The pressure of gas in the collision region was regulated by a model 216 pressure/flow controller from Granville/Phillips, Inc. The FAB experiments and trap-and-pulse ion lifetime experiments were conducted by Greg Dolnikowski (14); all other experiments were conducted by myself.

Computer System

An 8086-based microcomputer controls the TQMS and acquires data using a software control system based upon the programming language FORTH. The control system, based upon the work of Dr. Carl Myerholtz (13) provided the modular software and flexibility needed to create new types of experiments.

Chemicals Used

Glacial acetic acid was obtained from Mallinckrodt as an analytical reagent. Acetone and benzene were obtained from Fischer Scientific. All of the above were used without further purification. The glycerol used in this project was vacuum-distilled. Xenon was obtained from Matheson Scientific and was 99.99% pure.

Ion/Molecule Reactions

The reaction of the methyl cation with acetone was carried out in the TQMS by introducing $5x10^{-6}$ torr acetone into the ion source and $5x10^{-4}$ torr acetone into the central quadrupole region. The acetone in the ion source region was ionized by 70 eV EI which produced the methyl cation in addition to many other fragment ions. The methyl cation was mass selected using the first quadrupole and reacted with the neutral acetone in the central quadrupole (Reaction 1).

$$CH_3^+ + (CH_3CH_2)_2CO \rightarrow C_6H_{13}O^+ \qquad (1)$$

The reaction of the protonated acetone molecule with acetone was carried out in a similar manner, except that the pressure in the ion source region was $5x10^{-4}$ torr. This created conditions suitable for chemical ionization and protonated acetone was formed in abundance. The protonated acetone molecule was mass selected using the first

quadrupole and reacted with the neutral acetone in the second quadrupole (Reaction 2).

$$(CH_3CH_2)_2CO^+ + (CH_3CH_2)_2CO \rightarrow C_{10}H_{20}O_2^+ \qquad (2)$$

The reaction of protonated glycerol with acetic acid was implemented by protonating the glycerol during FAB in the ion source of the TQMS and by regulating the partial pressure of acetic acid in the center quadrupole region. The ion source and central quadrupole regions are differentially pumped so that there is little mixing of the neutral glycerol and acetic acid. The protonated glycerol was mass selected by the first quadrupole and reacted with the acetic acid in the second quadrupole to form the proton-bound adduct ion (Reaction 3).

$$C_4H_{11}(OH)_2^+ + CH_3CH_2COOH \rightarrow C_7H_{19}O_4^+ \qquad (3)$$

Experiments were also performed with the benzene molecular ion, using acetone as a nonreactive collision gas in the central quadrupole. Repeated scans under various conditions showed that no reaction between the two reagents occurs. Thus, the benzene/acetone system makes an effective probe for the physical characteristics of ion trapping, since it maintains similar conditions to the protonated acetone/acetone CAR reaction without interference from chemical reactions (ion stability).

Instrumental Conditions for Ion Trapping

Typical TQMS instrument parameters for EI, CI, and FAB are shown in Table 6.1.

Table 6.1

Typical Conditions for Ion-Trapping in the Center Quadrupole of the TQMS

Parameters	EI	+CI	-CI	FAB
Filament	70 eV	300 eV	300 eV	----
Repeller	14.3 V	33.0 V	-46.6 V	0.0 V
CIV	2.7 V	32.2 V	-37.6 V	0.0 V
EIV	14.9 V	-185.8 V	-1.5 V	-106.3 V
EXT	23.7 V	-3.7 V	75.9 V	-26.4 V
L1	-30.0 V	-13.0 V	-23.3 V	18.9 V
L3	10.0 V	-191.3 V	110.4 V	-72.2 V
Q1	- 0.5 V	20.0 V	-33.0 V	-9.4 V
L4	5.9 V	4.0 V	66.9 V	5.0 V
Q2	14.0 V	32.2 V	-26.6 V	-1.8 V
L5	var	var	var	var
Q3	4.0 V	20.0 V	-32.7 V	-13.2 V
Multiplier	-1.7 kV	-1.7 kV	-2.0 kV	-1.7 kV
Conversion Dynode	-3.0 kV	-3.0 kV	-3.0 kV	.-3.0 kV
FAB Gun	-------	---------	-------	10.0 kV

3. The Trap-and-Pulse Experiment

The trap-and-pulse experiment is based upon an experimental observation by Greg Dolnikowski (13,14) that CAR product ion currents would increase dramatically after first raising and then abruptly lowering the voltage on lens 5 (the interquad lens between the second and third quadrupole). To implement this concept, the voltage on lens 5 is raised to prevent all positive ions from exiting the collision region then, after waiting a specific length of time, the potential is lowered to release the trapped ions.

Repetitive acquisition of intensity data after pulsing out the trapped ions will define the trap-and-pulse profile. The sequence of lens voltages and the resulting trap-and-pulse ion intensity profile are shown in Figure 6.1. In order to perform trap-and-pulse experiments on negative ions, the trapping potential would be negative and the potential for pulsing the ions out of the collision region would be positive. The maximum instantaneous ion current obtained in a trap-and-pulse experiment is many times larger than that obtained under steady-state conditions for stable CAR products.

Typically, one uses a trapping potential of +100 V for positive ions and -100 V for negative ions. The pulsing potential is then -100 V for positive ions and +100 V for negative ions. Smaller voltages can be used and still trap the ions, but the most reproducible data were obtained with these values. Part of the reason for this phenomenon is shown in Figure 6.1. The maximum for the trap-and-pulse profile occurs after the voltage on L5 has reached a significantly negative value. The potential for lens 5 decreases exponentially after the control system updates the new value to the digital-to-analog converter due to the RC time constant of the L5 power supply. The data system requires a finite time (approximately 2 uS) to start acquiring intensity values after dropping the potential on lens 5. Thus, the maximum of the peak will be missed if the potential on lens 5 decays below zero before the system can

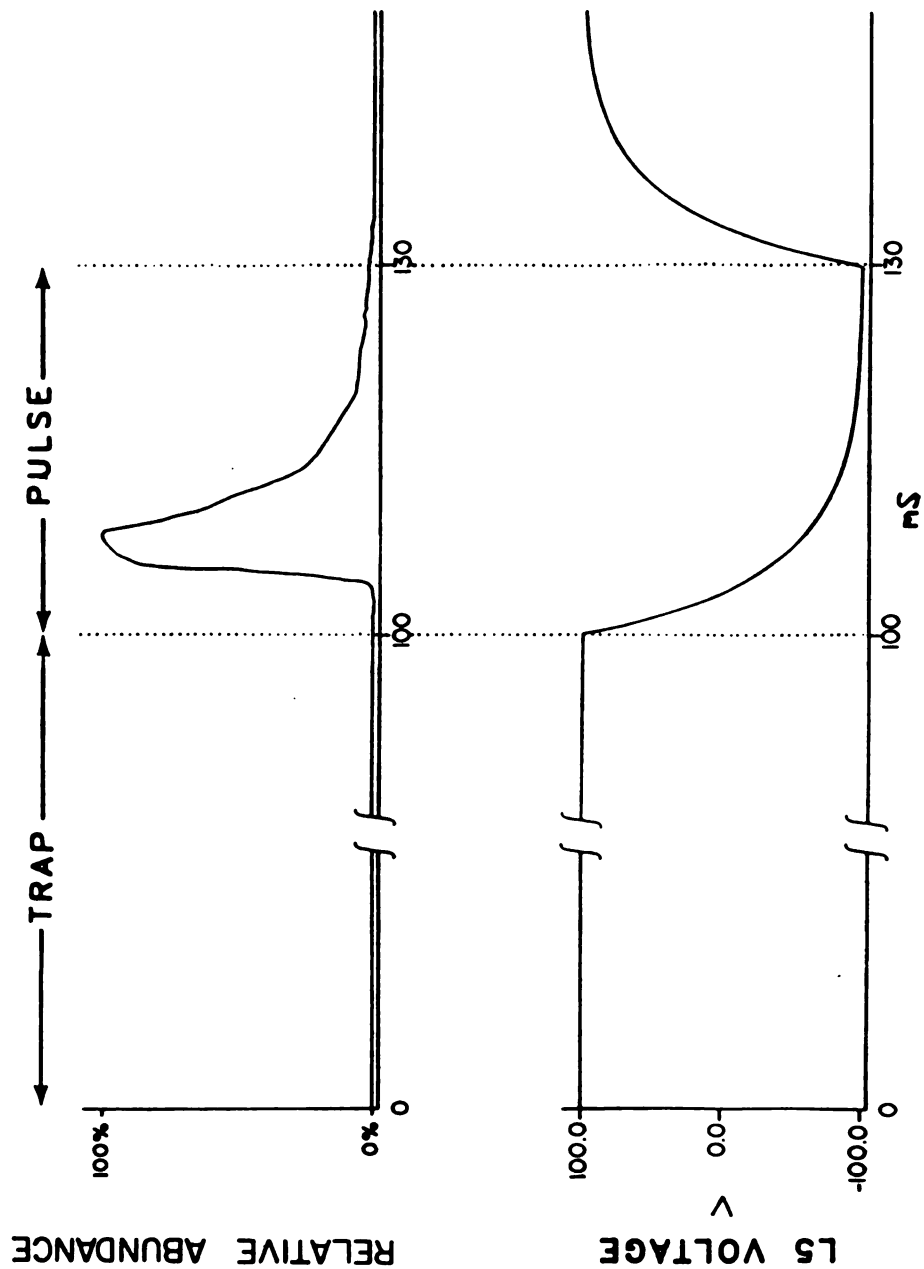


Figure 6.1 Plots of ion abundance versus time and L5 potential versus time, showing one full trap-and-pulse cycle.

acquire the first datum. The decay from +100 to -100 V ensures that this does not happen and that the trap-and-pulse profile is accurately characterized and the maximum ascertained. The large negative value for the pulse voltage also serves to purge the exit of the collision region before the next trap-and-pulse experiment.

The mass spectrometrist also has control over the trapping time. The maximum intensity of the trap-and-pulse profile increases with increasing trapping time, as one would expect, but then levels off. This effect will be more fully discussed in a later section. Although one could theoretically improve the ion intensity value indefinitely, 90 % of the final intensity normally has been reached within 100 mS. Thus, generally, the mass spectrometrist will select a trapping time less than 100 mS, by compromising between sensitivity and the length of the experiment (the quality of the data versus scan rate).

In the trap-and-pulse experiment, ions continuously enter the central quadrupole from the source through the first quadrupole. In a related technique, the "inject-trap-and-pulse" experiment, only a discrete packet of ions is allowed into the central quadrupole in order to characterize ion lifetimes in the collision region. In inject-trap-and-pulse, lens 3 (the interquad lens between the first and second quadrupole) serves as an ion gate for the central quadrupole. If the potential on lens 3 is low enough, ions can pass into the central quadrupole; if the potential on

lens 3 is too high, they cannot. In inject-trap-and-pulse, lens 3 allows ions into the central quadrupole only for a fixed period of time while lens 5 holds the collision region in the trapping mode. Again, after a specific period of time, the remaining ions from the incident packet are pulsed out of the central quadrupole. Figure 6.2 shows the sequence of voltages on L3 and L5, as well as the resulting inject-trap-and-pulse ion intensity profile. A plot of the integral intensity of the resulting inject-trap-and-pulse profile versus trapping time reveals ion lifetime statistics for the trap-and-pulse method, namely the half-life for that ionic species in the collision region. In order to study the length of time of that ions could be trapped inside the central quadrupole, the methyl cation/acetone reaction (Reaction 1) was chosen. This reaction forms the products listed in Table 6.2.

Table 6.2

Ion/Molecule Products of the Reaction Between the Methyl Cation and Acetone

m/z	Formula
15.0	CH_3^+
29.0	$C_2H_5^+$
31.0	CH_3O^+
41.0	$C_3H_5^+$
43.0	CH_3CO^+
44.0	$CH_3OCH_2^+$
57.0	$C_3H_5O^+$
58.0	$C_3H_6O^+$
59.0	$C_3H_7O^+$
117.0	$C_6H_{13}O_2^+$

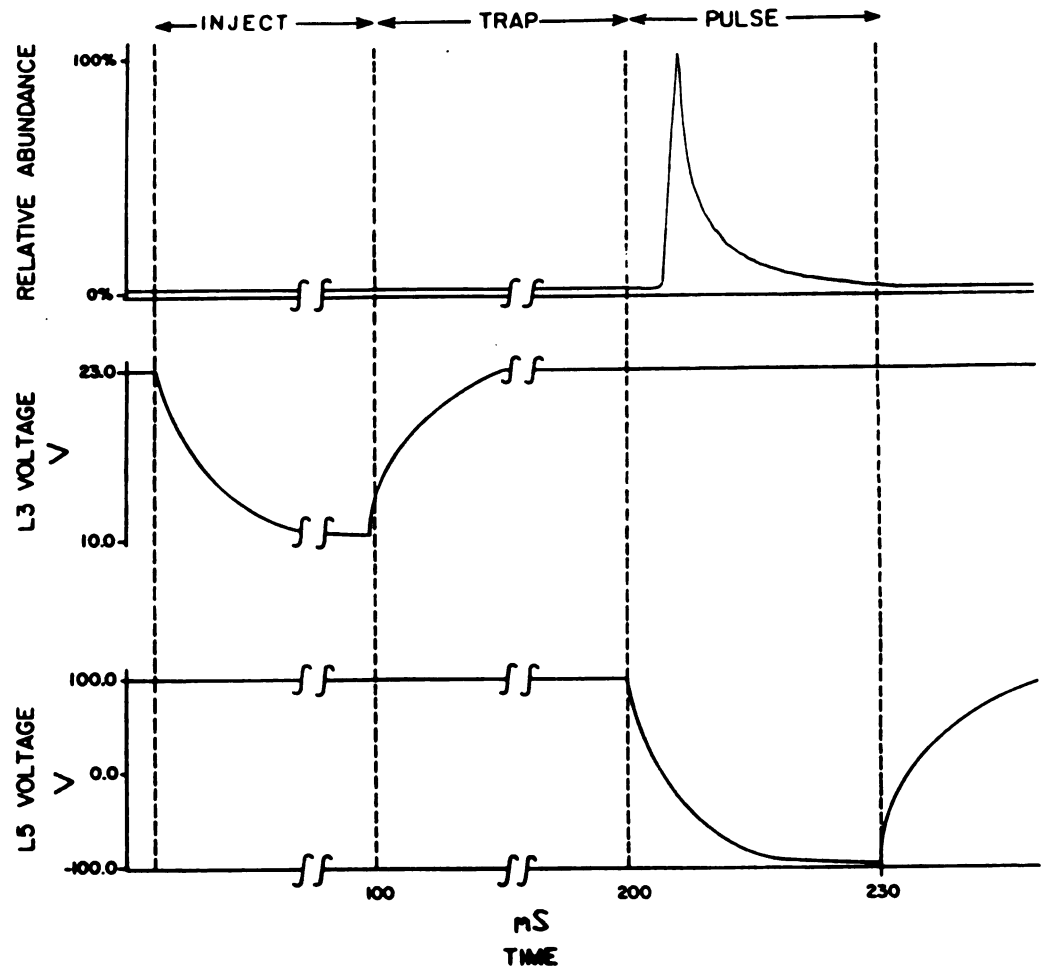


Figure 6.2 Plots of ion abundance versus time, and L3 and L5 potentials versus time, showing one full inject,trap, and pulse cycle.

It is possible to trap stable products such as those at m/z 59 $(acetone+H)^+$ and 117 $(acetone\ dimer+H)^+$ in the central quadrupole for up to 20 S. The number of trapped ions decays exponentially with increasing trapping time. The time constant for the ion at m/z 117 at $5x10^{-4}$ torr acetone was 2.8 S or, in other terms, the rate constant for ion losses was 0.358 S^{-1}. Other product ions could only be detected in the inject-trap-and-pulse experiment at lower pressures around $7x10^{-6}$ torr acetone. The less stable ions fall prey to other competitive reactions at higher pressures leading to more stable products. At the lower pressures, these less stable ions, like the ion of mass 43, could be trapped for short periods of time. The ion of mass 43 could be trapped for about 100 mS in the collision region before its signal faded into background.

4. Investigation of the Trap-and-Pulse Process

An investigation into the nature of the trap-and-pulse method was undertaken. There are three possibile means for ion loss or gain in the trap-and-pulse method. First, ions which enter the exit region of the central quadrupole will be lost if they are not trapped with 100% efficiency. If one assumes that all losses in the central quadrupole are ion concentration dependent, then the rate of change of the number of ions in this quadrupole trap will be equal to the incident ion flux minus the fraction of ions being lost (the

inverse of the trapping efficiency) per unit time multiplied by the ion concentration (Equation 1).

$$dI/dt = P - fI \qquad [1]$$

(where I is the number of ions in the trap, P is the incident ion flux, and f is the ion loss rate constant)

The solution to this differential equation, given the starting condition $I_0=0$, is given in Equation 2.

$$I = P/f*(1-\exp(-ft)) \qquad [2]$$

Thus, deviations from linearity in a plot of the integral ion intensity versus trapping time will reveal any ion concentration-dependent losses. Such a plot is shown in Figure 6.3 for the protonated acetone/acetone reaction ($5x10^{-4}$ torr collision gas pressure) (reaction 2). By fitting the resulting data to Equation 2, a rate constant of 12 +/- 2 S^{-1} was calculated. On the millisecond time scale, which is the time scale for the experiment, this corresponds to a trapping efficiency of about 99%. Although, this may seem like an almost perfect ion trap, experiments indicate and Equation 2 shows that there is a maximum ion capacity, less than the theoretical space charge limit, which will be determined by the trapping efficiency (f) and the input ion flux rate (P). For stable, nonreactive parent ions, like the benzene molecular ion, this mass transfer equilibrium determines the maximum trapping capacity of the central quadrupole.

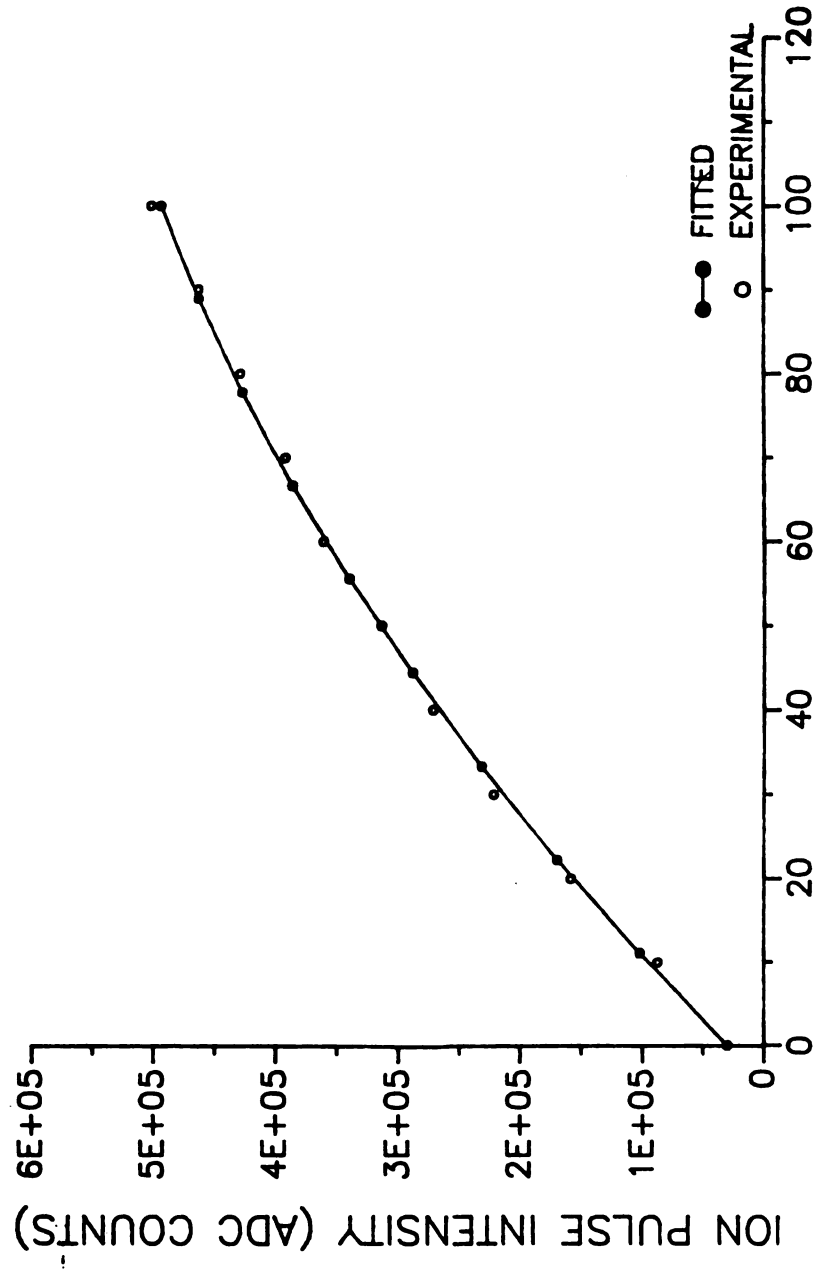


Figure 6.3 Plot of ion pulse intensity versus storage time for the proton-bound dimer of acetone and the curve obtained by fitting the data to Equation 2.

The second means of ion loss or gain in the trap-and-pulse experiment is inefficient extraction of trapped ions from the central quadrupole. The fraction of ions which can be extracted can be calculated from the number of ions represented in the trap-and-pulse profile divided by the number of ions which expected based upon 100% trapping and extraction efficiencies (the steady state product ion current times the trapping time). Experiments with the benzene molecular ion and acetone indicate an extraction efficiency on the order of 19%. For stable product ions like the proton-bound dimer of acetone formed from the reaction of protonated acetone with acetone, the extraction efficiency as calculated above yields values above 100%. Apparent extraction efficiencies above 100% indicate that an ion is being produced faster than it is being lost, usually at the expense of less stable species in the reaction chamber.

Thus, the third method of ion loss or gain is formation or degradation of product ions in the central quadrupole during the trapping period. For a stable, nonreactive ion, like the benzene molecular ion, the net ion gain due to reaction should be 0. For a stable product ion like the proton-bound dimer of acetone, the net ion gain will be positive and, in fact, overcome losses due to imperfect trapping and extraction. For an unstable product ion like CH_3CO^+ from reaction 1, the net ion gain will be negative

and may prevent detection of the ion at high pressures and/or long trapping times.

For stable product ions with reactions yields that overcome losses due to imperfect trapping, the trap-and-pulse ion intensity limit reflects a limit to the accumulation of ions in the collision chamber due to space-charge. This effect is also seen in other techniques (14). When monitoring a parent ion with a significant abundance, such as protonated glycerol generated by FAB, a high rate of parent ion influx occurs into the central quadrupole and the space-charge limit will be reached quickly (within 100 mS). Lower abundance parent ions will, of course, reach the space-charge limit more slowly.

By raising the voltage on L4 (the interquad lens between quadrupoles one and two) after a certain length of time after raising the voltage on L5 to begin trapping, all losses due to diffusion back through the entrance to the quadrupole are eliminated. This technique makes a nice probe for the causes of imperfect ion trapping and the effects of ion energy and collision gas pressure. Figure 6.4 shows a plot of integral ion intensity versus trapping time for the benzene molecular ion with a wide variety of ion energies with no collision gas present (5×10^{-7} torr). This plot serves to show that the oscillations in ion pulse intensity are real, since the oscillations from different experiments reflect the same pattern. Apparently, there is significant transverse ion movement in this experiment,

Figure 6.4 Plots of ion pulse intensity versus storage time for the benzene molecular ion (no collision gas) for several parent ion energies.

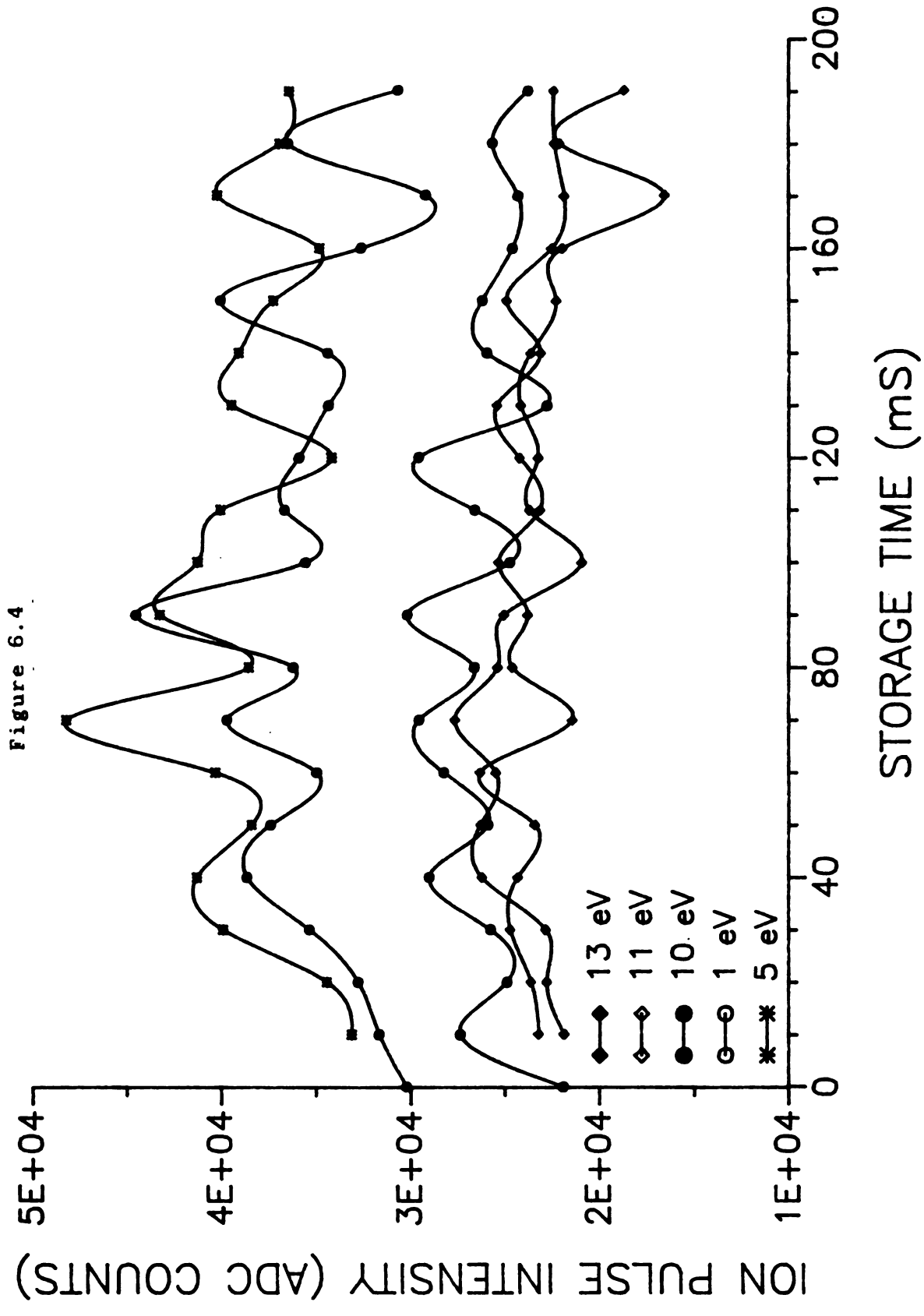

ION PULSE INTENSITY (ADC COUNTS)
5E+04
4E+04
3E+04
2E+04
1E+04
0
40
80
120
160
200
STORAGE TIME (mS)
13 eV
11 eV
10 eV
1 eV
5 eV

Figure 6.4

causing the extracted ion intensity to increase when the ions are moving towards L5 upon extraction. The low frequency of the oscillations shows that the ion motion is not due to the initial kinetic energy of the ion (at 1 eV the ion is moving at 1.57×10^5 cm/S and the quadrupole is 20 cm long, yielding frequencies in the kilohertz range). This movement is caused by repulsion of ions due either to the raising of L5 or, more likely, the raising of L4. All of the following studies involve thermalizing the benzene molecular ion with acetone collision gas.

Figures 6.5 and 6.6 show the effect of collision gas pressure for two different kinetic energies of the benzene parent ion. Both figures show that the trapping of ions where they can be extracted (either the exit region or, for this experiment, the entrance region where L4 can push the ions toward the exit) is enhanced by higher collision gas pressures. It is also clear that the ion lifetimes are very much dependent on pressure. Analysis of the data from Figure 6.5 (1 eV) shows the dependence of ion loss on collision gas pressure (Table 6.3).

Table 6.3
Dependence of Ion Loss
on Collision Gas Pressure

Pressure	Loss Rate Constant
2×10^{-4} torr	7 +/- 1 S^{-1}
2×10^{-5} torr	5 +/- 1 S^{-1}
2×10^{-6} torr	2 +/- 1 S^{-1}
6×10^{-7} torr	0.9 +/- 0.5 S^{-1}

(6×10^{-7} torr not shown on graph)

Figure 6.5 Plots of ion pulse intensity versus storage time for the benzene molecular ion for several pressures of acetone collision gas (parent ion kinetic energy of 1 eV).

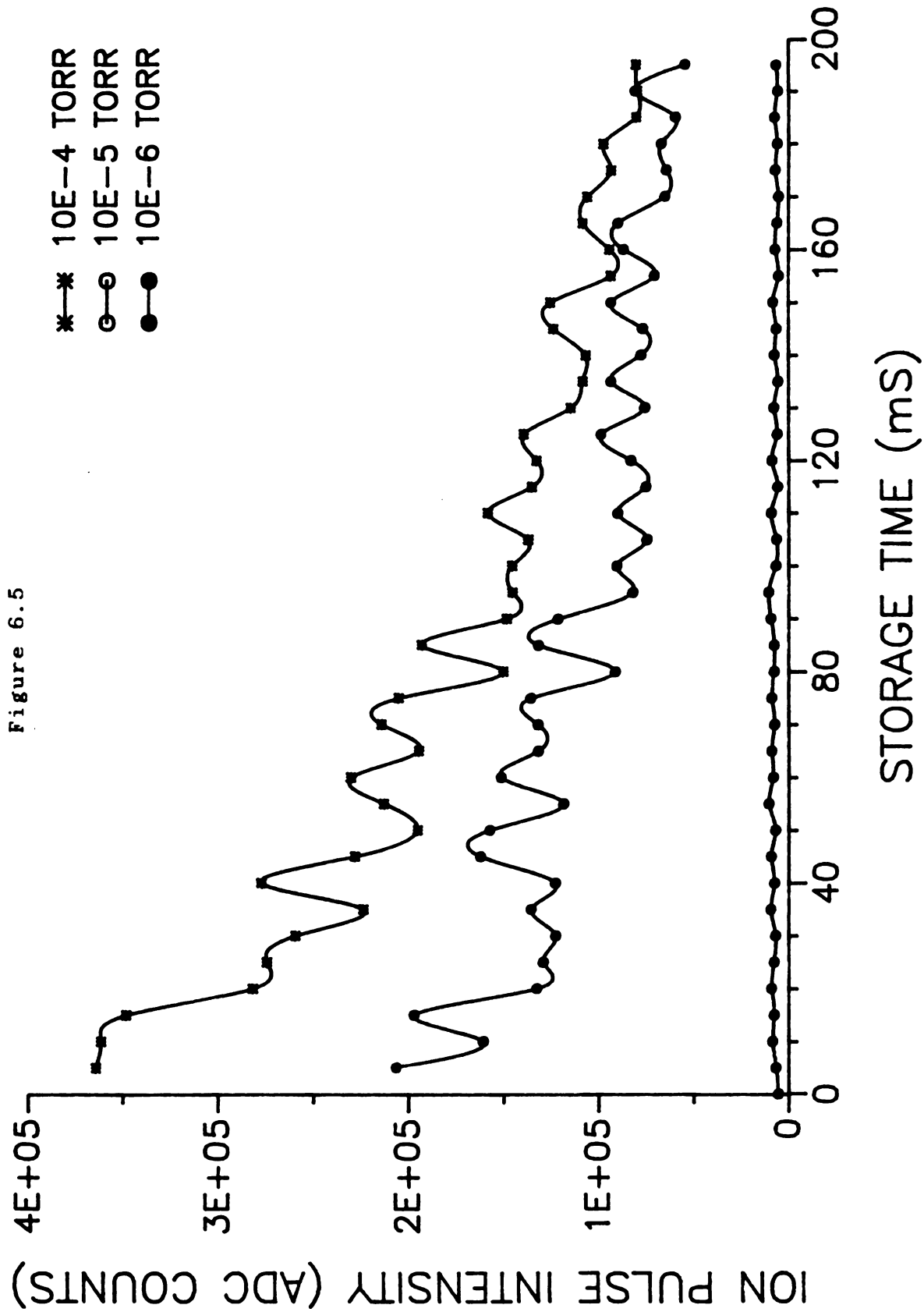

10E-4 TORR
10E-5 TORR
10E-6 TORR
ION PULSE INTENSITY (ADC COUNTS)
4E+05
3E+05
2E+05
1E+05
0
0
40
80
120
160
200
STORAGE TIME (mS)

Figure 6.5

Figure 6.6 Plots of ion pulse intensity versus storage time for the benzene molecular ion for several pressures of acetone collision gas (parent ion kinetic energy of 10 eV).

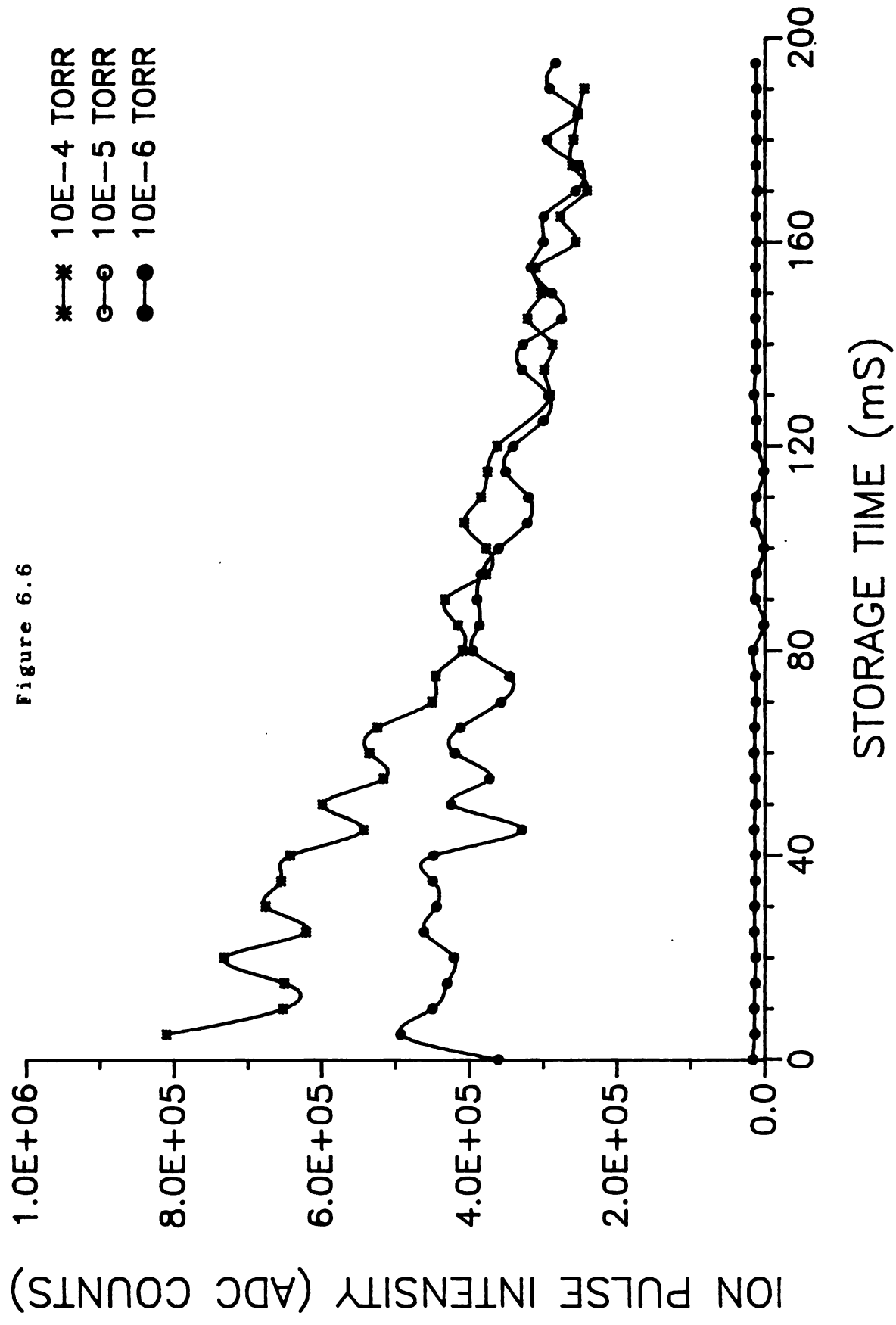

10E−4 TORR
10E−5 TORR
10E−6 TORR
1.0E+06
8.0E+05
6.0E+05
4.0E+05
2.0E+05
0.0
0
40
80
120
160
200
STORAGE TIME (mS)
ION PULSE INTENSITY (ADC COUNTS)

Figure 6.6

Therefore, as collision gas pressure increases, ion loss also increases. This loss could either be caused by increased ion scattering or a quenching of the ions due to the higher pressure. Also note the decreased rate constant for 10^{-4} torr with this experiment (7 S^{-1}) as opposed to that calculated from the data of Figure 6.3 (12 S^{-1}). Thus, ion loss is decreased when the potential is raised to prevent ions from leaving the central quadrupole, as in the data of Table 6.3, over the normal trap-and-pulse experiment of Figure 6.3. This indicates that a certain fraction of the trapped ions diffuse back through the entrance to the collision chamber past L4.

Figures 6.7-6.9 show the effect of ion energy for different collision gas pressures. At high collision gas pressures (Figure 6.7 and 6.8), higher ion energies produce increased ion yield. This is most likely due to the increased ability for ions of higher energy to penetrate the central quadrupole to reach the exit of the quadrupole, where they can be extracted by L5. This phenomenon must be balanced against the need for low ion energies for increased reaction yields in exothermic CAR when determining the proper ion energy for analysis. At low collision gas pressures (Figure 6.9), e. g., vacuum, lower ion energies produce increased ion yield. This probably occurs, because more of the ions probably reflect back out of the collision cell at higher than at lower ion energies.

Figure 6.7 Plots of ion pulse intensity versus storage time for the benzene molecular ion for several parent ion kinetic energies (pressure of acetone collision gas of 2×10^{-4} torr.

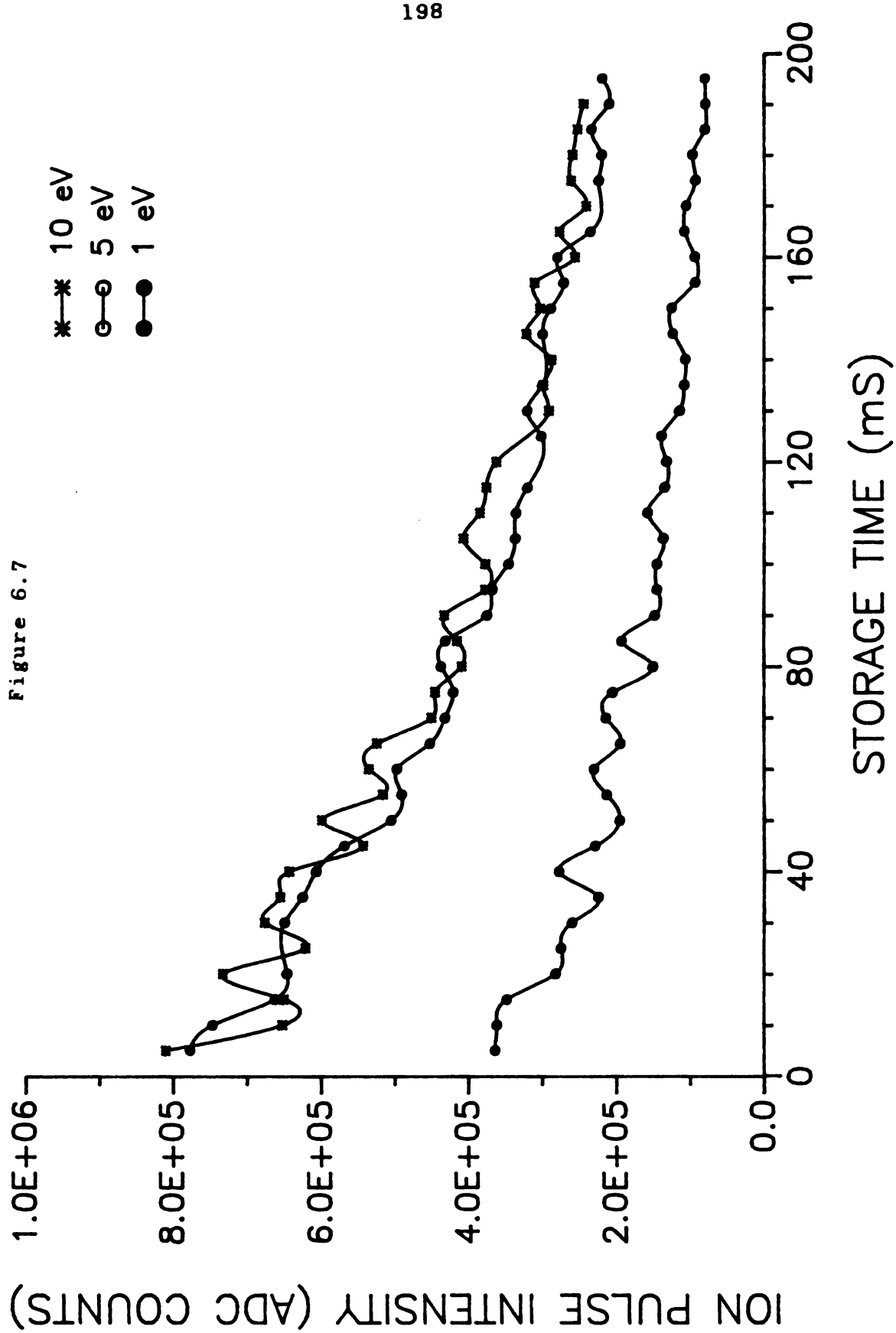

10 eV
5 eV
1 eV
1.0E+06
8.0E+05
6.0E+05
4.0E+05
2.0E+05
0.0
0
40
80
120
160
200
ION PULSE INTENSITY (ADC COUNTS)
STORAGE TIME (mS)

Figure 6.7

Figure 6.8 Plots of ion pulse intensity versus storage time for the benzene molecular ion for several parent ion kinetic energies (pressure of acetone collision gas of 2×10^{-5} torr.

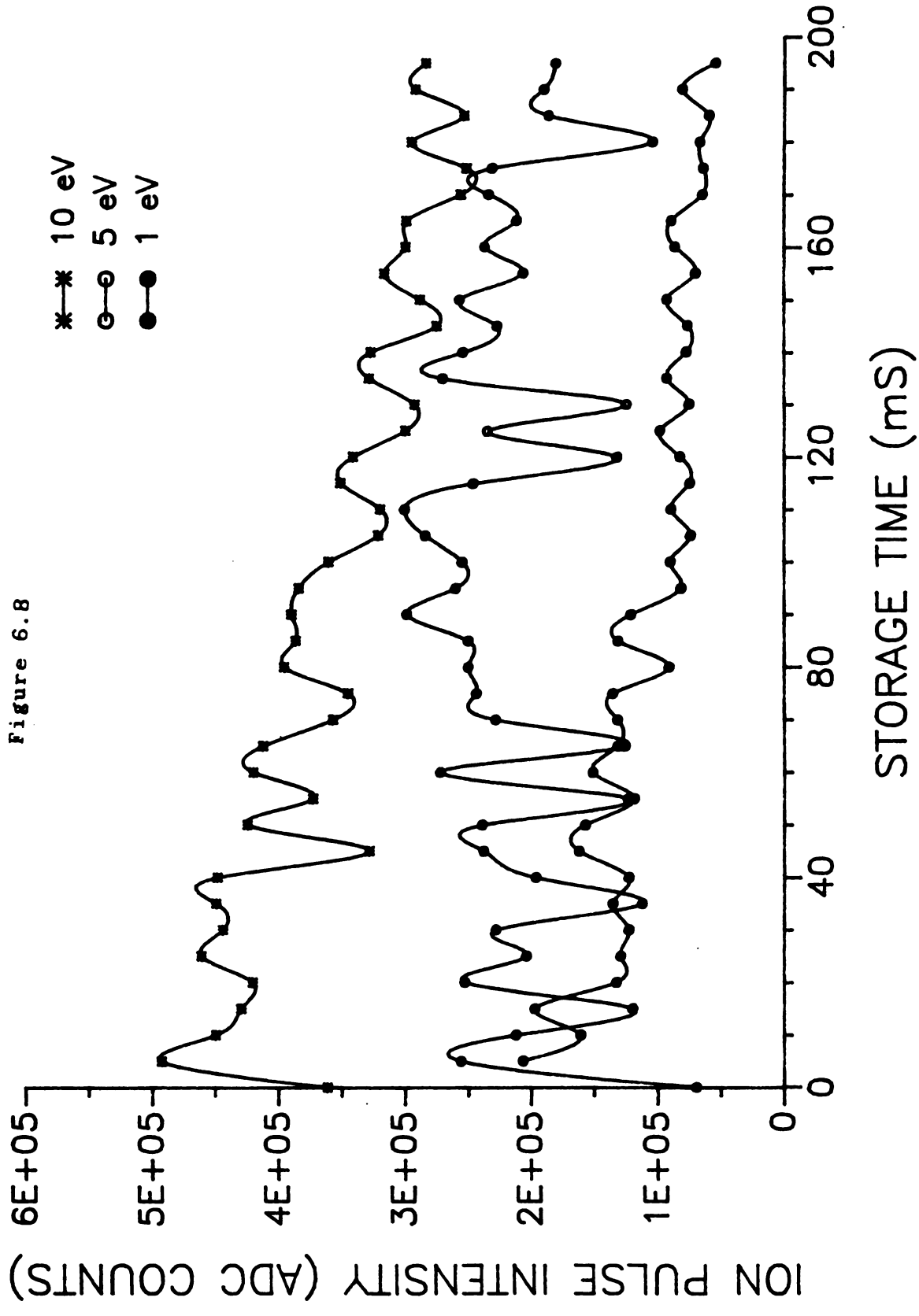

10 eV
5 eV
1 eV
ION PULSE INTENSITY (ADC COUNTS)
6E+05
5E+05
4E+05
3E+05
2E+05
1E+05
0
0
40
80
120
160
200
STORAGE TIME (mS)

Figure 6.8

Figure 6.9 Plots of ion pulse intensity versus storage time for the benzene molecular ion for several parent ion kinetic energies ; pressure of acetone collision gas of 6×10^{-7} torr.

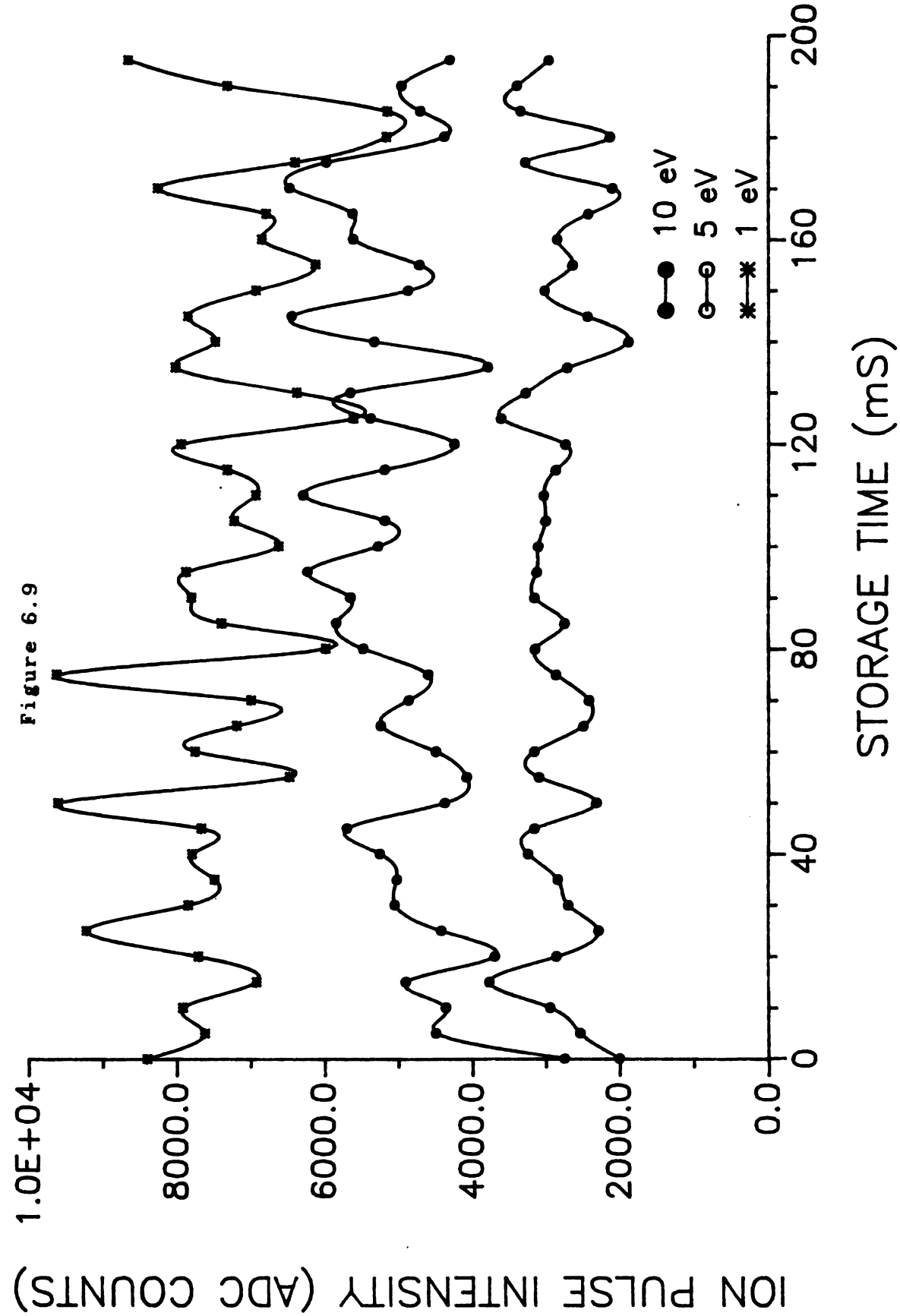

ION PULSE INTENSITY (ADC COUNTS)
1.0E+04
8000.0
6000.0
4000.0
2000.0
0.0
0
40
80
120
160
200
STORAGE TIME (mS)
10 eV
5 eV
1 eV

Figure 6.9

Analyses of the trap-and-pulse and the inject-trap-and-pulse profiles also provides some insight into the trap-and-pulse experiment. The inject-trap-and-pulse profile quickly reaches a maximum and then decays exponentially with a time constant of 4 mS, which is also the time constant of the L5 power supply. The trap-and-pulse profile is not cleanly exponential, however. The first part of the temporal decay is exponential, but the last part has a square root dependency, indicating diffusional control. Furthermore, the width of the trap-and-pulse profile is wider at higher than at lower collision gas pressures. This further indicates that the trap-and-pulse profile is at least partly controlled by diffusional effects.

5. The Trap and Pulse Algorithm and Software Considerations

The trap-and-pulse experiment has been incorporated into the full range of MS/MS scans and device sweeps. Thus, if the mass spectrometrist wishes to employ the trap-and-pulse method of data collection for a set of experiments he invokes the reaction scans. RDSCAN, the reaction daughter scan or reaction product scan, identifies the mass-to-charge ratio of the products resulting from the reaction of the selected parent ion with the reactive collision gas. RNSCAN, the reaction neutral loss scan, identifies the mass-to-charge ratio of parent ions which undergo a given mass change when reacting with the collision gas. RPSCAN, the

reaction parent scan, identifies the mass-to-charge ratio of parent ions which produce a given product upon reaction with the collision gas. RSWEEP, the reaction device sweep, generates the full ion current profile upon varying the value of a physical device. RSWEEP is especially useful in finding the optimum values for parameters, such as the RF voltage value in the central quadrupole (M2) and the voltage offset for quadrupole three, which will be held constant during a reaction scan.

These reaction scans perform exactly like their conventional counterparts, except that they perform a trap-and-pulse experiment after every step of the device being scanned. For a scan in which the device is either quadrupole 1 or 3 or both, the mass value is typically incremented in steps of 0.1 AMU. In that case, a trap-and-pulse experiment would be performed every 0.1 AMU. The maximum intensity of the trap-and-pulse profile is then used as the ion intensity for that mass-to-charge ratio. These intensity values versus the mass-to-charge ratio at which they were acquired define the mass spectral peak shape, which, in turn, can be further processed to find peak locations and peak intensities.

The software for the RSCANS and RSWEEPS can be found in Appendix I. The essential difference in the software between the reaction scans and conventional scans is the use of the routine RACQUIRE (reaction acquire) instead of ACQUIRE. Both routines return a 32-bit intensity value to

the stack. ACQUIRE returns a 32-bit averaged intensity value, which is proportional to the steady state ion current at the detector. RACQUIRE, on the other hand, returns 32-bit value to the stack which is proportional to the maximum ion current generated by the trap-and-pulse experiment.

In order to obtain this value, RACQUIRE first sets the voltage on L5 to the value specified in the variable VTRAP. This places the central quadrupole in the trapping mode. Second, RACQUIRE waits the number of milliseconds specified in variable TSTORE. This determines the accumulation of ions and the average ion residence time in the trap. Third, RACQUIRE changes the voltage on L5 to the value specified in the variable VPULSE. This pulses the ions out of the trap towards the detector and purges the trap for the next experiment. Fourth, RACQUIRE begins acquiring ion intensity values, using the routine ACQUIRE. Each value could have a user-specified amount of averaging, but generally unaveraged intensity values are used, since they can be obtained more frequently, in order to be certain of determining the exact peak maximum. RACQUIRE obtains a finite number of ion intensity values, specified by the variable #ACQS. The user selects #ACQS so as to be sure to catch the entire trap-and-pulse profile. While acquiring these intensity values, RACQUIRE finds the maximum ion intensity value. This maximum then becomes the intensity value for the current mass-to-charge ratio, which will be further used to define the mass spectral peak profile. All of the variables

mentioned above are set by the user in a menu called RXSET. In RXSET, the user can choose the values appropriate to his analysis or accept the default values.

An alternative set of reaction scans uses the integrated intensity of the trap-and-pulse profile as the intensity for that device value. These scans and sweeps are designated +RDSCAN (integrating reaction daughter, or product, scan), +RPSCAN (integrating reaction parent scan), etc. The software for these scans is identical to the normal (maximum value) reaction scans, except that the integrating reaction scans use the routine +RACQUIRE, which sums the resulting intensities for the trap-and-pulse profile, rather than finding the maximum. However, we have found that these scans give poorer signal-to-background noise ratios than the equivalent reaction scan which just uses the trap-and-pulse maxima. This is probably due to the poor definition of the baseline for these peak shapes, due to the asymmetry of the peak, thus yielding poorly reproducible peak areas for the intensity. More sophisticated peak-defining and integration algorithms might have improved the reproducibility of the peak areas, but would have slowed the reaction scans even further.

Reaction scans are much slower than conventional scans, because of the time spent trapping at each step of the scan. Typical durations for a reaction scan from 50-500 AMU are 1 to 2 minutes depending on the trapping time selected. Thus, reaction scans are unsuitable for transient samples such as

gas chromatographic peaks. However, the trap-and-pulse algorithm has also been implemented for single reaction monitoring (SRM) and multiple reaction monitoring (MRM). In SRM, ions of a specific mass are selected in quadrupoles 1 and 3 and intensity values are acquired repetitively. Thus, SRM monitors the time-dependent intensity of a particular parent-daughter ion pair. MRM repetitively monitors a number of these pairs, obtaining the time-dependent intensity on a less frequent basis. Obviously these modes are used for targeted analysis of transient samples. They not only achieve better characterization of the time profiles of such transient samples, but also better single-to-noise ratios for the peak profiles, (e. g., chromatographic peak area), since more time is spent collecting data at the reaction pairs of interest than is possible in a full scan. Thus, the trap-and-pulse technique can be used to characterize transient samples as well using R-MRM. The FORTH code for R-MRM also appears in Appendix I.

6. Results

Figures 6.10 and 6.11 demonstrate the two advantages of the trap-and-pulse technique over conventional CAR scans. Figure 6.10 shows the increase in the ratio of signal to noise for the trap-and-pulse technique over conventional CAR for equivalent amounts of averaging. Each of sweeps A through C are of the acetone dimer pseudomolecular ion formed from Reaction 3. Sweep A is a conventional CAR

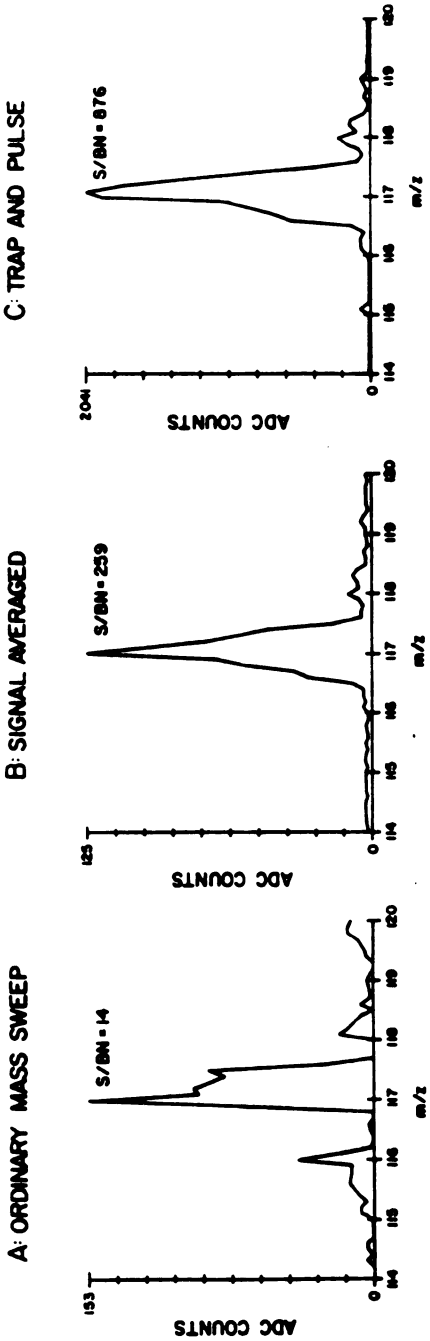


Figure 6.10 Three mass sweeps of Q3 over the isotope peaks of the proton-bound dimer of acetone (A) by conventional data collection (250 amu/S), (B) by conventional data collection with real-time signal averaging (5 amu/S), (C) by trap and pulse data collection (5 amu/S).

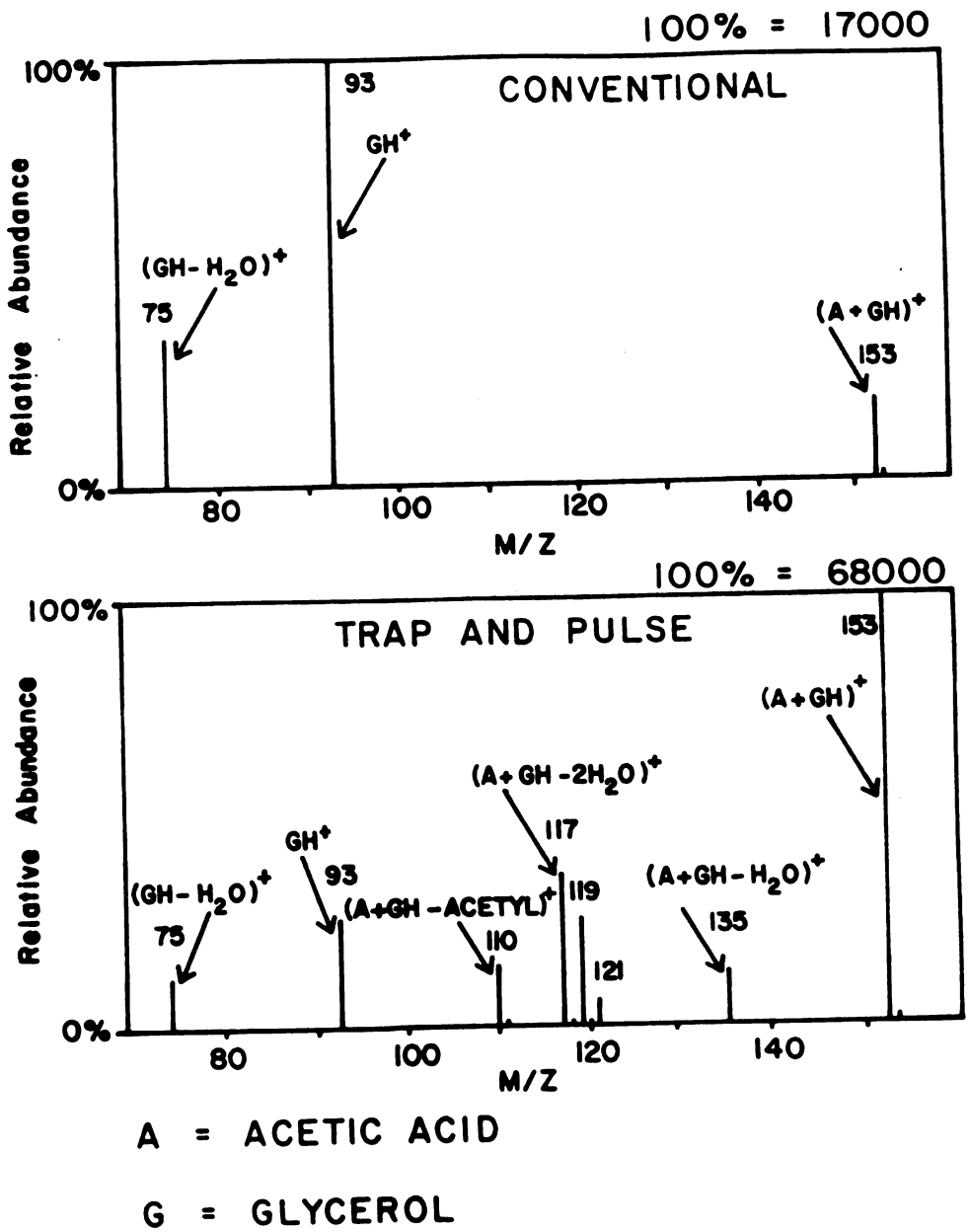


Figure 6.11 Two product ion scans of the ion/molecule reaction between protonated glycerol and acetic acid (parent ion is $(M+H)^+$ of glycerol at m/z 93) by (A) conventional data collection and by (B) trap-and-pulse data collection.

daughter scan with no signal averaging. Sweep B is a conventional CAR daughter scan with 256 averages per datum (20 mS of averaging). Sweep C is a reaction scan with 20 mS of trapping time. The resulting signal-to-background noise ratios were calculated as the peak signal intensity divided by the standard deviation of the background. The S/BN ratio for the trap-and-pulse peaks is four times larger than for the signal averaged peak. It is clear that, for CAR product ions, averaging the ion current in the central quadrupole is superior to averaging at the detector for equivalent scan rates. This occurs not from a decrease in the amount of noise, as in signal averaging, but rather from increasing the signal more than the noise. Furthermore, the reaction scans can help discriminate against certain noise sources, such as FAB neutral noise and synchronous noise in the detection electronics, since these noise sources are only present in one analog-to-digital conversion of the maximum trap-and-pulse intensity, while the ion current is summed for the whole trapping time. Figure 6.10 also shows that the mass spectral peak profile or resolution is not degraded through the trap-and-pulse technique.

Figure 6.11 shows the increase in the number and abundance of products obtained from the trap-and-pulse technique over a conventional CAR scan. Figure 6.11A shows a conventional daughter scan of the products formed from the reaction of protonated glycerol and acetic acid in the central quadrupole (Reaction 3). Figure 6.11B shows the

trap-and-pulse scan for the same reaction. The appearance of the spectrum is obviously greatly different. First, the ratio of product ion intensities to the parent ion intensity is greatly increased in the trap-and-pulse spectrum. This should be expected since parent ions are being converted to product ions continuously during the trapping period. The intensity of some product ions are enhanced more than others, since unstable product ions can decompose or react further with the collision gas given long trapping times. Figure 6.11 also shows that at $7x10^{-5}$ torr acetic acid in the collision chamber, ionic products are observed that are not seen in a conventional scan. The peaks at m/z 110, 117, 119, 121, and 135 are not present even at low intensity in the conventional spectrum unless the pressure of acetic acid is increased by at least an order of magnitude. These products are observed during the chemical ionization of glycerol and acetic acid; similar products have been observed in a reaction of protonated alcohols and acetic acid in ion cyclotron resonance (15). The peaks at m/z 119 and 121 are due to proton transfer to the acetic acid dimer; the peaks at m/z 110, 117 , and 135 represent new ions that are the result of characteristic neutral losses from the proton-bound adduct of glycerol and acetic acid. These extra peaks could provide more information about the adduct ion or even the analyte ion.

7. Conclusion

It is possible to trap stable ions with very low axial kinetic energies, such as CAR products, with up to 100% efficiency in the central quarupole and extract them with up to 19% efficiency of a TQMS instrument with no physical modifications. The ability to trap ions efficiently in the central quadrupole increases the detectability (signal-to-background noise ratio) of CAR products and the number of products observed in the CAR spectrum. The trap-and-pulse method can conceivably be used to enhance ion signals for other reactions in the central quadrupole, including charge transfer or photodissociation.

REFERENCES

1. R. A. Yost, C. G. Enke, D. C. McGilvery, and J. D. Morrison, Int. J. Mass Spectrom. Ion Phys., 1979, 30, 127-136.

2. D. C. McGilvery and J. D. Morrison, Int. J. Mass Spectrom. Ion Phys., 1978, 28, 81-92.

3. M. L. Vestal and J. H. Futrell, Chem. Phys. Lett., 1974, 28, 559-560.

4. J. D. Morrison, K. Stanney, and J. M. Tedder, J. Chem. Soc. Perkin Trans. II, 1981, 838-841.

5. J. D. Morrison, K. Stanney, and J. M. Tedder, J. Chem. Soc. Perkin Trans. II, 1981, 967-969.

6. J. A. Chakel and C. G. Enke, Anal. Chem., in press.

7. D. D. Fetterolf, R. A. Yost, and J. R. Eyler, Org. Mass Spectrom., 1984, 19, 104-5.

8. R. A. Yost and D. D. Fetterolf, Mass Spectrom. Rev., 1983, 2, 1-45.

9. D. D. Fetterolf and R. A. Yost, Int. J. Mass Spectrom. Ion Proc., 1984, 62, 33-49.

10. J. P. Schmit and P. H. Dawson, N. Beaulieu, Org. Mass Spectrom., 1985, 20, 269-275.

11. J. Jalonen, J. Chem. Soc., Chem. Commun., 1985, 872-874.

12. J. P. Schmit, S. Beaudet, and A. Brisson, Org. Mass Spectrom., 1986, 21, 493-498.

13. G. G. Dolnikowski, M. J. Kristo, C. G. Enke, and J. T. Watson, Int. J. Mass Spectrom. Ion Proc., in press.

14. G. G. Dolnikowski, Ph. D. Dissertation, Michigan State University, 1987.

15. C. A. Myerholtz, Ph. D. Dissertation, Michigan State University, 1983.

16. R. T. McGiver, Jr. and R. L. Hunter, Int. J. Mass Spectrom. Ion Proc., 1985, 64, 67-77.

17. P. W. Tiedeman and J. M. Riveros, J. Am. Chem. Soc., 1974, 96, 185-9.

CHAPTER SEVEN

SUGGESTIONS FOR FUTURE WORK

The completion of the multimicroprocessor TQMS control system has made the prototype TQMS instrument in our laboratory (the U. S. S. Enke) a viable and useful instrument. Preliminary studies have shown that the U. S. S. Enke is more sensitive than our EL 400/3 TQMS instrument. However, more research should be conducted on this instrument in order to fully exploit its enhanced control features- real-time graphics and linked device sweeps.

The creation of the Novix Control System has demonstrated the exciting possibilities, as well as the problems, of scanning quadrupole instruments very rapidly. Since scanning faster requires amplifiers with greater bandwidths and greater bandwidths pass higher frequencies of noise without attenuation, synchronous noise must be reduced in the instrument and detection system. My experience has shown that very little attention is paid to reducing noise in scientific instruments. For example, the mechanically convenient stack configuration increases capacitive coupling between the RF and the signal, since the RF high voltage leads are in close proximity to the low-level current signals. However, beyond totally redesigning the vacuum chamber to eliminate the stack, many other improvements can be made to the 400/3 instrument to reduce this noise. The distance between the preamplifier and the analog anode of

the ion multiplier should be reduced, the various power supplies should be decoupled to eliminate transmission of noise through the power lines, and the placement of voltage references should be carefully designed. Upon reduction of synchronous noise in the system, design of a high bandwidth preamplifier and multiamp amplification scheme will allow full exploitation of both the dynamic range of TQMS and the scan speed of the Novix control system. These modifications would allow the full power of high-speed data acquisition to be realized on this instrument.

The extra speed of the Novix control system should also be used to develop real-time experimental optimization and control. I have started exploring this possibility by writing a small MS/MS database manager and expert system. This system can be found starting at Block 3000 on the hard disk. Basically, this database contains TQMS experiments appropriate to identification of various chemical compounds. I have already entered a few organophosphorous pesticides explored in the work of Mark Bauer (1,2) into the database, but did not test the real-time identification. Eventually, such a database/expert system could be linked with a chromatography expert system in order to identify peaks as they elute into the ion source.

My work with peak-finding algorithms will hopefully open up more discussion in the literature about this very important subject. Incorrect peak-finding can render useless even the most carefully acquired data. The

development of the hardware peak-finder has shown that the peak-finding function can be performed in hardware, yet maintain the flexibility of exploring new algorithms through downloadable software sequencing. Peak-finding in hardware provides faster execution and allows the use of more sophisticated, yet more time-consuming algorithms. In retrospect, the design of the hardware peak-finder would have been simplified by using the new Weitek 32-bit integer processor (3), which came into production after the start of the project. The Weitek chip can perform most arithmetic and logical operations in a single 100-nS clock cycle, as well as perform quick multiplication of two 32-bit numbers. This would provide a simpler and less expensive design, since most of the needed functions would be provided in the processor itself. However, the progress of electronics practically outdates any design before its completion.

The dual-mode control system is an excellent tool for mass spectrometrists. With the proper commercial development and exploitation, the dual-mode detector will become the detector of choice for mass spectrometry and possibly other areas of analytical chemistry, such as optical spectroscopy.

The trap-and-pulse experiment has shown that collisionally assisted reactions can be analytically useful (4). The primary limitations to the utility of CAR in the past have been the poor yields for CAR products and the lack of structural information about the adduct ions. The trap-

and-pulse experiment has greatly enhanced the yield of product ions and new products have been detected that increase the structural information about the adduct ions. Nevertheless, this experiment points to an even better solution to the problems of CAR. By redesigning the collision chamber, one can create and control an electric field gradient in the axial direction, allowing storage of the ions in a potential well and then forcing all ions towards the exit of the central quadrupole, generally improving the creation and collection of ionic products from CAR.

Although much work has been done already, the development of TQMS instrumentation is far from complete. As shown by the recommendations above, the bulk of research in this area should fall into four areas: improving the speed of data collection, improving data reduction, improving the accuracy and dynamic range in measuring the ion current, and improving the collision cell design.

REFERENCES

1. M. R. Bauer, M. S. Thesis, Michigan State University, 1983.

2. M. R. Bauer, Ph. D. Thesis, Michigan State University, 1986.

3. WTL-1033, Weitek Corp., Santa Clara, CA 95054.

4. G. G. Dolnikowski, Ph. D. Dissertation, Michigan State University, 1987.

APPENDIX A.

FORTH CODE FOR LINKED SCANS

```
SCR#1
  0 ( Load Block for Linked Device Scans- MJK 11/16/86)
  1 8 LOAD    ( Basic FORTH Stuff)
  2 ' (CREATE) 'CREATE !
  3 540 LOAD    ( Math Load)
  4 : COMMAND ;
  5 612 625 THRU    ( Slave 1 Device Access)
  6 : IGET    8070 BLOCK  0 0 ITABLE  128 (CMOVE ;
  7 IGET
  8 1250 1253 THRU    ( Spline Fitting)
  9 1254 1256 THRU    ( User Interface)
 10 ' ?CREATE 'CREATE !
 11
 12
 13
 14
 15

SCR#2
  0 ( Spline Fitting- MJK 11/16/87- adopted from Novix routines)
  1 20 CONSTANT PTS    PTS 2ARRAY raw    999 CONSTANT n
  2 : )raw ( v)    n raw 2!  1  [']  n  +! ;
  3
  4 : x ( i-f)    raw @  0  2)N ;
  5 : y ( i-f)    raw 2+ @  0  2)N ;
  6 : h ( i-f)    DUP )R  1+ x  R)  x  F- ;
  7 : v ( i-f)    DUP )R  1- h  R)  h  F/ ;
  8 : w ( i-f)    DUP )R  1+ y  I y  F-  R)  h F/ ;
  9
 10 6 FINTEGER F6.
 11 : ai ( i-f)    v ;
 12 : bi ( i-f)    v  1.0  F+  2.0  F* ;
 13 : ci ( i-f)    DROP  1.0 ;
 14 : di ( i-f)    DUP )R  w  I 1-  w  F-  F6.  F*  R)  h  F/ ;
 15

SCR#3
  0 ( Spline Fitting - MJK 11/16/86)
  1 PTS SARRAY beta    PTS SARRAY gamma    PTS SARRAY phi
  2
  3 : BETAS    1 bi  FDUP  1 beta s!  n 1-  2 DO  I ai  I 1-  ci  F*
  4    FSWAP  F/  I bi  FSWAP  F-  FDUP  I beta s!  LOOP  FDROP ;
  5
  6 : GAMMAS    1 di  1 beta s@  F/  FDUP  1 gamma s!
  7    n 1-  2 DO  I ai  F*  I di  FSWAP  F-
  8    I beta s@  F/  FDUP  I gamma s!  LOOP FDROP ;
  9
 10 : PHIS    0 phi  PTS 2*  ERASE  n 2-  gamma s@
 11    FDUP  n 2-  phi s!  1 n 3 -  DO  I ci  F*  I beta s@  F/
 12    I gamma s@  FSWAP  F-  FDUP  I phi s!  -1  +LOOP  FDROP ;
 13
 14 : MATRIX    BETAS  GAMMAS  PHIS ;
 15
```

```
SCR#4
  0 ( Spline Fitting - MJK 11/16/86)
  1 PTS SARRAY aj   PTS SARRAY bj   PTS SARRAY cj  PTS SARRAY dj
  2
  3 : SOLVE   MATRIX  n 1-  0 DO  I h  F6.  F*
  4    I phi s@  FOVER  F/  I aj s!
  5    I 1+ phi s@  FSWAP  F/  I bj s!  I h  F6.  F/
  6    I 1+ y  I h  F/  FOVER  I 1+ phi s@  F*  F-  I cj s!
  7    I y  I h  F/  FSWAP  I phi s@  F*  F-  I dj s!  LOOP ;
  8
  9 : SUBDIVIDE ( n-n i ! -32k)   DUP  0 raw @  n 1- raw @  1+
 10    WITHIN  IF  0 BEGIN  1+  2DUP  raw @  > NOT  END  1-
 11    ELSE  R>  2DROP  32768  THEN ;
 12
 13
 14
 15

SCR#5
  0 ( Spline Fitting- MJK 11/16/87)
  1
  2 : SPLINE ( n-n)   SUBDIVIDE  >R  >N  I 1+ x  FOVER  F-
  3    FDUP  FDUP  FDUP  F* F*  I aj s@  F*  FSWAP  I dj s@  F*  F+
  4    FSWAP  I x  F-  FDUP  FDUP  FDUP  F* F*  I bj s@  F*
  5    FSWAP  R>  cj s@  F*  F+ F+  N> ;
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15

SCR#6
  0 ( User Interface for Linked Devices- MJK 11/16/86)
  1 4 CONSTANT #LMAX
  2 VARIABLE #LDEVICES
  3 CREATE LDEVICES  #LMAX 2* ALLOT
  4
  5 : ?LDEVICES   CR ." Number of Devices to Link: "  #INPUT DUP
  6    DUP  #LMAX >  ABORT" Too many devices to link"
  7    #LDEVICES !  CR CR   0 DO
  8    ." Linked Device #"  I .  ." : "  #INPUT
  9    LDEVICES  I 2* +  !   CR LOOP ;
 10
 11
 12
 13
 14
 15
```

```
SCR#7
 0 ( Input Values for Devices- MJK 11/16/86)
 1 : ?LVALUES   CR ." The Value for Mass 1000.0 will stop entries"
 2    0 ['] n !  CR CR  1024 0 DO  REGULAR ( usual ver. of NUMBER)
 3    ." Mass #"  I .  ." : "  #INPUT  DUP CR
 4    ." Value #"  I .  ." : "  #INPUT  CR CR  MATH
 5    SWAP  )raw  10000 =  IF  LEAVE  THEN  LOOP ;
 6
 7 8073 CONSTANT LBLOCK
 8 CREATE LBUFFER 2048 ALLOT
 9
10 : )LBLOCK ( n-)   )R  LBUFFER  LBLOCK I 2* +  BLOCK  1024 MOVE
11    UPDATE  LBUFFER 1024 +  LBLOCK R) 2* +  1+  BLOCK  1024 MOVE
12    UPDATE ;
13
14
15

SCR#8
 0 ( Input Values for Devices- MJK 11/16/86)
 1 8082 CONSTANT Dtable
 2
 3 : LINKS   #LDEVICES @ 0 DO  CR CR  ." Device: "
 4    LDEVICES  I 2*  +  @   DUP #DEVICE !  U.  CR  ?LVALUES  SOLVE
 5    1024 0 DO  #DEVICE @  M1  I 64 *  )UNITS  SWAP  #DEVICE !
 6    SPLINE  )DAC  LBUFFER I 2* +  !  LOOP
 7    I  )LBLOCK  LOOPP  FLUSH ;
 8
 9 : )Dtable   #LDEVICES @ 0 DO  I 2*  LDEVICES + @  #DEVICE !
10    DEVICE-ADDRESS  Dtable BLOCK  I 2* +  2+  !  UPDATE  LOOP
11    #LDEVICES @  Dtable BLOCK !  UPDATE FLUSH ;
12
13 : LSCANS   7947 SCREEN  CR  ?LDEVICES  LINKS  )Dtable ;
14
15

SCR#9
 0 Welcome to the Wonderful World of Linked Devices
 1
 2 You will first be prompted for the number of devices that you
 3 wish to link (up to 4) and the number of those devices- check
 4 the TQMS user's manual or PED for the number of a given device.
 5
 6 You will next be prompted to enter the value for each device
 7 at various masses.  You can determine these values by experiment
 8 or you can play around.  To stop entering mass-device values
 9 enter the values for mass 1000.0.  You must have values for mass
10 1000.0 and 0.0.  Enter values as integers to avoid confusing
11 the floating point processor which looks for fractions, e. g.,
12 enter 10.0 as 100.  Between entered values, values for the
13 devices will be calculated at roughly 1 amu intervals using a
14 spline fitting routine translated from one for the Novix
15 polyFORTH environment by the good people at polyFORTH.
```

```
SCR#1
  0 ( Device Tracking - MJK from CAM 11/18/86)
  1 VARIABLE Dtable COMMAND   8 ALLOT
  2 VARIABLE #devices COMMAND
  3 VARIABLE table COMMAND   8190 ALLOT
  4
  5 CODE TRACKING ( n-)   0 POP   R PUSH   I PUSH   06 # 1 MOV
  6    0 SAR V   0 W MOV   table # W ADD   Dtable # I MOV
  7    #devices 1 MOV   1NZ IF   BEGIN   W ) 0 MOV   R 0 XCHG
  8    0 R ) MOV   2048 # W ADD   LOOP   THEN   I POP
  9    R POP   NEXT
 10
 11
 12
 13
 14
 15

SCR#2
  0 ( Linked Scans- MJK 12/14/86) HEX
  1 : lstart   0 FCC0 C!  FSTOP  rate @ RATE  0 AMUF/F C! ;
  2 DECIMAL
  3
  4 : (LSWEEP) ( #steps, startdac-)   DUP  !DATA !SCRATCH
  5    0 DO
  6          ?YOK   ( detection done?)
  7          +DEVICE   ( increment and out)
  8          scratch @  DUP   ( current value)
  9          TRACKING   ( track all other devices)
 10          XOK   ( go-ahead to detection)
 11          )PEAK   ( reduction)
 12          LOOP ;
 13
 14 : LSWEEP ( step end start dev-)   DUP  !DEVICE# SWEEPDEV !
 15    )DAC&STEP  lstart  (LSWEEP)  SCANEND ; COMMAND

SCR#3
  0 ( Linked Scans - MJK 12/14/86)
  1
  2 : (L1SCAN) ( #steps startdac-)   Q1INIT
  3    0 DO
  4           ?YOK ( detection done?)
  5           +Q1)3 ( increment and out)
  6           M1SCRATCH @ DUP ( current values)
  7           TRACKING ( track all devices)
  8           XOK ( ion path done)
  9           )PEAK ( to reduction)
 10    LOOP ;
 11
 12 : L1SCAN ( step end start-)   DRR  M1  3)DAC
 13    #STEPS  lstart  (L1SCAN)  SCANEND ; COMMAND
 14
 15
```

```
SCR#4
  0 ( Linked Scans - MJK 12/14/86)
  1
  2 : (L3SCAN) ( #steps startdac-)   Q3INIT
  3    0 DO
  4           ?YOK ( detection done?)
  5           +Q3>1 ( increment and out)
  6           M3SCRATCH @ DUP ( current values)
  7           TRACKING ( track all devices)
  8           XOK ( ion path done)
  9           >PEAK ( to reduction)
 10    LOOP ;
 11
 12 : L3SCAN ( step end start-)   DRR  M3  3>DAC
 13    #STEPS  lstart  (L3SCAN)  SCANEND ; COMMAND
 14
 15

SCR#5
  0 ( Linked Scans - MJK 12/14/86)
  1
  2 : (LDSCAN) ( #steps startdac-)   Q3INIT
  3    0 DO
  4           ?YOK ( detection done?)
  5           +Q3 ( increment and out)
  6           M3SCRATCH @ DUP ( current values)
  7           TRACKING ( track all devices)
  8           XOK ( ion path done)
  9           >PEAK ( to reduction)
 10    LOOP ;
 11
 12 : LDSCAN ( step end start-)   DRR  M3  3>DAC
 13    #STEPS  lstart  (LDSCAN)  SCANEND ; COMMAND
 14
 15

SCR#6
  0 ( Linked Scans - MJK 12/14/86)
  1
  2 : (LPSCAN) ( #steps startdac-)   Q1INIT
  3    0 DO
  4           ?YOK ( detection done?)
  5           +Q1 ( increment and out)
  6           M1SCRATCH @ DUP ( current values)
  7           TRACKING ( track all devices)
  8           XOK ( ion path done)
  9           >PEAK ( to reduction)
 10    LOOP ;
 11
 12 : L1SCAN ( step end start-)   DRR  M1  3>DAC
 13    #STEPS  lstart  (LPSCAN)  SCANEND ; COMMAND
 14
 15
```

```
SCR#7
  0 ( Linked Scans - MJK 12/14/86)
  1
  2 : (LNSCAN) ( #steps startdac-)   Q3INIT  Q1INIT
  3    0 DO
  4           ?YOK ( detection done?)
  5           +Q13 ( increment and out)
  6           M1SCRATCH @ DUP ( current values)
  7           TRACKING ( track all devices)
  8           XOK ( ion path done)
  9           )PEAK ( to reduction)
 10    LOOP ;
 11
 12 : LNSCAN ( step end start-)   DRR  NSET  M1  3)DAC
 13    #STEPS  1start  (LNSCAN)  SCANEND ; COMMAND
 14
 15

SCR#8
  0 ( Master Linked Scans- MJK 12/14/86)
  1 : LSWEEP   statCLR  #DEVICE @ DUP  2PUSH
  2    @PARAMS  1PUSH  SL3 SWEEP  SL1 SWEEP  SL2 SWEEP
  3    SWEEPEND ;
  4
  5 : L1SCAN   M1 SCANINIT  SL1 L1SCAN  0 SCANEND ;
  6
  7 : L3SCAN   M3 SCANINIT  SL1 L3SCAN  1 SCANEND ;
  8
  9 : LDSCAN   M3 SCANINIT  SL1 LDSCAN  2 SCANEND ;
 10
 11 : LPSCAN   M1 SCANINIT  SL1 LPSCAN  3 SCANEND ;
 12
 13 : LNSCAN   M1 SCANINIT  NSET  SL1 LNSCAN  4 SCANEND ;
 14
 15

SCR#9
  0 ( Linked Scans for Master- MJK 12/14/86)
  1 4 CONSTANT #LDEVICES
  2 8073 CONSTANT LBLOCK
  3 8082 CONSTANT LDEVICES
  4 1LABEL table table
  5 1LABEL #devices #devices
  6
  7 : LTABLES   #LDEVICES 2*   I 1024 * +  0 1 )SLAVE
  8    LBLOCK I + BL  SL1 NOTHING  LOOP
  9    1024  IPMOVE  SL1 NOTHING  LOOP    2+  Dtable
 10    LDEVICES BLOCK  DUP @  #devices I!  2+ Dtable
 11    #LDEVICES 2*  IPMOVE ;
 12
 13 LTABLES
 14
 15
```

APPENDIX B.

FORTH CODE FOR REAL-TIME GRAPHICS

```
SCR#1
  0 ( Real-Time Graphics - MJK 12/16/86)
  1
  2 : RGINIT ( start end max-inten)   Max-Inten 2!
  3    1 CSIZE  RWAXES ; COMMAND
  4
  5 : RGSWEEP   SCANINIT  LPY@ LPX@ (CUR)  VECTOR  0
  6    BEGIN  )R
  7           XGET DUP  I !X  )UNITS
  8           YGET 2DUP  I !Y  NORMALIZE  ?LOG
  9           SWAP PLOT
 10           R)  1+  SSTAT 3 + C@  1 =
 11    END  #POINTS !
 12    SCANEND ; COMMAND
 13
 14
 15

SCR#2
  0 ( Real-Time Graphics - MJK 12/16/86)
  1 CODE ?RGPEAK   2 POP   0 POP   I PUSH   U PUSH   ?PEAK CALL
  2    U POP   I POP   NEXT
  3
  4 : ?RGPEAK   ?RGPEAK  ?peak @  0= NOT
  5    IF  XYLAST 2@  I @Y  NORMALIZE  ?LOG  I @X  HIST PLOT
  6    0 ?peak !  XYLAST 2!  THEN ;
  7
  8 : RGSCAN   SCANINIT  LPY @  LPX @  (CUR)
  9    BEGIN
 10           XGET DUP  xvalue !  )UNITS
 11           YGET 2DUP  ?RGPEAK
 12           NORMALIZE ?LOG  SWAP  VECTOR PLOT
 13           SSTAT 3 + C@  1 =
 14    END
 15    SCANEND ; COMMAND

SCR#3
  0 ( Real-Time Sweeps for Master- MJK 12/16/86)
  1
  2 : RGINIT ( d-)   'START @  2PUSH  'END @  2PUSH  SWAP
  3    2PUSH 2PUSH  SL2 RGINIT ;
  4
  5 : RGSWEEP ( d-)   statCLR  #DEVICE @  DUP  2PUSH  @PARAMS
  6    1PUSH  RGINIT ( put the maximum intensity on slave 2)
  7    SL1 SWEEP  SL2 RGSWEEP  SL3 SWEEP SWEEPEND ;
  8
  9
 10 : RGSCANINIT ( d-)   statCLR  #DEVICE @  2PUSH  @PARAMS
 11    RGINIT  SL2 RGSCAN  AC@SCAN ;
 12
 13
 14
 15
```

```
SCR#4
  0 ( Real-Time Graphics for Master- MJK 12/16/86)
  1
  2 : RG1SCAN    M1 RGSCANINIT  SL1 1SCAN  0 SCANEND ;
  3
  4 : RG3SCAN    M3 RGSCANINIT  SL1 3SCAN  1 SCANEND ;
  5
  6 : RGDSCAN    M1 RGSCANINIT  SL1 DSCAN  2 SCANEND ;
  7
  8 : RGPSCAN    M3 RGSCANINIT  SL1 PSCAN  3 SCANEND ;
  9
 10 : RGNSCAN    M1 RGSCANINIT  NSET  SL1 NSCAN  4 SCANEND ;
 11
 12
 13
 14
 15
```

APPENDIX C.

PAL SPECIFICATIONS FOR NOVIX INTERFACE

```
PAL20L10                                PAL DESIGN SPECIFICATION
N1000A                                  MICHAEL KRISTO  01/16/86
NOVIX I/O CHIP-SELECT GENERATOR
MSU CHEMISTRY DEPT.  EAST LANSING, MI
AA A9 A8 A7 A6 A5 /EN NC NC NC NC GND NC /INIT /DAQ /PEAK
 /P1SEL /P2SEL /9513 /TIME /START /CLKSEL /LBUSEN VCC

IF (VCC) LBUSEN  =  /AA* A9* /A8* /A7* EN + ; C200 WR
                    /AA* A9* A8* /A7      ; C300 RD

IF (VCC) CLKSEL  =  /AA* /A9* /A8* A7* A6* A5* EN + ;C0E0 WR
                    /AA* /A9* A8* A7* A6* A5 ; C1E0 RD

IF (VCC) START   =  /AA* /A9* /A8* A7* A6* /A5* EN + ; C0C0
WR
                   /AA* /A9* A8* A7* A6* /A5 ; C1C0 RD

IF (VCC) TIME    =  /AA* /A9* /A8* A7* /A6* A5* EN + ; C0A0
WR
                    /AA* /A9* A8* A7* /A6* A5 ; C1A0 RD

IF (VCC) 9513    =  /AA* /A9* /A8* A7* /A6* /A5* EN + ; C080
WR
                    /AA* /A9* A8* A7* /A6* /A5 ; C180 RD

IF (VCC) P2SEL   =  /AA* /A9* /A8* /A7* A6* A5* EN + ; C060
WR
                    /AA* /A9* A8* /A7* A6* A5 ; C160 RD

IF (VCC) P1SEL   =  /AA* /A9* /A8* /A7* A6* /A5* EN + ; C040
WR
                    /AA* /A9* A8* /A7* A6* /A5 ; C140 RD

IF (VCC) PEAK    =  /AA* /A9* /A8* /A7* /A6* A5* EN + ; C020
WR
                    /AA* /A9* A8* /A7* /A6* A5 ; C120 RD

IF (VCC) DAQ     =  /AA* /A9* /A8* /A7* /A6* /A5* EN + ;
C000 WR
                    /AA* /A9* A8* /A7* /A6* /A5 ; C100 RD

IF (VCC) INIT    =  /AA* A9* /A8* A7* EN ; C280 WR

DESCRIPTION
THIS PAL GENERATES THE CHIP SELECT SIGNALS FOR THE NOVIX I/O
BOARD.  /EN GOES LOW FOR A PERIOD OF TIME SELECTED BY THE 1-
TO-8 JUMPER COMING FROM THE 74164 SHIFT REGISTER.  THUS, THE
WRITE CHIP SELECT IS OF A DETERMINATE LENGTH.  THE READ
STAYS ACTIVE UNTIL WRITTEN TO AGAIN.  THE READ TRANSACTION
IS TWO STEPS: GENERATE THE READ STROBE AND THEN ENABLE THE
74HCT244'S IN ORDER TO READ THE DATA ON THE NOVIX I/O BUS.
ALL ADDRESSES ARE BASED ON A C000 BASE ADDRESS.
```

APPENDIX D.

FORTH CODE FOR PEAK-FINDING ALGORITHM

```
SCR#1
  0 ( Peak-Finding Algorithm- MJK 5/9/87)
  1
  2 VARIABLE UPFLAG                ( Number of increases in peak)
  3 VARIABLE +LAST                 ( Last pt increasing -1- or not-0)
  4 VARIABLE PWIDTH                ( Minimum acceptable width)
  5 VARIABLE POINTS                ( Number of points in peak)
  6 VARIABLE THRESHOLD 2 ALLOT     ( User-adjustable threshold)
  7 VARIABLE MAX-PEAK 2 ALLOT      ( Current maximum intensity)
  8 VARIABLE YPREV 2 ALLOT         ( Intensity of previous point)
  9 VARIABLE XVALUE                ( Device value of current point)
 10 VARIABLE STEP                  ( Step for scanned device)
 11 VARIABLE XMAX                  ( Device value for MAX-PEAK)
 12
 13
 14
 15

SCR#2
  0 ( Peak-Finding Algorithm- !PEAK - MJK 5/9/86)
  1
  2 VARIABLE #POINTS        ( The number of peaks found so far)
  3 VARIABLE DATABUFFER 8000 ALLOT       ( Where the data goes)
  4 VARIABLE B/P       ( Bytes per stored point)
  5
  6 8 B/P !            ( For our system)
  7
  8 ( !PEAK moves the parameters of the new peak to the DATABUFFER)
  9
 10 : !PEAK   #POINTS @  B/P *  DATABUFFER +  )R
 11    XMAX @  I !  ( bytes 1,2)  POINTS @  I 2+  C!  ( byte 3)
 12    0  I 3 +  C!  ( any flags go here- in byte 4)
 13    MAX-PEAK 2@  I 2+ 2+  2!  ( bytes 5-8)
 14    #POINTS 1+!  ( one more peak found)
 15    0. MAX-PEAK 2! ;

SCR#3
  0 ( Peak-Finding Algorithm- MJK 5/9/86)
  1 : ?PEAK ( d-)   2DUP  YPREV 2@  D(
  2    IF ( decr.)  +LAST @  0=
  3      IF ( decr.)  POINTS @  PWIDTH @  )
  4        IF ( width ok)  MAX-PEAK 2@  THRESHOLD 2@  D( NOT
  5          IF ( above thld)  !PEAK  THEN
  6        0 POINTS !  0 UPFLAG !
  7        ELSE   POINTS @  0= NOT  IF  POINTS 1+!  THEN   THEN
  8      ELSE ( at top)  UPFLAG @  1 )
  9        IF ( up twice)  YPREV 2@  MAX-PEAK 2@  D( NOT
 10          IF ( new max.)  YPREV 2@  MAX-PEAK 2!
 11                          XVALUE @  STEP @ -  XMAX !  THEN
 12        THEN  0 +LAST !  ( intensity was decr.)  POINTS 1+! THEN
 13    ELSE ( incr.)  +LAST 1+!  UPFLAG 1+!  POINTS 1+!
 14    THEN  YPREV 2! ;
 15
```

```
SCR#4
  0 ( Peak-Finding Algorithm- 8086/8 !PEAK - MJK 5/10/86)
  1 ( !PEAK moves parameters of the new peak to the DATABUFFER)
  2
  3 ASSEMBLER
  4
  5 CREATE   #POINTS W MOV   W SHL W SHL W SHL
  6    DATABUFFER # W ADD   XMAX 1 MOV   W 1 ) MOV   POINTS LDA
  7    0 2 W) MOV   0 # 3 MOV ( flag again)   MAX-PEAK 2+ 1 MOV
  8    1 6 W) MOV   MAX-PEAK 1 MOV   1 4 W) MOV   #POINTS INC
  9    0 0 SUB   MAX-PEAK STA   MAX-PEAK 2+ STA   RET
 10
 11 FORTH
 12
 13
 14
 15

SCR#5
  0 ( Peak-Finding Algorithm in 8086/8 Code- MJK 5/10/86)
  1 ( Note:  This algorithm uses previous variables) ASSEMBLER
  2 CREATE PEAK-up   UPFLAG INC   +LAST INC   POINTS INC   RET
  3 CREATE PEAK-down   POINTS LDA   PWIDTH 0 SUB   0< NOT
  4    IF   MAX-PEAK 2+ 1 MOV   MAX-PEAK 2 MOV  THRESHOLD 1 SUB
  5         THRESHOLD 2+ 2 SBB   0< NOT
  6         IF  !PEAK CALL   THEN
  7         0 0 SUB   POINTS STA   UPFLAG STA
  8    ELSE   -1 #POINTS TEST   0= NOT   IF   POINTS INC   THEN  RET
  9 CREATE PEAK-0   2 # UPFLAG CMP   CS NOT
 10    IF   YPREV 2+ 1 MOV   YPREV 2 MOV   MAX-PEAK 2+ 1 SUB
 11         MAX-PEAK 2 SBB   0< NOT
 12         IF   YPREV LDA   MAX-PEAK STA
 13              YPREV 2+ LDA   MAX-PEAK 2+ STA
 14              XVALUE LDA   STEP 0 SUB   XMAX STA
 15         THEN    THEN   POINTS INC   0 0 SUB   +LAST STA   RET

SCR#6
  0 ( Peak-Finding Algorithm in 8086/8 Code- Cont. - MJK 10/31/86)
  1
  2 CODE ?PEAK
  3    2 POP   1 POP   U PUSH   I PUSH   2 U MOV   1 I MOV
  4    YPREV 2+ 1 SUB   YPREV 2 SBB   0<
  5
  6       IF   -1 # +LAST TEST   0=
  7          IF   PEAK-down CALL
  8          ELSE   PEAK-0 CALL
  9          THEN
 10       ELSE   PEAK-up CALL
 11       THEN
 12
 13    I YPREV 2+ MOV   U YPREV MOV   I POP   U POP
 14    NEXT
 15
```

```
SCR#7
  0 ( Timer for Algorithms )
  1
  2 : TESTER   0 0 ?PEAK  1000 0 ?PEAK  2000 0 ?PEAK
  3    3000 0 ?PEAK  4000 0 ?PEAK  5000 0 ?PEAK  6000 0 ?PEAK
  4    5000 0 ?PEAK  4000 0 ?PEAK  3000 0 ?PEAK  2000 0 ?PEAK
  5    1000 0 ?PEAK  0 0 ?PEAK ;
  6 ( 13 POINTS)
  7
  8
  9 : GO   COUNTER   TESTER   TIMER ;
 10
 11
 12 ( CODE- 215 uS on 5 MHz 8088)
 13 ( CODE- 823 uS on 5 MHz 8088)
 14
 15
```

APPENDIX E.

FORTH CODE FOR ALGORITHM TESTING

```
SCR#1
 0 ( Testing Routines for Peak-Finding Algorithms- MJK 6/30/86)
 1
 2 17 25 MATRIX PDATA   ( creates a 17x25 matrix which can be
 3                        accessed as- row# col# PDATA)
 4
 5 ( RFILL fills a row in PDATA with n elements on the stack)
 6 : RFILL ( n elements, n, row#)   2DUP   0 PDATA ! ( #elements)
 7   0 PDATA ! ( starting addr.)  SWAP 1+  1 DO  DUP ROT SWAP
 8   I 2* +  ( current addr. in matrix) !  LOOP  DROP ;
 9
10 ( Sample Input to Matrix)
11 150 250 570 990 2140 4730 7760 10000 9690 8690 3890
12 770 310 280 ( GOOD PEAK)
13 14 ( #elements)  1 ( row#)  RFILL
14
15

SCR#2
 0 ( More Testing of Algorithms- MJK 6/30/86)
 1
 2 : REFRESH   0 #POINTS ! ;
 3
 4 : ANALYZE ( n-)   DUP @ ( #elements) 0 DO  I XVALUE !
 5   DUP 2+  I 2*  +  @ ( new datum)  0 ( n->d)  ?PEAK
 6   LOOP  DROP ;
 7
 8 : REPORT   CR  #POINTS @  DUP .  ." PEAKS FOUND "  CR 0 DO
 9   ." POINT # "  DATABUFFER  I B/P *  + @  .
10   ." INTENSITY = "  DATABUFFER I B/P *  +  4 +  2@  D.
11   CR LOO ;
12
13 : ANALYZER   REFRESH  ANALYZE  REPORT ;
14
15

SCR#3
 0 ( More Testing of Algorithms- MJK 6/30/86)
 1 ( This version of PRESTATE initializes the algorithm to
 2 a well-behaved starting state- other versions can vary the
 3 starting conditions to really exercise the algorithm)
 4 : PRESTATE   0 0 YPREV 2!  0 +LAST !  0 UPFLAG !
 5   0 0 MAX-PEAK 2! ;
 6
 7 : .PRESTATE   CR  ." YPREV = "  YPREV 2@  D.  CR
 8                   ." +LAST = "  +LAST @   U.  CR
 9                   ." UPFLAG = "  UPFLAG @  U.  CR
10                   ." PWIDTH = "  PWIDTH @  U.  CR ;
11
12 : ALGORITHM   7 3 DO   I PWIDTH ! ( different min. widths)
13   17 0 DO  ( different peak-shapes = rows in matrix)
14   I 0 PDATA   PRESTATE  .PRESTATE  ANALYZER  LOOP  LOOP ;
15
```

APPENDIX F.

FORTH CODE FOR MICROCODE COMPILER

```
SCR#1
  0 ( Microcode Compiler Load Block - MJK 4/28/87)
  1
  2 ' (CREATE)  'CREATE !
  3  2  5   THRU   ( Basic Compiler)
  4  6  7   THRU   ( Instruction Types)
  5  8 13   THRU   ( Instructions)
  6 19 20   THRU   ( Control Structures)
  7 14 18   THRU   ( Movements)
  8 21 23   THRU   ( Error Checking)
  9 26      LOAD   ( Algorithm)
 10 24      LOAD   ( )Disk)
 11 25      LOAD   ( Debugger)
 12 CR CR .( Algorithm has been resident compiled ) CR
 13 .( To compile the actual microcode type UCODE )
 14 ' ?CREATE 'CREATE !
 15

SCR#2
  0 ( Microcode Compiler - MJK 4/28/87 )
  1 CR .( Loading Basic Compiler )
  2   64 DUP CONSTANT WORDLENGTH  ( should be a multiple of 16)
  3     16 / CONSTANT W/W
  4     1024 CONSTANT RAM-DEPTH
  5
  6 CREATE CODE-BUFFER   W/W RAM-DEPTH *  ALLOT
  7  CODE-BUFFER  W/W RAM-DEPTH *   ERASE
  8
  9 ( The following 2 VARIABLES should range from 0 to RAM-DEPTH)
 10 VARIABLE current   ( microinst. currently being compiled)
 11 VARIABLE node 1 ALLOT  ( inst. from which branch occurs)
 12                        ( node 1+ is flag for branching)
 13 VARIABLE available   ( next available virtual address)
 14 VARIABLE start   ( start of repetitive part of algorithm)
 15

SCR#3
  0 ( Microcode Compiler - MJK 4/28/87) HEX
  1
  2 : here ( -a)   current @  W/W *  CODE-BUFFER + ;
  3
  4 : T2/ ( t-t)   OVER 1 AND  )R  D2/  ROT 2/  7FFF AND  R)
  5   IF 8000 OR THEN   ROT ROT ;
  6
  7 : BIT-TRANSLATE ( d n - n1 n2 n3)   0 SWAP  2SWAP  ROT
  8   ?DUP  0= NOT  IF  0 DO  T2/  LOOP  THEN  SWAP  ROT  ;
  9
 10 : &MOVE ( src, dest, cnt - )   0 DO   2DUP  I + @  SWAP I + @
 11   OR   OVER  I + !  LOOP  2DROP ;
 12
 13
 14
 15
```

```
SCR#4
  0 ( Masking of appropriate bits - MJK 4/28/87) HEX
  1 ( This word allow the initialization of code space to something
  2 other than zeroes )
  3
  4 : MASK ( a, bit offset, length)   -1 DUP ROT
  5   ?DUP 0= NOT  IF  0 DO  D2*   SWAP FFFE AND SWAP  LOOP THEN
  6   ROT  ?DUP 0= NOT  IF  0 DO  D2*  SWAP  1 OR  SWAP  LOOP THEN
  7   ROT  >R  I 1+ @  AND  I 1+ !   I  @  AND  R>   !  ;
  8
  9
 10
 11
 12
 13
 14
 15

SCR#5
  0 ( Microcode Compiler - MJK 4/28/87 )
  1 : ENCODE ( startbit length- )   CREATE , ,  DOES> ( d-)
  2   2@  CREATE  DUP , ( length)  OVER  16  MOD  +  16  /MOD
  3   SWAP  IF  1+  THEN  DUP  >R    ,    ( #words required)
  4   ( #wr=len + sb MOD 16  / 16)
  5   16  /MOD           ,      ( sbyte offset )
  6   ( sbyte=sbit/16)   DUP   , ( mask offset)
  7   16  SWAP -   ( bit offset for d in BIT-TRANSLATE
  8                  =  16 - sb 16 MOD )
  9   BIT-TRANSLATE   I
 10   0 DO  , LOOP   3  R> -  0 DO  DROP  LOOP  ( drop useless data)
 11   DOES>   >R  I 4 + ( source)
 12   here  I 2 + @  ( sbyte offset )  +  ( destination)
 13   DUP ( addr)   I 3 + @  ( sbit off)  I  @  ( length)  MASK
 14   R>  1+  @  ( count)   &MOVE  ;
 15

SCR#6
  0 ( Microcode Compiler - MJK  4/28/87)
  1 CR .( Loading Basic Instruction Types)
  2 ( sb      length )
  3    0          8       ENCODE      variables  ( d-)
  4    8         10       ENCODE      NEXT.ADDRESS
  5   18          4       ENCODE      MUX.SELECT
  6   22          3       ENCODE      ?PEAK
  7   25          1       ENCODE      CLKON/OFF
  8   26          5       ENCODE      THRESHOLD
  9   31          1       ENCODE      NEW.DATA
 10   32          2       ENCODE      MAX.PEAK   ( 1DCMP)
 11   34          2       ENCODE      XMAX
 12   36          2       ENCODE      YPREV  ( 2DCMP)
 13   38          2       ENCODE      XPREV
 14   40          2       ENCODE      1FLAG
 15   42          2       ENCODE      2FLAG
```

```
SCR#7
  0 ( Microcode Compiler - MJK  4/28/87 )
  1 ( sb      length )
  2   44          4         ENCODE       1CNTR  ( d-)
  3   48          2         ENCODE       1CCMP
  4   50          4         ENCODE       2CNTR
  5   54          2         ENCODE       2CCMP
  6   56          1         ENCODE       CNT>
  7   57          1         ENCODE       VAR>
  8   58          6         ENCODE       EXPANSION
  9
 10
 11
 12
 13
 14
 15

SCR#8
  0 ( Microcode Compiler - MJK  4/28/87)
  1 CR .( Loading Intructions )
  2 0. variables NADA
  3 1. variables T0BYTE           5. variables 1CMP
  4 2. variables T1BYTE           6. variables 2CMP
  5 3. variables T2BYTE           7. variables 1CNTRLD
  6 4. variables T3BYTE           8. variables 2CNTRLD
  7 ( Condition Codes)
  8 0. MUX.SELECT ?NEWPT        8. MUX.SELECT ?1F/F
  9 1. MUX.SELECT ?YPMAX        9. MUX.SELECT 9MUX
 10 2. MUX.SELECT ?MPMAX       10. MUX.SELECT 10MUX
 11 3. MUX.SELECT ?2CMPMAX     11. MUX.SELECT 11MUX
 12 4. MUX.SELECT ?2CMPEQ      12. MUX.SELECT 12MUX
 13 5. MUX.SELECT ?1CMPMAX     13. MUX.SELECT 13MUX
 14 6. MUX.SELECT ?1CMPEQ      14. MUX.SELECT 14MUX
 15 7. MUX.SELECT ?2F/F        15. MUX.SELECT 15MUX

SCR#9
  0 ( Microcode Compiler - MJK 4/28/87)
  1 0. MAX-PEAK XMAX-PEAK
  2 3. MAX-PEAK !MAX-PEAK
  3 1. MAX-PEAK /MAX-PEAK
  4 2. MAX-PEAK @MAX-PEAK
  5
  6 0. YPREV XYPREV
  7 3. YPREV !YPREV
  8 1. YPREV /YPREV
  9 2. YPREV @YPREV
 10
 11 1. 1FLAG 1CLR          3. 1FLAG 1HLD
 12 2. 1FLAG 1SET
 13 1. 2FLAG 2CLR          3. 2FLAG 2HLD
 14 2. 2FLAG 2SET
 15
```

```
SCR#10
  0 ( Microcode Compiler - MJK 4/28/87)
  1 3. XMAX XMHLD
  2 2. XMAX !XMAX
  3 1. XMAX @XMAX
  4
  5 3. XPREV XPHLD
  6 2. XPREV !XPREV
  7 1. XPREV @XPREV
  8
  9 0. ?PEAK !PEAK
 10 6. ?PEAK NO-PEAK
 11 7. ?PEAK XPEAK
 12
 13 0. NEXT-ADDRESS rst
 14 0. VAR> @VAR
 15 1. VAR> XVAR

SCR#11
  0 ( Microcode Compiler - MJK 4/28/87)
  1 0. CLKON/OFF NOCLK
  2 1. CLKON/OFF ONCLK
  3
  4 HEX
  5 1E. THRESHOLD !0THRESH
  6 1D. THRESHOLD !1THRESH
  7 1B. THRESHOLD !2THRESH
  8 17. THRESHOLD !3THRESH
  9 0F. THRESHOLD @THRESH
 10 1F. THRESHOLD XTHRESH
 11 DECIMAL
 12
 13 0. NEW.DATA @NEW
 14 1. NEW.DATA XNEW
 15

SCR#12
  0 ( Microcode Compiler - MJK 4/28/87)
  1 0. CNT> !CNT
  2 1. CNT> XCNT
  3
  4 HEX
  5 0A. 1CNTR !1CNTR
  6 09. 1CNTR @1CNTR
  7 0B. 1CNTR U1CNTR
  8 03. 1CNTR D1CNTR
  9 0F. 1CNTR X1CNTR
 10 DECIMAL
 11
 12 0. EXPANSION E0FF
 13
 14
 15
```

```
SCR#13
  0 ( Microcode Compiler - MJK 4/28/87 )
  1
  2 HEX
  3 0A. 2CNTR !2CNTR
  4 09. 2CNTR @2CNTR
  5 0B. 2CNTR U2CNTR
  6 03. 2CNTR D2CNTR
  7 0F. 2CNTR X2CNTR
  8 DECIMAL
  9
 10 0. 1CCMP X1CMP          3. 1CCMP !1CMP
 11 2. 1CCMP /1CMP          1. 1CCMP @1CMP
 12
 13 0. 2CCMP X2CMP          3. 2CCMP !2CMP
 14 2. 2CCMP /2CMP          1. 2CCMP @2CMP
 15

SCR#14
  0 ( Microcode Compiler - MJK 4/28/87)
  1 CR .( Loading Movements )
  2
  3 : WAIT   NADA ?NEWPT XMAX-PEAK  XYPREV  1HLD  2HLD  XMHLD
  4   XPHLD  XPEAK  NOCLK  XTHRESH  XNEW  XCNT  X1CNTR  X2CNTR
  5   X1CMP  X2CMP  rst  XVAR  EOFF ;
  6 ( 0 0 C0 FD CC FF 3C 03 - SHOW AFTER CLEAR )
  7
  8 : ALL-QUIET   RAM-DEPTH 0 DO  I current !  WAIT  LOOP ;
  9 CR 2 SPACES .( Initializing CODE Area)
 10  ALL-QUIET
 11
 12
 13
 14
 15

SCR#15
  0 ( Microcode Compiler - MJK 4/29/87)
  1 : ZERO   NADA @VAR ;
  2 : NADA   NADA XVAR ;
  3
  4 : 1CMP   1CMP @VAR ;
  5 : 2CMP   2CMP @VAR ;
  6 : 1CNTRLD   1CNTRLD @VAR ;
  7 : 2CNTRLD   2CNTRLD @VAR ;
  8
  9 : 0THRESHOLD   T0BYTE @VAR   !0THRESH ;
 10 : 1THRESHOLD   T1BYTE @VAR   !1THRESH ;
 11 : 2THRESHOLD   T2BYTE @VAR   !2THRESH ;
 12 : 3THRESHOLD   T3BYTE @VAR   !3THRESH ;
 13 : THRESHOLD   next 0THRESHOLD  next 1THRESHOLD
 14   next 2THRESHOLD  next 3THRESHOLD ;
 15
```

```
SCR#16
  0 ( Microcode Compiler - MJK 4/29/87 )
  1 : !UPTST   2CMP  !2CMP ;
  2 : CLR   ZERO   !1CNTR  !2CNTR  !2CMP  !1CMP  1CLR  2CLR ;
  3 : ZAP-BUS   @THRESH  !MAX-PEAK  !YPREV ;
  4 : INITIALIZE   next  ZERO  next  !UPTST  next  2CMP  THRESHOLD
  5   next  ZAP-BUS  next  @THRESH ;
  6
  7 : INCREASING   U1CNTR  U2CNTR  1SET  NO-PEAK @NEW  !YPREV
  8   next @NEW ;
  9
 10 : @maxima   @MAX-PEAK  @XMAX ;
 11 : PEAK   @maxima  !PEAK   next @maxima ;
 12 : @minima  @THRESH  ZERO ;
 13 : PKRST   ZERO  !1CNTR  @minima !MAX-PEAK   next  @minima ;
 14 : PKSTORE   PEAK  next  ZERO !1CNTR @minima !MAX-PEAK  next
 15   @minima ;

SCR#17
  0 ( Microcode Compiler - MJK 4/29/87 )
  1 : DOWN   1CLR  U2CNTR  NO-PEAK @NEW  !YPREV  next @NEW ;
  2
  3 : @previous  @YPREV  @XPREV ;
  4
  5 : !MAX  @previous  !MAX-PEAK !XMAX  next  @previous ;
  6
  7 : NEWMAX   !MAX next 1CLR U2CNTR NO-PEAK @NEW  !YPREV
  8   next  @NEW ;
  9
 10 : IGNORE   1CLR  NO-PEAK @NEW  !YPREV  next @NEW ;
 11
 12
 13
 14
 15

SCR#18
  0 ( Microcode Compiler - MJK 4/29/87)
  1 ( Aliases)
  2 : @POINTS   @2CNTR ;
  3 : @UPFLAG   @1CNTR ;
  4
  5 : !POINTS   !2CNTR ;
  6 : !UPFLAG   !2CNTR ;
  7
  8 : ?+LAST   ?1F/F ;
  9
 10 : ?LESS-THAN-ONE   ?2CMPMAX ;
 11 : ?TOO-NARROW   ?1CMPMAX ;
 12
 13
 14
 15
```

```
SCR#19
  0 ( Microcode Compiler - MJK  4/29/87)
  1 CR .( Loading Control Structures)
  2 : !PREV.INST ( n n -)     W/W *  CODE-BUFFER +  BYTE   )R
  3   256 /MOD SWAP   I C@  OR  I C!  I 2 + C@  OR  R) 2 + C! ;
  4 : if ( -current)   available @  node @  !PREV.INST  node 1+ @
  5   IF  available @ node @ 1+  !PREV.INST  THEN  available @  DUP
  6   1+  DUP  current !  node !  2 available +!  0 node 1+ ! ;
  7 : else ( current-)   DUP  current ! node !  0 node 1+ ! ;
  8 : then ;
  9 : (next) ( a-)   available @ DUP DUP  current !  ROT DUP
 10   EXECUTE  SWAP 1+ current !  EXECUTE  2 available +!
 11   DUP node @  !PREV.INST  node 1+ @  IF  DUP  node @  1+
 12   !PREV.INST  ( last one was a next- so br needed for both)
 13   THEN  node !  1  node 1+ ! ;
 14 : next   '    \  LITERAL  COMPILE (next)   \  \\  ; IMMEDIATE
 15 : (next) '  (next) ;

SCR#20
  0 ( Microcode Compiler - MJK  4/29/87 )
  1 : HOME   start @  current @  !PREV.INST  node 1+ @
  2   IF  start @  current @  1-  !PREV.INST  THEN ;
  3
  4 : START   current @  DUP node !  1-  DUP DUP  !PREV.INST
  5   start !  0  node 1+ ! ;
  6
  7 : microcode   0 DUP  current !  node !  DUP  EXECUTE
  8   1  node 1+ !  1 current +!  EXECUTE  2 available  !  ;
  9
 10 : MICROCODE  '  \ LITERAL  COMPILE  microcode  \  \\ ; IMMEDIATE
 11
 12
 13
 14
 15

SCR#21
  0 ( Error Checking Routines - MJK 4/29/87)
  1 CR  .( Loading Error Checking Routines)
  2 ( bits- only one of which can be active at any time)
  3 CREATE DB1   3 , 51 , 45 , 57 ,
  4 CREATE DB2   2 , 54 , 48 ,
  5 CREATE YB1   2 , 31 , 30 ,
  6 CREATE YB2   2 , 36 , 32 ,
  7 CREATE XB1   3 , 31 , 39 , 35 ,
  8
  9 : ?BIT ( bit# add - f)    @  SWAP  ?DUP  0= NOT IF
 10   0 DO  2/  LOOP  THEN  1 AND ;
 11
 12 : ?2BITS ( sbit#, a-n)   @  SWAP  ?DUP  0=  NOT   IF
 13   0 DO  2/  LOOP  THEN  3 AND ;
 14
 15
```

```
SCR#22
 0 ( Error Checking- MJK  4/29/87)
 1 VARIABLE '.ERROR
 2 : .ERROR   '.ERROR @EXECUTE ;
 3 VARIABLE Errors   0 Errors !
 4
 5 : .DB  CR  ." Too Many Devices Driving the DB Bus in Instruction
 6  # "  current @ U.  1 Errors +! ;
 7 : .YB  CR ." Too Many Devices Driving the YB Bus in Instruction
 8 # " current @ U.  1 Errors +! ;
 9 : .XB  CR  ." Too Many Devices Driving the XB Bus in Instruction
10  # " current @ U.  1 Errors   +! ;
11
12
13
14
15

SCR#23
 0 ( Error Checking - MJK  4/29/87)
 1 : 1CHK   0  OVER  @   1+  1  DO   OVER  I +  @ ( bit of interes)
 2   16 /MOD   here +  ( word of interest)   ?BIT  0=  +  LOOP
 3   SWAP DROP \\ ;
 4
 5 : 2CHK ( n-)   OVER @  1+  1 DO  OVER  I +  @ ( selected bits)
 6   16 /MOD   here +  ( word of interest)  ?2BITS  2 =  +   LOOP
 7   SWAP  DROP \\ ;
 8
 9 : CHK ( 2's 1's -)   1CHK  2CHK  1 >  IF  .ERROR THEN ;
10
11 : CHECK  ['] .DB  '.ERROR !  DB2  DB1  CHK
12          ['] .YB  '.ERROR !  YB2  YB1  CHK
13          ['] .XB  '.ERROR !  XB1 1CHK  1 >  IF  .ERROR THEN ;
14 : UCHECK  RAM-DEPTH  0 DO  I current ! CHECK LOOP ;
15

SCR#24
 0 ( Disk Storage of Algorithm - MJK  4/29/87)
 1 ( In downloading we load byte 0's first, then byte 1's etc.
 2   However, in memory we have word 1 -byte 0-7, word 2, byte 0-7,
 3   etc.  This word rearranges the bytes for later downloading.)
 4
 5 5360 CONSTANT CODE-FILE
 6
 7 : UCODE>DISK   W/W 0 DO  RAM-DEPTH 0 DO
 8   CODE-BUFFER  I  W/W *  +  J +  @  256  /MOD DROP
 9   J 2*  RAM-DEPTH *  I +  1024 /MOD  CODE-FILE +  BLOCK BYTE +
10   C!  LOOP   UPDATE  LOOP  FLUSH  W/W 0 DO  RAM-DEPTH 0 DO
11   CODE-BUFFER I W/W *  +  J +  @  256  /MOD  SWAP  DROP
12   J 2* 1+ RAM-DEPTH *  I +  1024 /MOD  CODE-FILE + BLOCK  BYTE +
13   C!  LOOP   UPDATE   LOOP  FLUSH ;
14
15   : UCODE  ALGO  UCHECK  Errors @  0=  IF  UCODE>DISK  THEN ;
```

```
SCR#25
 0 ( DEBUGGERS)
 1
 2 : SHOW ( n-)   W/W *  CODE-BUFFER +  W/W DUMP ;
 3
 4 : CLEAR ( n-)  W/W *  CODE-BUFFER +  W/W   ERASE ;
 5
 6 : EXPOSE ( -n)   0 DO   I SHOW  2 +LOOP ;
 7
 8
 9
10
11
12
13
14
15

SCR#26
 0 ( Algorithm - MJK  7/31/87)
 1 : ALGO   MICROCODE  CLR  INITIALIZE  next  ?NEWPT  START
 2     if   @NEW  ?YPMAX
 3      if   ?+LAST
 4       if   @UPFLAG  ?LESS-THAN-ONE
 5     if   DOWN HOME
 6     else   @YPREV  ?MPMAX
 7       if   DOWN HOME  else  NEWMAX HOME  then then
 8        else   @POINTS  ?TOO-NARROW
 9          if   @POINTS  ?LESS-THAN-ONE
10            if   IGNORE HOME   else  DOWN HOME   then
11          else   @THRESH  ?MPMAX
12            if   PKSTORE HOME  else  DOWN HOME  then
13          then  then
14       else  INCREASING HOME  then
15     else  HOME then  ;
```

APPENDIX G.

FORTH CODE FOR AMD9513 COUNTER

```
SCR#1
  0 ( Dual-Mode Control- Slave 3- MJK 2/15/85)
  1
  2 VARIABLE 3LD ( Significant Count)  COMMAND
  3 VARIABLE 4LD ( Significant Time)  COMMAND
  4 VARIABLE 5LD ( Window Count)  COMMAND
  5 VARIABLE CDIV ( Count Divisor)  COMMAND
  6 VARIABLE FDIV ( Frequency Divisor)  COMMAND
  7 VARIABLE CTMODE ( Time or Count)  COMMAND
  8
  9
 10
 11
 12
 13
 14
 15

SCR#2
  0 ( 9513 Counter Utilities) HEX
  1 AMD9513 1+ CONSTANT COMMAND-REGISTER
  2
  3 : 95data ( n-)   AMD9513 C! ;
  4 : 95com ( n-)   COMMAND-REGISTER C! ;
  5
  6 : NZERL ( n-)   0 DO  00 95data  LOOP ;
  7
  8 CODE GRAB ( n-)   0 POP   COMMAND-REGISTER STA B
  9    AMD9513 0 MOV B   AMD9513 0 HI MOV B   0 PUSH   NEXT
 10
 11 CODE W)9513 ( n-)   0 POP   2 2 SUB   0 2 MOV B
 12    2 AMD9513 MOV B   0 HI 2 MOV B   2 AMD9513 MOV B   NEXT
 13
 14 DECIMAL
 15

SCR#3
  0 ( This is the 9513 Counting Scheme for Constant Count)
  1 HEX
  2 : CCLOAD   01 95com  ( counter autoload)
  3 ( 1CTR)  A8 95data  AB 95data  4 NZERL
  4 ( 2CTR)  28 95data  00 95data  4 NZERL
  5 ( 3CTR)  02 95data  02 95data  3LD @ W)9513  2 NZERL
  6 ( 4CTR)  02 95data  0B 95data  4LD @ W)9513  2 NZERL
  7 ( 5CTR)  02 95data  01 95data  5LD @ W)9513  2 NZERL
  8 07 95com  4 NZERL ( Alarm Register)
  9 20 95data  CDIV C@ 95data  ( Master Mode) ;
 10
 11 : COUNT   5F 95com  E5 95com  E4 95com  EB 95com  3F 95com ;
 12
 13 : DATUM   9F 95com  11 GRAB  12 GRAB ;
 14 DECIMAL
 15
```

```
SCR#4
 0 ( This is the 9513 Counting Scheme for Constant Time)
 1 HEX
 2 : CTLOAD   01 95com  ( counter autoload)
 3 ( 1CTR)  A8 95data  A1 95data  4 NZERL
 4 ( 2CTR)  28 95data  00 95data  4 NZERL
 5 ( 3CTR)  02 95data  02 95data  3LD @ W)9513  2 NZERL
 6 ( 4CTR)  02 95data  0B 95data  4LD @ W)9513  2 NZERL
 7 ( 5CTR)  02 95data  FDIV C@ 95data  5LD @ W)9513  2 NZERL
 8 07 95com  4 NZERL ( Alarm Register)
 9 20 95data  81 95data  ( Master Mode) ;
10
11
12 : 95RESET   FF 95com ;
13
14 DECIMAL
15
```

APPENDIX H.

FORTH CODE FOR DUAL-MODE SCANNING

```
SCR#1
  0 ( Dual-Mode Scanning- MJK 2/16/86)
  1 : PULSES ( -d)   COUNT ( Start counting in selected mode)
  2    BEGIN  PIO C@  4 AND  4 = END ( Check for end of count)
  3    DATUM ( Acquire 32-bit count of pulses and time) ;
  4
  5 ( Dual-Mode Interrupt)
  6 HEX ASSEMBLER HERE
  7 0 POP   0 POP   0 POP   0 POP  ( clean up int args)
  8 0 0 SUB   0 PUSH   0 PUSH ( null data)
  9 STI ( enable interrupts)   ' EXIT JMP ( next word = PMODIFY)
 10 OD INTERRUPT
 11 DECIMAL
 12 CREATE DMINT ( holds interrupt goodies)
 13 , , ,  ( value, value, addr)
 14 : !DMINT   DMINT 2+ 2@  DMINT @ 2! ;
 15

SCR#2
  0 ( Flux Calculations - MJK 6/18/86)  HEX
  1 3E7A  D752  9ABC  AF48  LCONSTANT  9513CLK
  2 DECIMAL
  3 : CFLUX ( d-d)   CDIV C@  ?DUP  0= IF  16  THEN
  4    )N  5LD @  )N  F*  ( #counts)  9513CLK  2)N  F*
  5    ( time)  F/  ( flux)  2N) ;
  6
  7 : TFLUX ( d-d)   2)N ( count)  FDIV C@  11 =
  8    IF  1  ELSE  1000  THEN  )N  5LD @  )N  F*
  9    9513CLK  F* ( time)  F/ ( flux)  2N) ;
 10
 11 : FLUX   CTMODE @  IF  TFLUX  ELSE  CFLUX  THEN ;
 12
 13
 14
 15

SCR#3
  0 ( Normalization Routines for Dual-Mode- MJK 6/18/86)
  1
  2    LVARIABLE RES  COMMAND
  3 (   +120.0E-09 RES 1! ( calculated 6/86- MJK )
  4    LVARIABLE DCAL
  5
  6 : RETROFIT ( d-f)   2)N  RES 1@  4.0  F* F*  ( 4ac)
  7    1.0  FSWAP  F-  ( 1-4ac)  FSQRT  1.0  FSWAP  F-
  8    2.0  RES 1@  F*  F/ ;
  9
 10
 11 : PMODIFY ( d-d)   FLUX  RETROFIT  2N) ;
 12 : AMODIFY ( d-d)   2)N  DCAL 1@  F*  2N) ;
 13
 14
 15
```

```
SCR#4
  0 ( Dual-Mode Calibration- MJK 6/18/86)  HEX
  1 : DMACQUIRE ( d-d)   COUNT  TEST-ACQUIRE  BEGIN  PIO C@
  2    4 AND  4 = END  DATUM ;
  3
  4 ( DSLOPE gives the slope of the pulses versus A-to-D curve)
  5 : DSLOPE ( d d - F)   RETROFIT  2)N  F/ ;
  6
  7 : DCALIBRATE   5 0 DO  DMACQUIRE  DSLOPE  LOOP
  8    F+  F+  F+  F+  5  )N  F/  DCAL 1! ; COMMAND
  9
 10 F386  402D SCONSTANT ELD  DECIMAL
 11 VARIABLE pulses  2 ALLOT
 12 : POISSON   YOK  BEGIN  ?XOK  PULSES  2DUP  pulses 2@
 13    D(  NOT IF  pulses 2!  ELSE  2DROP  THEN  YOK
 14    SSTAT 1+ C@  1 AND  1 =  END  1.0  ELD  )N  pulses 2@  2)N
 15    F*  F/  RES 1! ;  COMMAND

SCR#5
  0 ( Dual-Mode Control Scheme- MJK 2/15/86)
  1 CODE ANALOG   ACQUIRE   NEXT
  2
  3 : DUAL-ACQUIRE   -1 PRORST C! ( clr flag and then read)
  4    PIO C@  2 AND  IF  ANALOG  AMODIFY
  5                   ELSE  PULSES PMODIFY
  6                   THEN
  7                   )PEAK ;
  8 : DUAL-SCAN   CTMODE @  IF  CTLOAD  ELSE  CCLOAD  THEN  YOK
  9    BEGIN
 10            ?XOK ( wait for Ion Path)
 11            DUAL-ACQUIRE ( data-point)
 12            YOK ( send ok to Ion Path)
 13    SSTAT 1+ C@  1 AND  1 =  END
 14    1 STATOUT C! ; COMMAND
 15

SCR#6
  0 ( Master Debugging Additions)
  1
  2 HEX F880 CONSTANT AMD9513
  3     F8C0 CONSTANT PIO
  4
  5 : COMMAND ;
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
SCR#7
  0 ( Matrices for Control of Pulse Counting Modes- MJK 2/4/86)
  1
  2 ( CDIV, COUNT)
  3 CREATE Count-Control   ( 0) 0 C, 62500 ,  ( 1) 10 C, 50000 ,
  4    ( 2) 10 C, 10000 ,  ( 3) 1 C, 40000 ,  ( 4)  1 C, 17800 ,
  5    ( 5)  1 C, 10000 ,  ( 6) 1 C,  5000 ,  ( 7)  1 C,  1000 ,
  6    ( 8)  1 C,   400 ,  ( 9) 1 C,   178 ,  ( 10) 1 C,   100 ,
  7
  8 ( FDIV, COUNT)
  9 CREATE Time-Control   ( 0) 11 C,  1000 ,  ( 1) 11 C,  2000 ,
 10    ( 2) 11 C,  4000 , ( 3) 11 C, 10000 ,  ( 4) 11 C, 20000 ,
 11    ( 5) 11 C, 40000 , ( 6) 14 C,   100 ,  ( 7) 14 C,   200 ,
 12    ( 8) 14 C,   400 , ( 9) 14 C,  1000 ,  ( 10) 14 C, 2000 ,
 13    ( 11) 14 C, 4000 , ( 12) 14 C, 10000 ,
 14
 15

SCR#8
  0 ( Dual-Mode Setting Routine for Master- MJK 2/4/86)
  1
  2 CVARIABLE CDIV
  3 CVARIABLE FDIV
  4 VARIABLE 5LD
  5 VARIABLE 4LD
  6 VARIABLE 3LD
  7
  8 : ?PULSE-RATE ( a-n c)   CR CR ." Rate to Use: "  #INPUT 3 * +
  9    DUP 1+ @  SWAP C@ ;
 10
 11 : CCVARS   Count-Control  ?PULSE-RATE  CDIV C!  5LD ! ;
 12 : CTVARS   Time-Control  ?PULSE-RATE  FDIV C!  5LD ! ;
 13
 14
 15

SCR#9
  0 ( Dual-Mode Setting Scheme for Master- MJK 2/4/86)
  1
  2 10 CONSTANT 95MHZ
  3 VARIABLE CTMODE
  4
  5 : ?SCHEME   PAGE  10 0 CURSOR  ." DUAL MODE SETTING ROUTINE "
  6    CR CR  ." Do You Want Constant Time (T) "  CR
  7    ." or Constant Precision (no. of counts) (C) ? "
  8    KEY  84 =  DUP CTMODE !  IF  7923 SCREEN CTVARS
  9    ELSE  7924 SCREEN CCVARS  THEN ;
 10
 11 : ?WINDOW   7922 SCREEN  ." Time Window (usec): "
 12    #INPUT  95MHZ *  4LD !  CR
 13    ." Counts to Occur in Window: "  #INPUT  3LD ! ;
 14
 15
```

```
SCR#10
  0 ( Dual-Mode Setting Scheme for Master- MJK 2/4/86)
  1 3LABEL FDIV fdiv      3LABEL CDIV cdiv
  2 3LABEL 5LD 5ld        3LABEL 4LD 4ld
  3 3LABEL 3LD 3ld        3LABEL CTMODE ctmode
  4
  5 : DMPARAMS   FDIV C@ fdiv IC!  CDIV C@ cdiv IC!  5LD @ 5ld I!
  6    4LD @ 4ld I!  3LD @ 3ld !  CTMODE @ ctmode I! ;
  7
  8
  9
 10
 11
 12
 13
 14
 15

SCR#11
  0 ( Dual-Mode Resolution Determination- MJK 8/12/86)
  1 3LABEL RES RES
  2 LVARIABLE DMRES  +120.0E-09 DMRES 1!
  3
  4 : IL@ ( a3,a1-)   3 0 )SLAVE  16 IPMOVE ;
  5
  6 : IL! ( a1,a3-)   0 3 )SLAVE  16 IPMOVE ;
  7
  8 : POISSON   L2 'CURRENT @  'START @  'END @  'STEP @
  9    1 'STEP !  0 'START !  250 'END !  statCLR  @PARAMS
 10    #DEVICE @ 1PUSH  SL3 POISSON  SL1 SWEEP
 11    'STEP !  'END !  'START !  'CURRENT !  DMRES RES IL@ ;
 12
 13 DMRES RES IL!
 14
 15

SCR#12
  0 ( Dual-Mode Setting Scheme- MJK 2/4/86)
  1 : DUAL-SCAN   SL3  DUAL-SCAN ;
  2 : ASCAN   SL3 SCAN ;
  3
  4 : .ANALOG   .status  12  .MNAME ;
  5 : .DUAL   .status  13 .MNAME ;
  6
  7 : DMCALIBRATE   RRR  L2 'CURRENT @  130 SET  SL3 DCALIBRATE
  8    SET  DRR ;
  9
 10 : DUAL-MODE   ?SCHEME  ?WINDOW  DMPARAMS
 11    ['] DUAL-SCAN 'ACQSCAN !  .DUAL  CR ;
 12
 13 : ANALOG   ['] ASCAN  'ACQSCAN !  .ANALOG  CR ;
 14
 15
```

```
SCR#13
  0 Null Data Window Control
  1
  2 Studies ( e. g. Darland, Enke, Leroi ) have shown that about
  3 two-thirds of the time spent in data acquisition is spent
  4 acquiring null data ( data with only negative information ).
  5 The Null Data Window Control serves two puposes in Dual-Mode
  6 Detection.  First, it can drastically shorten the time spent in
  7 collecting null data, sicne it is often evident very quickly
  8 that no significant ion current exists at a given point.  The
  9 user can set the point to determine whether or not this looks
 10 like a null data point.  Secondly, it prevents the detection
 11 scheme from "hanging up" in Constant Count Mode, when there are
 12 no counts at all.
 13
 14
 15

SCR#14
  0 Dual-Mode Setting Routine - Constant Time
  1 For Pulse Counting Only
  2 Rate       Time/Pt
  3    0           100
  4    1           200
  5    2           400 usec
  6    3             1 msec
  7    4             2
  8    5             4
  9    6            10
 10    7            20
 11    8            40
 12    9           100
 13   10           200
 14   11           400 msec
 15   12             1 sec

SCR#15
  0 Dual-Mode Setting Routine - For Constant Count - Precision
  1 For Pulse Counting Only
  2 Rate       No. of Counts       Precision
  3    0          1 x 10E6           0.100
  4    1          5 x 10E5           0.140
  5    2          1 x 10E5           0.320
  6    3          4 x 10E4           0.500
  7    4       1.78 x 10E4           0.750
  8    5          1 x 10E4           1.000
  9    6          5 x 10E3           1.400
 10    7          1 x 10E3           3.200
 11    8          4 x 10E2           5.000
 12    9       1.78 x 10E2           7.500
 13   10          1 x 10E2          10.000
 14
 15
```

APPENDIX I.

FORTH CODE FOR REACTION SCANS

```
SCR#1
  0 ( Variables for Reaction Scanning- MJK 8/6/85)
  1
  2 VARIABLE VTRAP   20.0 VTRAP !   ( Voltage on L5 to trap ions)
  3 VARIABLE TSTORE  1000 TSTORE !  ( Time to store ions in MS)
  4 VARIABLE VPULSE -100.0 VPULSE ! ( Pulsing voltage on L5)
  5 VARIABLE #ACQS   100 #ACQS !    ( Number of acqs in pulse)
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15

SCR#2
  0 ( Reaction Scanning Primitives- MJK 8/6/85)
  1
  2 : RACQUIRE   #DEVICE @
  3    L5 'CURRENT @  VTRAP @ SET  TSTORE @ MS  VPULS5 @ SET
  4    0.  #ACQS @ 0 DO  ACQUIRE DMAX  LOOP  >R >R  L4 SET
  5    #DEVICE !  R> R> ;
  6
  7 : (RPSCAN)   Q1INIT  +Q1  10 MS
  8    0 DO  RACQUIRE  ?PEAK  +Q1  RSYNC  1 /LOOP ;
  9
 10 : (RDSCAN)   Q3INIT  +Q3  10 MS
 11    0 DO  RACQUIRE  ?PEAK  +Q3  RSYNC  1 /LOOP ;
 12
 13 : (RNSCAN)   DUP  Q1INIT  Q3INIT  NSET  +Q13  10 MS
 14    0 DO  RACQUIRE  ?PEAK  +Q13  RSYNC  1 /LOOP ;
 15

SCR#3
  0 ( Reaction Scanning- MJK 8/6/85)
  1 : RPSCAN   M3 DATAOUT  DRD SCANINIT  M1 @PARAMS  3>DAC
  2    #STEPS  (RPSCAN)  8 SCANEND ;
  3
  4 : RDSCAN   M1 DATAOUT  DRD SCANINIT  M3 @PARAMS  3>DAC
  5    #STEPS  (RDSCAN)  9 SCANEND ;
  6
  7 : RNSCAN   DRD SCANINIT  M1 @PARAMS  3>DAC
  8    #STEPS  (RNSCAN)  10 SCANEND ;
  9
 10 : (RSWEEP) ( #steps, startdac-)   !DATA  DUP #POINTS !
 11    0 DO  0 VALUE @  I 'X 2!  RACQUIRE  I 'Y 2!
 12    +DEVICE  RSYNC LOOP ;
 13
 14 : RSWEEP   FSTOP  STEP @  #DEVICE @ SWEEPDEV !  0. MAX-INTEN 2!
 15    0. TIC 2!  @PARAMS >DAC&STEP  (RSWEEP)  5 SCANEND ;
```

```
SCR#4
  0 ( User Interface for the Reaction Scans- MJK 8/14/85)
  1 4 8040 7 LABELS RXNNAME
  2
  3 : .RXNNAME ( n-)   RXNNAME  )TYPE ;
  4
  5 : NORML   12 U.R ;
  6 : VTGE   6 SPACES  N.1 ;
  7
  8 :CASE RXNVAR   VTRAP  TSTORE  VPULSE  #ACQS ;
  9 :CASE .RVALUE   VTGE  NORML  VTGE  NORML ;
 10
 11 : .RXSET   PAGE 8041 SCREEN  8040 LOAD
 12    4 0 DO  CR  I .RXNNAME  I RXNVAR @  I .RVALUE  LOOP
 13    20 SPACES ;
 14 : RXSET   .RXSET  4 0 DO  20 I + 23 CURSOR  63 EMIT
 15    8 EMIT  #INPUT  ?DUP  IF  DUP I RXNVAR !  THEN  LOOP ;

SCR#5
  0 ( Reaction Scanning Primitives- MJK 8/6/85 )
  1 : +RACQUIRE   #DEVICE @
  2    L5 'CURRENT @  VTRAP @ L5 SET  TSTORE @ MS  VPULSE @ SET
  3    0. #ACQS @ 0 DO  ACQUIRE  D+  LOOP  )R )R  L5 SET
  4    #DEVICE !  R) R) ;
  5
  6 : (+PSCAN)   Q1INIT  +Q1  10 MS
  7    0 DO  +RACQUIRE  ?PEAK  +Q1  RSYNC  1 /LOOP ;
  8
  9 : (+DSCAN)   Q3INIT  +Q3  10 MS
 10    0 DO  +RACQUIRE  ?PEAK  +Q13  RSYNC  1 /LOOP ;
 11
 12 : (+NSCAN)   DUP Q1INIT Q3INIT  NSET  +Q13  10 MS
 13    0 DO  +RACQUIRE  ?PEAK  +Q13  RSYNC  1 /LOOP ;
 14
 15

SCR#6
  0 ( Reaction Scanning- MJK 8/6/85 )
  1 : +RPSCAN   M3 DATAOUT  DRD SCANINIT  M1 @PARAMS  3)DAC
  2    #STEPS  (+PSCAN)  8 SCANEND ;
  3
  4 : +RDSCAN   M1 DATAOUT  DRD SCANINIT  M3 @PARAMS  3)DAC
  5    #STEPS  (+DSCAN)  9 SCANEND ;
  6
  7 : +RNSCAN   DRD SCANINIT  M1 @PARAMS  3)DAC  #STEPS  (+NSCAN)
  8    10 SCANEND ;
  9
 10 : (+SWEEP) ( #steps startdac-)   !DATA  DUP #POINTS !
 11    0 DO  0 VALUE @  I 'X 2!  +RACQUIRE  I 'Y 2!
 12
 13 : +RSWEEP   FSTOP  STEP @  #DEVICE @ SWEEPDEV !  0. MAX-INTEN 2!
 14    0. TIC 2!  @PARAMS )DAC&STEP  (+SWEEP)  5 SCANEND ;
 15
```

```
SCR#7
 0 ( MRM for Reaction Scans- MJK 3/23/86 )
 1 : RCOLLECT ( cycle#-)   #RXNS @  0 DO
 2    I  DUP MASS-SET  5 MS  RACQUIRE
 3       I  5 <  IF  4DUP
 4       NORMALIZE  ?LOG  SWAP  FIELD# !  SWAP 1000 MOD
 5       DOT PLOT  THEN   ROT 'Y 2!  LOOP  DROP
 6    MSSLOT   !RXREC ;
 7
 8
 9
10
11
12
13
14
15

SCR#8
 0 ( MRM with the Reaction Scans- MJK 3/23/87)
 1 : R-MRM   DRD  (RATE) @ >R  RXINIT  0 DO  I DUP  ?RESET
 2    RCOLLECT  ?LEAVE  RSYNC  LOOP  TERMINAL  2TIMER
 3    ." elapsed"  CR ." MRM complete"  RXEND  CR BELL
 4    R> RATE ;
 5
 6 : R-1SIM   DRR  (RATE) @ >R  RXINIT 0 DO  I DUP  ?RESET
 7    RCOLLECT  ?LEAVE  RSYNC  LOOP  TERMINAL  2TIMER
 8    ." elapsed"  CR ." MRM complete"  RXEND  CR BELL
 9    R> RATE ;
10
11 : R-3SIM   RRD  (RATE) @ >R  RXINIT 0 DO  I DUP  ?RESET
12    RCOLLECT  ?LEAVE  RSYNC  LOOP  TERMINAL  2TIMER
13    ." elapsed"  CR ." MRM complete"  RXEND  CR BELL
14    R> RATE ;
15

SCR#9
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
SCR#1
  0 CR CR {   VARIABLE     VALUE}
  1 CR
  2 EXIT
  3
  4   VTRAP  TSTORE  VPULSE   #ACQS
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15

SCR#2
  0 REACTION SCAN EDITOR
  1
  2 This allows one to set the extra parameters available in the
  3 reaction scans.  In a reaction scan, LENS 5 is held positive for
  4 a user-specified period of time in order to store up + ions.
  5 The lens is then pulsed negative, pulling the resulting ions
  6 out of quadrupole 2.  A user-specified number of acquisitions
  7 are then scanned for the highest value (in RSCANS) or are
  8 integrated (in +RSCANS).  RSCANS seem to give the highest
  9 signal-to-noise ratio.  VTRAP is the voltage used to store up
 10 the ions in the collision chamber.  TSTORE is the amount of time
 11 in milliseconds for which the ions are stored.  VPULSE is the
 12 voltage used to pulse the ions out.  #ACQS is the number of
 13 acquisitions made for each pulse (each datum).  Note:  Each
 14 acquisition is made up of a number of averages (set by RATE).
 15 It is best to use 2 rate to catch the maximum.

SCR#3
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```