A COMPUTERIZED DATA ACQUISITION AND CONTROL SYSTEM FOR FABRY-PEROT INTERFEROMETRY

SHIDONG TONG, M.Sc.

National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

A Computerized Data Acquisition and Control System for Fabry-Perot Interferometry

© by

Shidong Tong, M. Sc. (Central China Normal Univ.)

A thesis submitted to the School of Graduate
Studies in partial fulfillment of the
requirements for the degree of
Master of Science

Department of Physics
Memorial University of Newfoundland
July 6, 1990

St. John's Newfoundland

ISBN 0-612-08486-8

Abstract

Fabry-Perot interferometry is one of the most powerful techniques in high resolution spectroscopy, especially in Raman and Brillouin scattering. Proper operation of a Fabry-Perot interferometer presumes the Fabry-Perot plates to be maintained in parallel alignment for a reasonable period of time while the experiment is in progress. Misalignment and laser frequency drift can cause very serious losses of finesse, and therefore of resolving power and contrast.

Various methods of stabilizing the interferometer system have been reported previously. May *et al.* described a data acquisition and stabilization system for the Fabry-Perot interferometer which automatically compensates for all instrumental frequency drifts, maintains interferometer alignment, and permits extended data acquisition time in selected spectral regions of interest. The commercial version of this data acquisition system is known as DAS–1. It was originally produced by Burleigh Instruments, Inc. in 1976 and has been the accepted standard among commercially available instruments. However it has not been available for several years.

Hardware and software packages which perform multichannel scaling (MCS) are now commercially available from several sources. The ACE–MCS package as supplied by EG&G–ORTEC and used in the present work consists of an IBM PC plug-in card and an MCS emulation program. This ACE-MCS package is simple and easy to use, but it lacks some essential features required in Fabry-Perot interferometry. The main problem is that it does not permit active feedback control of the interferometer

alignment. In addition, it does not provide for a segmented ramp-voltage output, which is necessary in situations where the scattered light is very weak.

In order to adapt this MCS package for Fabry-Perot interferometry, it was consequently necessary to modify and augment the hardware and software to accommodate the following functions:

- Drift stabilization
- Finesse optimization
- Segmented ramp scanning

The drift stabilization and finesse optimization functions mainly required software development, while segmented scanning required hardware development in the form of a digital electronic circuit.

A prototype system, based on an IBM-XT clone, has been developed at relatively modest cost and has been tested to record Brillouin scattering spectra of fused quartz. The experimental results show that the system is able to maintain Fabry-Perot alignment as well as the original Burleigh DAS–1 system.

Acknowledgements

I would like to express my sincere appreciation and gratitude to my supervisors Dr. M. J. Clouter and Dr. H. Kiefte for their excellent guidance, valuable help, and abundant encouragement at every stage of this research work and in the preparation of this thesis.

Special thanks are also extended to Dr. J. Żuk for his assistance in performing the Brillouin experiments.

The cooperation of EG&G ORTEC (Oakridge, Tennessee) in providing the source code for the MCS program is gratefully acknowledged.

I also wish to acknowledge the financial support from Memorial University of Newfoundland in the form of a Fellowship and Teaching/Research Assistantship during my graduate studies.

Finally, I would like to thank my wife, Siyuan, for her understanding and constant encouragement.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Brillouin Scattering

Light scattering spectroscopy has been found to be a powerful tool for the investigation of molecular structures. When a beam of monochromatic light falls on an atomic or molecular sample, a small fraction of the light is scattered in all directions. The scattering can be divided into three types; they are the (quasi-elastic) Rayleigh scattering, and the inelastic Raman and Brillouin contributions. It is only the latter, frequency-shifted components that are of interest here, and Brillouin scattering is emphasized as being most relevant for this project.

Brillouin scattering can be described classically as the scattering of light by sound waves. The thermal motion of molecules creates regions of compression and rarefaction resulting in fluctuations in the density of a medium. These variations propagate through the medium as sound waves. Since these density variations produce corresponding changes in the refractive index, the sound wave may be thought of as a three-dimensional diffraction grating moving at the sound velocity through the

medium. Incident light waves are reflected from this grating according to the Bragg condition. Since the incident light is reflected by the sound wave, or a grating which is moving at the velocity of the sound, the frequency of the reflected light is shifted due to the Doppler effect. It is readily shown that the amount of the frequency shift is equal to the frequency of the sound waves which are effective in the scattering along the particular direction of observation, and is given by

$$\Delta\nu = \pm\nu_s = \pm\frac{2nv_s}{\lambda}\sin\frac{\theta}{2} \tag{1.1}$$

where n is the mean refractive index of the medium, λ the wavelength of the incident light, v_s the velocity of the sound waves in the medium, and θ the scattering angle.

Quantum mechanically, the event is considered as interactions between incident photons and phonons in a medium. An incident photon of frequency ν_o and wavevector $\mathbf{k}_o$ interacts with a phonon of frequency ν_s and wavevector $\mathbf{k}_s$ resulting in a scattered photon of frequency ν and wavevector $\mathbf{k}$, as shown in Fig. 1.1.

The conservation laws of the energy and momentum for this interaction give:

$$h\nu_o = h\nu \pm h\nu_s \tag{1.2}$$

$$\hbar\mathbf{k}_o = \hbar\mathbf{k} \pm \hbar\mathbf{k}_s \tag{1.3}$$

It is easily seen from Eq. (1.2) that the frequency difference between the incident photon and the scattered photon $\Delta\nu$ is equal to the frequency of the phonon ν_s. Since the momentum of the phonon k_s is much less than that of the photon k_o, therefore

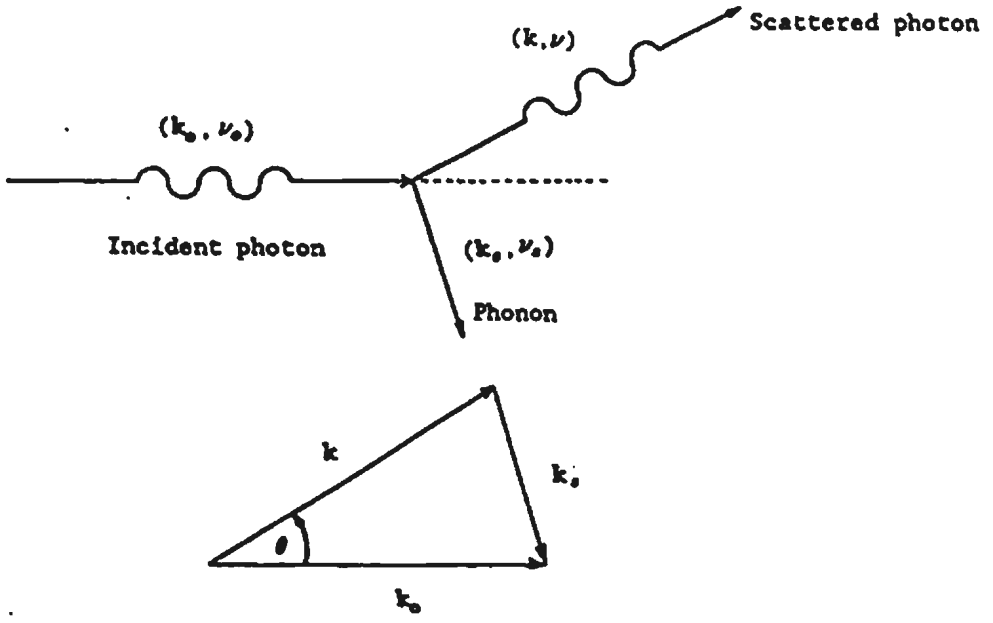


Figure 1.1: Brillouin scattering process of a photon by a phonon

$|\mathbf{k}| \approx |\mathbf{k}_0|$. From Fig. 1.1 we can write

$$k_s = 2k \sin \frac{\theta}{2}. \tag{1.4}$$

Substituting $k_s = 2\pi\nu_s/v_s$ and $k = 2\pi n\nu/c$, where c is the velocity of light, we obtain the Brillouin equation

$$\Delta\nu = \pm\nu_s = \pm\frac{2nv_s\nu}{c} \sin \frac{\theta}{2}. \tag{1.5}$$

In general, for a crystalline substance, there are three types of acoustic waves, one longitudinal and two transverse, giving rise to three frequency-shifted components in a typical Brillouin spectrum.

1.2 The Purpose of the Present Work

In the field of laser light scattering spectroscopy the most widely used instrument for detailed analysis of the frequency shifted components is the Fabry-Perot interferometer. The Fabry-Perot interferometer is able to achieve a very high resolving power and is particularly well suited for resolving the Brillouin components from the Rayleigh peak. A simplified arrangement for a light scattering experiment is shown in Fig. 1.2.

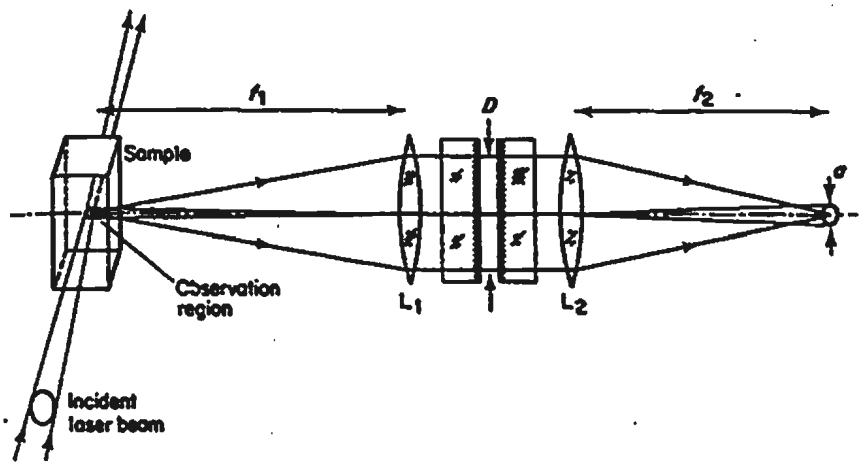


Figure 1.2: A simplified arrangement for a light scattering experiment

Proper operation of the Fabry-Perot interferometer presumes the Fabry-Perot plates to be maintained in parallel alignment during the entire course of the experiment. Misalignment and laser frequency drift can cause very serious losses of finesse, and therefore of resolving power and contrast.

Various methods of stabilizing the interferometer system have been reported previously. A data acquisition and stabilization system for the Fabry-perot interferometer

has been described by May *et al.*[1] This system automatically compensates for all instrumental drifts, maintains interferometer alignment, and permits extended data acquisition time in selected spectral regions of interest. The commercial version of this data acquisition system is known as DAS–1. It was originally produced by Burleigh Instruments, Inc.[2] in 1976. However it has not been available for several years.

Hardware and software packages which perform multichannel scaling (MCS) are now commercially available from several sources. The ACE–MCS package as supplied by EG&G–ORTEC[3] and used in the present work consists of an IBM PC plug-in card and an MCS emulation program. The MCS emulation program is written in Microsoft Pascal and 8086/8088 Assembly Language. The MCS card can continue to acquire data while the IBM PC performs other tasks. This ACE-MCS package is simple and easy to use, but it lacks some essential features required in Fabry-Perot interferometry. The main problem is that it does not permit active feedback control of the interferometer alignment. In addition, it does not provide for a segmented ramp-voltage output, which is necessary in situations where the scattered light is very weak.

The purpose of the present work was to modify and augment this MCS package to perform all the functions of data acquisition and feedback control required for scanning Fabry-Perot interferometry using either single-pass or multi-pass modes. It was consequently necessary to modify and augment the hardware and software to accommodate the following functions:

- Drift stabilization
- Finesse optimization
- Segmented ramp scanning

The drift stabilization and finesse optimization functions mainly required software development, while segmented scanning required hardware development in the form of a digital electronic circuit.

The organization of this thesis is as follows: Chapter 2 discusses the Fabry-Perot interferometer and the technique of system stabilization. Chapter 3 describes the design and implementation of computer control functions. Chapter 4 presents the experimental results obtained using this computerized data acquisition system in the Brillouin scattering experiments. Appendix A lists the MCS program. Appendix B lists the interrupt service routine.

Chapter 2

FABRY-PEROT INTERFEROMETRY

The Fabry-Perot interferometer is one of the most useful spectroscopic instruments for high resolution spectroscopy. This instrument was first introduced by Fabry and Perot in the last years of the 19th century. It is simple, elegant and powerful. Today, the Fabry-Perot interferometer is used not only in high resolution spectroscopy, but also in many other fields. Here we briefly discuss the general characteristics of the Fabry-Perot interferometer. Detailed treatments can be found in books such as those by Born and Wolf[4], Vaughan[5], and Hernandez[6].

2.1 Basic Characteristics

A plane Fabry-Perot interferometer consists of two plates of glass or fused quartz which are parallel to each other and separated by a distance d. The inner surfaces of the plates are worked extremely flat and coated with films of high reflectivity. The plates are made slightly wedged in order to avoid beams reflected from the outer surfaces.

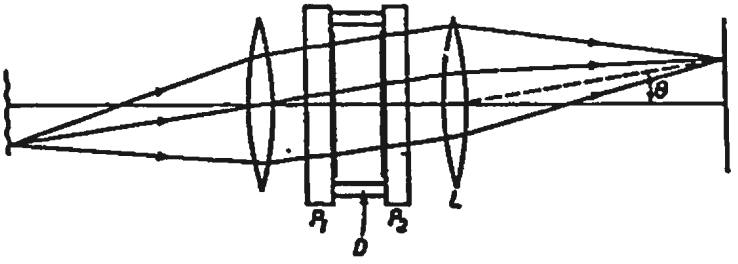


Figure 2.1: Fabry-Perot interferometer

When a beam of monochromatic light from an extended source is incident on the Fabry-Perot interferometer (see Fig. 2.1), it undergoes multiple reflections between the two plates and forms a set of narrow circular fringes in the focal plane of the lens L. For the case in which a monochromatic light beam falls on the plates and its intensity is taken to be unity, the transmission intensity $I(\delta)$ is given by the well-known Airy function,

$$I(\delta) = (\frac{T}{1-R})^2 \frac{1}{1+4R/(1-R)^2 \sin^2(\delta/2)} \tag{2.1}$$

where R and T are the reflectance and transmittance respectively. The phase difference δ between two successive reflections is given by $\delta = (2\pi/\lambda)2nd\cos\theta$, here n is the refractive index of the medium between the plates, d the distance between the plates, θ the angle of the incident rays with respect to the optical axis, and λ the wavelength of the incident light. Maximum transmission occurs if the phase difference δ equals $2m\pi$, i.e. the optical path is equal to an integer multiple of λ,

$$2nd\cos\theta = m\lambda \tag{2.2}$$

where m is an integer and is called the *order* of interference. For normal incidence of

the light, and $n = 1$ for air, Eq. (2.2) becomes

$$2d = m\lambda. \tag{2.3}$$

The spectral separation between the mth and the $(m+1)$th order can be considered in terms of a change in wavelength of the radiation, or a change in the optical spacing. If this optical spacing is fixed, the necessary change in wavelength to move the order by one unit is called the *free spectral range* (FSR). Consider two wavelengths λ and λ'. If λ' exceeds λ by one free spectral range $\Delta(\lambda)$, the mth-order fringe for λ' will coincide with the $(m+1)$th-order fringe for λ,

$$(m+1)\lambda = m\lambda' = m(\lambda + \Delta\lambda). \tag{2.4}$$

From Eq. (2.3) and Eq. (2.4), we have

$$(\Delta\lambda)_{FSR} = \frac{\lambda^2}{2d} \tag{2.5}$$

which is the usual free spectral range expressed in wavelength units. The corresponding free spectral range expressed in terms of the frequency is

$$(\Delta\nu)_{FSR} = \frac{c}{2d} \tag{2.6}$$

where c is the velocity of light. The free spectral range is the maximum spectral separation of adjacent transmission maxima which can be observed without overlap.

One of the important characteristics of the Fabry-Perot interferometer is the *finesse*, F, which is defined as the ratio of the fringe separation and the fringe full

width at half maximum (FWHM). The phase interval ε for the fringe FWHM can be found from Eq. (2.1) when solved for $I(\varepsilon/2) = I_{max}/2$, ie.,

$$4R/(1-R)^2 \sin^2(\varepsilon/4) = 1. \tag{2.7}$$

Since ε is usually very small, $\sin^2(\varepsilon/4) \approx (\varepsilon/4)^2$, then we can find the fringe FWHM,

$$\varepsilon = \frac{2(1-R)}{\sqrt{R}}. \tag{2.8}$$

The phase interval of fringe separation is 2π, we then obtain the expression

$$F = \frac{2\pi}{\varepsilon} = \frac{\pi\sqrt{R}}{1-R}. \tag{2.9}$$

An important consideration in any spectrometer is the ability of the instrument to resolve spectral lines which are close to each other. This is determined by the *resolving power* of the instrument. The resolving power is defined by the ratio $\lambda/\Delta\lambda$, where the spectrometer can just resolve the lines of wavelength λ and $\lambda + \Delta\lambda$. There exist different criteria to specify whether two spectral lines can be said to be just resolved. The most commonly used criterion is that two identical lines are considered to be just resolved when their maxima are separated by their full-width at half-maximum (FWHM). In order to obtain the expression for the resolving power, we need to derive the wavelength interval $\Delta\lambda$ corresponding to the phase interval $\Delta\delta$ which is equal to the fringe FWHM ε. Since $\delta = (2\pi/\lambda)2d$, we have

$$\Delta\delta = 2\pi\frac{\Delta\lambda}{\lambda^2}2d. \tag{2.10}$$

Let $\Delta\delta = \varepsilon$, we find $\lambda/\Delta\lambda$ in terms of FWHM ε,

$$\Delta\delta = \varepsilon = 2\pi\frac{\Delta\lambda}{\lambda^2}2d \tag{2.11}$$

$$\frac{\lambda}{\Delta\lambda} = \frac{2\pi}{\varepsilon}\frac{2d}{\lambda}. \tag{2.12}$$

With Eq. (2.3) and Eq. (2.9), we obtain

$$\frac{\lambda}{\Delta\lambda} = \frac{m\pi\sqrt{R}}{1-R} = mF. \tag{2.13}$$

As can be seen from the above expression, the resolving power of the Fabry-Perot interferometer is proportional to the interference order m and the finesse F.

Since $m = 2nd\cos\theta/\lambda$, it seems that a higher resolving power can be obtained by simply increasing the spacing between the plates. But increasing the spacing will decrease the free spectral range of the Fabry-Perot because the free spectral range is inversely proportional to the spacing. Hence we need to be able to attain a large enough free spectral range with sufficient resolution, and this usually involves a compromise which avoids the confusion of overlapping orders.

Another important factor of the Fabry-Perot is the *contrast*, defined as the ratio of the maximum transmission intensity to the minimum transmission intensity. The contrast is the ability of the Fabry-Perot to detect weak lines in the presence of strong lines in the same spectrum. From Eq. (2.1), it is easy to find the maximum and minimum of the transmission intensity. For $\delta = 2m\pi$,

$$I_{max} = (\frac{T}{1-R})^2, \tag{2.14}$$

for $\delta = (2m+1)\pi$,

$$I_{min} = (\frac{T}{1+R})^2. \tag{2.15}$$

The contrast is

$$C = \frac{I_{max}}{I_{min}} = (\frac{1+R}{1-R})^2 = 1 + \frac{4F^2}{\pi^2}. \tag{2.16}$$

Contrast has a very important significance in light scattering spectroscopy. In the field of Brillouin scattering, the Fabry-Perot interferometer is the most satisfactory instrument available for the detection of components that are frequency-shifted by only a few GHz. The resolving power of the Fabry-Perot is sufficient for measuring the Brillouin components. But the contrast of a conventional Fabry-Perot is usually less than 10^4 which is not enough in some cases, such as in the measurements of non-transparent crystals in backscattering experiments. This kind of difficulty can be overcome by using a multi-pass Fabry-Perot interferometer.

2.2 Multi-Pass Fabry-Perot

Houston[7] first introduced a compound interferometer having two Fabry-Perot etalons of different thickness arranged in tandem. It was shown that if the ratio of the thicknesses of the two etalons is an integer, then the free spectral range of this instrument will be determined by that of the thin etalon, while the resolving power will be somewhat larger than that of the thick one. According to the calculations of Meissner[8], the intensity distribution of the compound interferometer is just the product of the intensity expression for the each interferometer. If the two etalons are identical, then the contrast C is equal to ${C_1}^2$, where C_1 is the contrast of the single-pass Fabry-Perot.

The difficulty of using two Fabry-Perots in tandem is that of keeping the two

interferometers in precise alignment. A simple and attractive method was suggested by Dufour and successfully demonstrated by Hariharan and Sen[9]. Instead of using two Fabry-Perots in tandem, they employed a *double-pass* interferometer which only used one Fabry-Perot, but with the light made to pass through it twice.

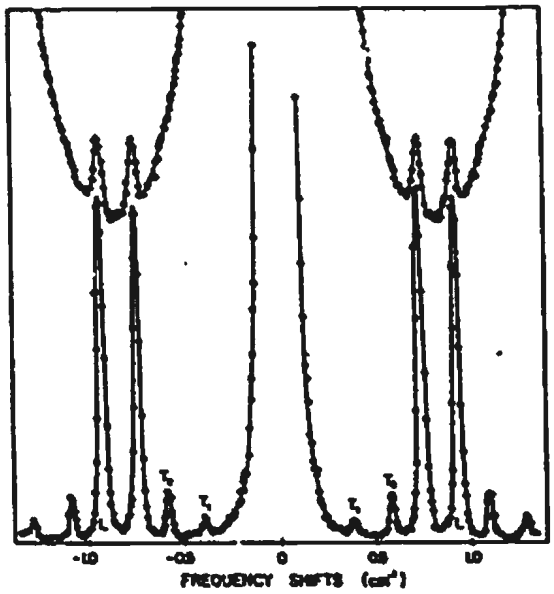


Figure 2.2: Brillouin back scattering spectrum of SbSI using a double-pass interferometer. The upper curve shows the same spectrum obtained using a single pass through the same interferometer.

In a series of excellent studies, Sandercock successfully applied the multi-pass Fabry-Perot to Brillouin scattering experiments. For example, a double-pass Fabry-Perot interferometer was used to study the Brillouin scattering spectrum (Fig. 2.2) in the ferroelectric SbSI[10]. The scattered light was made to reflect back through the Fabry-Perot again by means of a corner cube. The upper curve shows the backscattering spectrum of SbSI using a conventional single-pass interferometer where the

longitudinal phonon peak is just resolved in the wings of the Rayleigh peak. As can be seen from the lower curve in Fig. 2.2, the contrast was greatly improved for double-passing through the same Fabry-Perot. The transverse modes are now resolved in addition to the longitudinal modes. The wings of the Rayleigh peak are drastically reduced.

Later Sandercock further improved the system described above by constructing a five-pass Fabry-Perot interferometer[11, 12], using two triple-reflection retroreflectors. With a high contrast in excess of 10^9, Sandercock was able to measure the Brillouin scattering spectra of non-transparent semiconductors[13] and thin films[14].

2.3 Fabry-Perot Scanning

In order to obtain a light scattering spectrum by a scanning process, we must continuously move the fringes relative to an aperture. The possible means to achieve this relative movement, or scanning, can be found in Eq. (2.2), i.e.,

$$2nd\cos\theta = m\lambda.$$

This expression indicates that in order to change the order m, one or more of the left side variables n, d, and/or θ must be changed.

A change in the refractive index n may be considered as a change in the optical thickness nd, or in the effective wavelength of the radiation $n\lambda$. This method of scanning is called *refractive index scanning*. Changing the physical distance between the plates is called *mechanical scanning*, while changing the angle θ is called *spatial*

scanning.

The most recent developments are associated with the method of mechanical scanning. Mechanical scanning of a Fabry-Perot is very attractive, since it is necessary only to change the spacing between the plates by $\lambda/2$ in order to scan one order of interference. The most successful method to achieve mechanical scanning has been associated with the use of piezoelectric materials. Piezoelectric elements made of barium titanate have been widely used. The use of piezoelectric materials satisfies most of the requirements for mechanical scanning, and has very few drawbacks. Among the advantages of using piezoelectric scanning is the mechanical stability and the rather modest requirements for the operation of these devices. However, the most important advantage of piezoelectric scanning is the realization of dynamic alignment of the Fabry-Perot, rather than relying on the mechanical stability itself, which will be discussed later. The presently available piezoelectric materials for mechanical scanning are found to have measurable non-linearity and their behavior has been studied in some detail[15]. It is necessary to select and match the behavior of the individual piezoelectric elements, and to modify the scanning ramp function in order to achieve linear scanning. The remainder of the discussion will be limited the piezoelectric scanning technique.

2.4 System Stabilization

Proper operation of a Fabry-Perot presumes the plates of the Fabry-Perot to be parallel, or aligned, and maintained stable for a reasonable period of time while the

experiment is in progress. Misalignment can cause very serious losses of finesse, and therefore of resolving power and contrast. This problem of maintaining parallelism is not easy to solve. In general there are two ways to solve the problem: one is by using very refined mechanical construction and thermostating of the Fabry-Perot interferometer, and the other is by using automatic devices to maintain dynamic parallelism. The former is a passive method and the latter is active. In practice it is more effective to use an active method to maintain parallelism.

Ramsay[16] first developed a rapid scanning Fabry-Perot interferometer with automatic parallelism control. Sandercock[10] also used a stabilization technique in his first double-pass Fabry-Perot interferometer. Various methods for automatic parallelism control have been reported in the literature[17]–[21]. One of the earliest hard wired data acquisition and control systems for a Fabry-Perot interferometer was developed by Jones in 1969[5]. It has been used ever since in laser light scattering spectroscopy with little change in the basic equipment and logic.

May *et al.*[1] described a data acquisition and stabilization system for Fabry-Perot interferometry which automatically compensates for all instrumental drifts, maintains Fabry-Perot alignment, and allows extended data acquisition time in selected regions of a spectrum. The spectrometer consists of a piezoelectrically scanned Fabry-Perot interferometer, a photomultiplier detector, photon counting electronics, and a data acquisition system. A digital clock generates a series of pulses which are accumulated in a scaler. The scaler output feeds a digital-to-analog converter to produce

a voltage ramp. This voltage ramp is applied to the three piezoelectric elements on which one of the plates is mounted. The zero-level of the scaler is controlled by the drift compensation logic circuitry. The clock is also used to sequentially address the channels of a 1024 (or 512) channel memory. Therefore the channel number is directly proportional to the frequency shift $\Delta\nu$.

The principle of frequency-drift compensation is similar to that described by McLaren and Stegeman[17]. In order to eliminate the effect of frequency drift, a prominent spectral feature is locked to a selected memory channel, say channel N. At first, the spectral peak to be locked is brought to channel N by manually adjusting the zero-level of the ramp scaler. Two digital registers are used to accumulate the photon counts falling into the two channel windows beside channel N. On each sweep, register A keeps the counts in the lower window from channel $N - \Delta N$ while register B keeps the counts in the upper window from $N + \Delta N$. Ignoring the statistical fluctuations, if the counts in register A are more than in register B, it means that the spectral peak has drifted away from the channel N toward lower channel numbers, or vice versa. Upon comparing the contents of the two registers, a correction is made to the zero-level of the scaler controlling the ramp so as to bring the peak back toward channel N.

The procedure for finesse optimization is a little more complicated. The principle of finesse optimization is based on the fact that the maximum transmission of a spectral line by the Fabry-Perot is determined by the finesse of the Fabry-Perot.

Assuming a constant incident light intensity, any decrease in this peak transmission indicates that Fabry-Perot misalignment has occurred. Similarly, test voltages can be intentionally applied to the piezoelectric elements, resulting in a slight change in the finesse. Once the outcome of this test is known, appropriate correction voltages may be applied to achieve improved finesse. This approach can be implemented as follows.

A window centered at channel N is selected, and the counts falling into the window are stored in a register. This window usually encompasses less than the full-width at half-maximum of the reference spectral line. Since the maximum transmission of the Fabry-Perot is determined by the instrumental finesse, the change of counts in the register shows whether the system is in the optimum alignment. The procedure of finesse optimization consists of four sweeps. On the first sweep the contents of the register are stored as a measure of the existing finesse. On the next sweep the plates are tilted a small amount in such a way that the distance between the centers of the plates remains unchanged. By comparison of the contents accumulated during the two sweeps, appropriate voltages are then applied to the piezoelectric elements to make the correction. During the next two sweeps the alignment is tested and correction is made about the orthogonal axis. This procedure maintains an optimum finesse which is better than that achieved manually.

Another important feature of the data acquisition system[1] is segmented scanning. Because the same clock is applied to generate the voltage ramp and to address

the memory, the relation between piezoelectric extension and memory address is independent of the ramp clock rate. Therefore the scanning voltage can be chosen as a segmented, or multi-slope ramp, which permits rapid scanning through uninteresting part of the spectrum and devotes most of the real experimental time to the more interesting spectral regions. This technique greatly enhances the signals and therefore the signal to noise ratio (SNR) in the selected regions.

The commercial version of this data acquisition system is known as DAS–1. It was originally produced by Burleigh Instruments, Inc. in 1976 and has been the accepted standard among commercially available instruments. However it has not been available for several years. It was necessary to develop an IBM PC-based data acquisition and control system for Fabry-Perot interferometry.

Chapter 3

COMPUTER CONTROL FUNCTIONS

3.1 Existing Hardware and Software

The multichannel analyzer (MCA) is widely used in spectroscopic data acquisition. The basic requirement of an MCA is that it be able to gather and store spectral data acquired from a detector, and be able to present that data for display and further analysis. The most commonly used MCA acquisition modes are: pulse height analysis (PHA), and multichannel scaling (MCS). Pulse height analysis is the traditional operating mode of the MCA, and is used for accumulating a spectrum of the frequency distribution of the height from a sequence of input pulses. The PHA mode is usually used for nuclear and x-ray energy spectroscopy. The desired spectrum is accumulated by measuring the amplitude of each input event, converting it to a channel number that is proportional to the pulse height, and storing the information in a memory composed of individual channels. The count value of each channel is equal to the total number of pulses processed whose amplitudes correspond to the channel

number.

In the multichannel-scaling mode, the individual channels of the memory are analogous to a sequence of counters, with each channel counting the data for a preset dwell time. At the completion of each dwell time the counting operation is automatically passed to the next channel in the memory, resulting in a time histogram of the count-rate data where each channel represents a sequential time interval. The dwell time for each channel is set by an internal clock or by an external channel advance signal. The MCS mode is useful in those applications where analysis of count-rate data related to elapsed time is of interest. Measurement of decay rate of short-lived isotopes, X-ray diffraction, and Mössbauer analysis are typical applications of the MCS mode. The present thesis focuses on its application to Fabry-Perot interferometry.

Basic MCS hardware and software packages that perform multichannel scaling are commercially available. The package used in the present case was that developed by EG&G ORTEC[3]. This MCS package is made up of an IBM PC plug-in card and associated software which was written in the Pascal language. It includes a graphics routine which provides a dynamic display of the spectral data being collected.

The spectral data collected by the MCS card can at any time be transferred to a special data area in the IBM PC memory. This data buffer is used to manipulate collected data while the MCS card is acquiring the next spectrum. The spectral data, as well as results of calculation, can be displayed or printed, and can be stored as disk files for later retrieval.

A number of setup functions allow the user to define the operating characteristics of the system. There are two basic modes for data acquisition, namely, the *auto* (erase) mode and the *collecting* mode. When working in auto mode, old data from the previous sweep are automatically erased by newly acquired data and the behavior of the graphics display is similar to that of an oscilloscope. This mode is useful, indeed essential, for the performance of initial adjustments to the optical system. The collecting mode, as its name implies, is used for actual data collection; i.e., new data are summed with old data until the process is terminated by the operator, or until a preset number of sweeps has been completed. In either case the total number of channels can be preset in the range from 4 to 4096, and the total maximum number of counts that can be accumulated in any given channel is 2^{24} at a maximum count rate of 100 MHz.

In addition to the foregoing selection, a choice can also be made to operate the MCS card under either *internal* or *external* control. If the internal (free-running) mode is selected, successive sweeps are automatically initiated after a 200 microsecond delay. Futhermore, the preset (2 microsecond to 30 min) dwell time must be the same for each channel and cannot be changed without terminating the data collection process. This method of operation did not provide sufficient flexibility for the present purpose, and has only limited application. When operating under external control, an independent external device is needed to provide (1) a start signal for each and every sweep and (2) separate channel-advance signals for stepping the data collection

process through successive channels. This method provided the complete control over the operation of the MCS board which was necessary in the present application.

An essential requirement of the system was, of course, that the scanning of the MCS board be locked in synchronism with the scanning of the Fabry-Perot interferometer. In this connection, the EG&G package conveniently provided an optional (piggy-back) board which generated a 0 to 10 V (ramp) output in direct proportion to the MCS channel number. It was consequently only necessary to amplify this output to the levels (0 to 1000 V) required to drive the piezoelectric elements in the Fabry-Perot. Details of the various hardware modifications, including the external timing circuitry, will be discussed later.

The EG&G software was designed to control up to eight MCS cards. Most commands involve one or two keystrokes, with the key definitions being displayed on the monitor screen. When the program is started, the monitor display appears as in Fig. 3.1. The boxes on the left side of the screen labelled F1 through F10 represents the 10 function keys. The text and symbols inside the boxes indicate their functions and sometimes indicate the status of the system.

The second line up from the bottom of the screen display is the menu line. There are five menus: MAIN menu, PRESETS menu, CALC menu, I/O menu, and UTILITIES menu. The commands displayed in each menu are invoked by holding down the ALT key and then pressing the appropriate upper-row number key. For example, while in the MAIN, ALT-1 starts data acquisition, ALT-2 stops it, ALT-3 clears the data

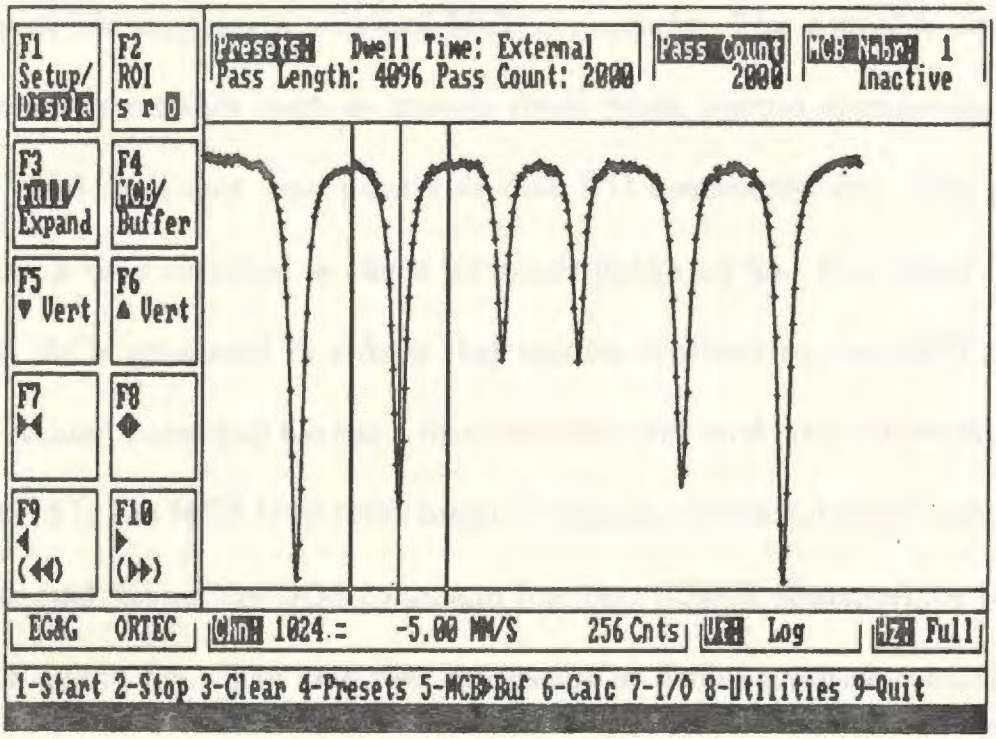


Figure 3.1: MCS program screen display

spectrum, etc. Pressing ALT-7 in the MAIN menu switches to the I/O menu. Then ALT-1 will save a spectrum to disk, or ALT-2 will retrieve a spectrum from the disk. The PRESETS menu is used to preset the number of channels and dwell time per channel. The CALC menu permits calculations to be performed on spectral data. The UTILITIES menu provides some generally useful commands. For example, DOS COM or EXE programs can be run within the MCS system, or an exit to DOS can be accomplished while MCS system continues data acquisition. Finally, ALT-9 in MAIN menu results in termination of the MCS program.

One of the features of this MCS software is the use of MCS command files. A text editor can be used to create a text command file with a filename extension **txt**. This

text command file contains a series of MCS commands. The available commands include presets commands such as presets dwell time, control commands such as start, stop, wait, and quit, and printer or disk I/O commands, etc. The program *parsemcs.exe* is then executed to check for errors in the txt file. If no error is found, a command file is generated in a form that can be executed by the MCS software. The newly created command file has a filename extension cmd, and can be invoked by pressing ALT-3 in the MCS UTILITIES menu. It can also be executed by including the name of the cmd file on the DOS command line (eg. C:MCS filename), or including it in a DOS batch file. The cmd files are useful in defining initial conditions and performing repetitive tasks.

The MCS package provided by EG&G is simple and easy to use. However, as previously noted, it is not adequate for direct application to Fabry-Perot interferometry. The principal shortcoming is that there is no provision for the active feedback control required to maintain the interferometer alignment. In this connection the required new features include both *drift stabilization* and *finesse optimization*, and involve software development as well as hardware modifications. In addition, it was necessary to add the segmented ramp feature which was needed in the often-encountered situation where the level of scattered light is very low. This modification primarily involved hardware development in the form of digital timing circuitry. The implementation of these additional control features will now be described in some detail.

3.2 The Segmented Ramp

The partitioning of the scanning ramp for the Fabry-Perot into alternately slow and fast segments is frequently desirable as a means of economizing on accumulation time. The minimal requirement, as implemented here, is for two different scanning rates — one fast (or normal) rate for covering uninteresting regions of the spectrum, and one slow rate for concentrating the data collection time in regions of particular interest. Each of these two scanning rates must be independently selectable as part of the initial setup procedure.

As already noted, the most convenient means of generating the scanning ramp was via an optional output from the MCS card which was proportional in magnitude to the channel number being addressed. Segmentation of this ramp was consequently achieved by changing the per-channel dwell time with the MCS card operating under the control of an external clock which was capable of generating two different pulse trains with independently selectable frequencies. For example, if the (normal) dwell time for a 1000 channel sweep is chosen to be 1 ms, but is to be increased to 50 ms over the range from channels 490 to 510, then the external clock must be programmed to deliver 490 pulses at 1 kHz, followed by 21 pulses at 20 Hz, and the remaining 489 pulses at 1 kHz again. This, in turn, must be followed by a programmed 100 ms delay to allow the piezoelectric elements to recover before starting the next sweep.

The pulse trains were conveniently provided by readily available IBM Data Acquisition and Control Adapters (#6451502) which were designed as plug-in expansion

boards for the IBM PC. Two such boards (CARD-0 & CARD-1) were installed to provide most of the additional control requirements for the system. Each of these boards provides the following functions: (i) four analog input channels multiplexed into a 12-bit ADC, (ii) two analog output channels each with its own 12-bit DAC, (iii) a 16-bit digital input port, (iv) a 16-bit digital output port, (v) two 16-bit timers cascaded as a 32-bit timer, (vi) a 16-bit externally-clocked timer/counter, and (vii) an expansion bus.

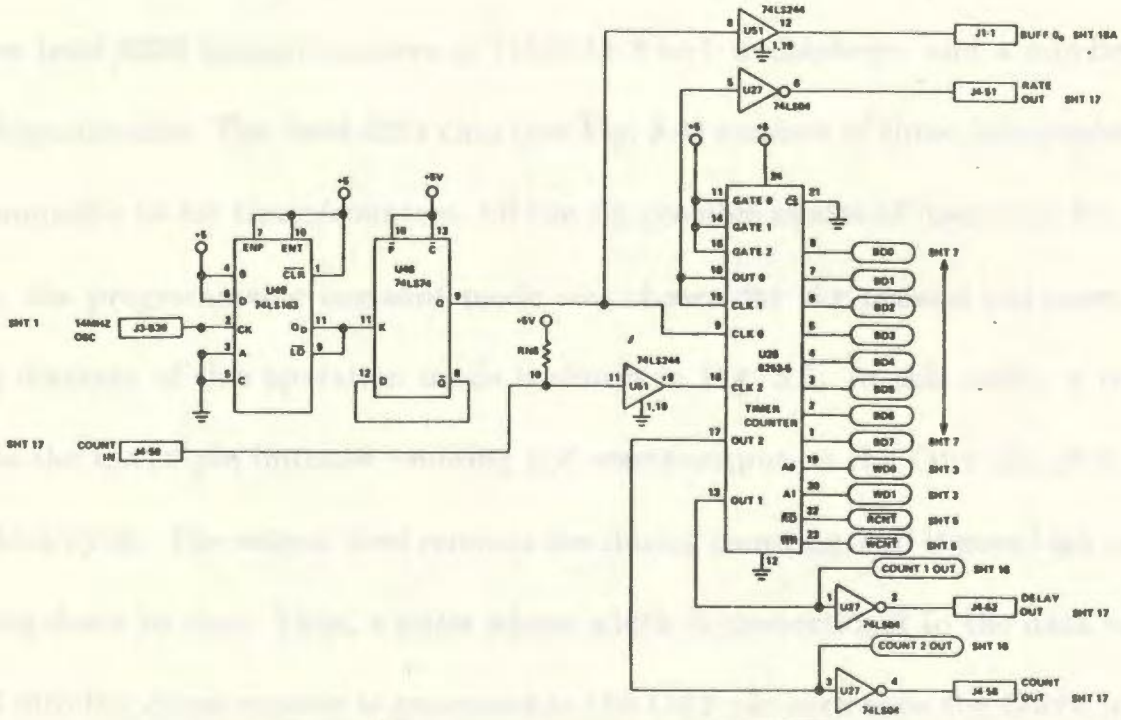


Figure 3.2: Logic diagram of Timer/Counter

The two pulse trains required for segmented ramp operation were provided by the pair of programmable timers, OUT0 and OUT1, on CARD-0 (see Fig. 3.2). OUT0 was chosen to determine the normal dwell time in the range from 10 to 65535 microseconds. The pulses from OUT0 were used as input for OUT1 so that the period of the output

pulses from OUT1 was a multiple of that from OUT0. These (OUT1) pulses were used to define the slow, or extended, dwell time with available multipliers in the range from 2 to 999. The output pulses from both OUT0 and OUT1 were delivered as input to a separate timing circuit external to the computer which could be programmed to determine the distribution of fast and slow segments in each sweep, and which delivered the appropriate sequence of channel-advance pulses to the MCS board.

The logic diagram for the external clock circuit is shown in Fig. 3.3. It consists of three Intel 8253 timers/counters, a 74LS151 8-to-1 multiplexer, and a number of gated logic circuits. The Intel 8253 chip (see Fig. 3.4) consists of three independently programmable 16-bit timer/counters. Of the six possible modes of operation for this device, the *programmable one-shot* mode was chosen for the present purposes. A timing diagram of this operation mode is shown in Fig. 3.5. In this mode, a rising edge on the GATE pin initiates counting and resets output on the OUT pin after the next clock cycle. The output level remains low during counting, and it goes high upon counting down to zero. Thus, a pulse whose width is proportional to the data value loaded into the count register is generated at the OUT pin each time the GATE input is triggered. The loading of the registers, i.e. the programming of the counters, was conveniently accomplished from the computer keyboard via the 16-bit digital output port provided as a standard function on CARD-0.

As can be seen in Fig. 3.3, each GATE input of a timer is connected to the OUT pin of another timer. The rising edge of the output pulse from the first timer triggers

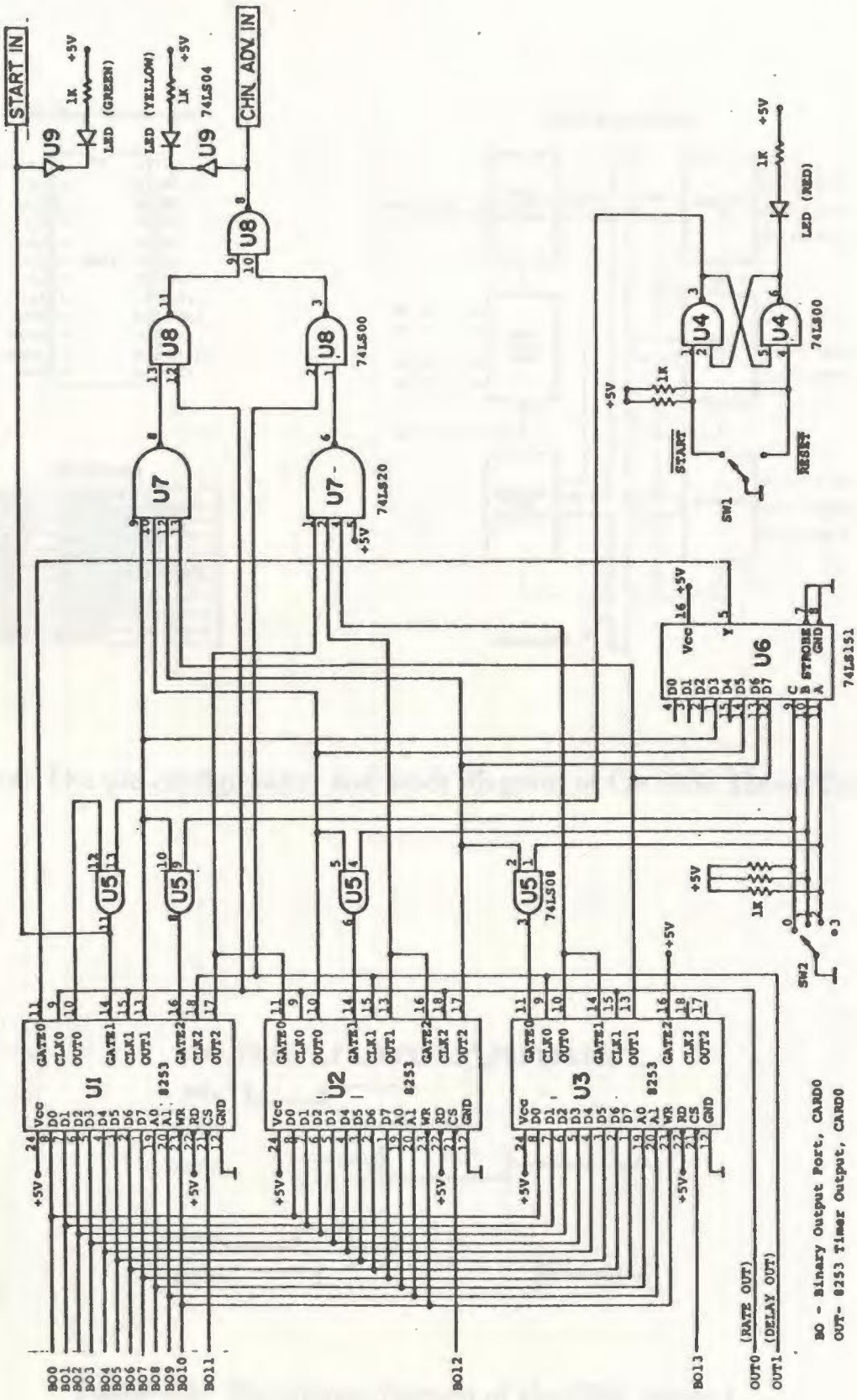


Figure 3.3: Logic diagram of external clock for segmented ramp

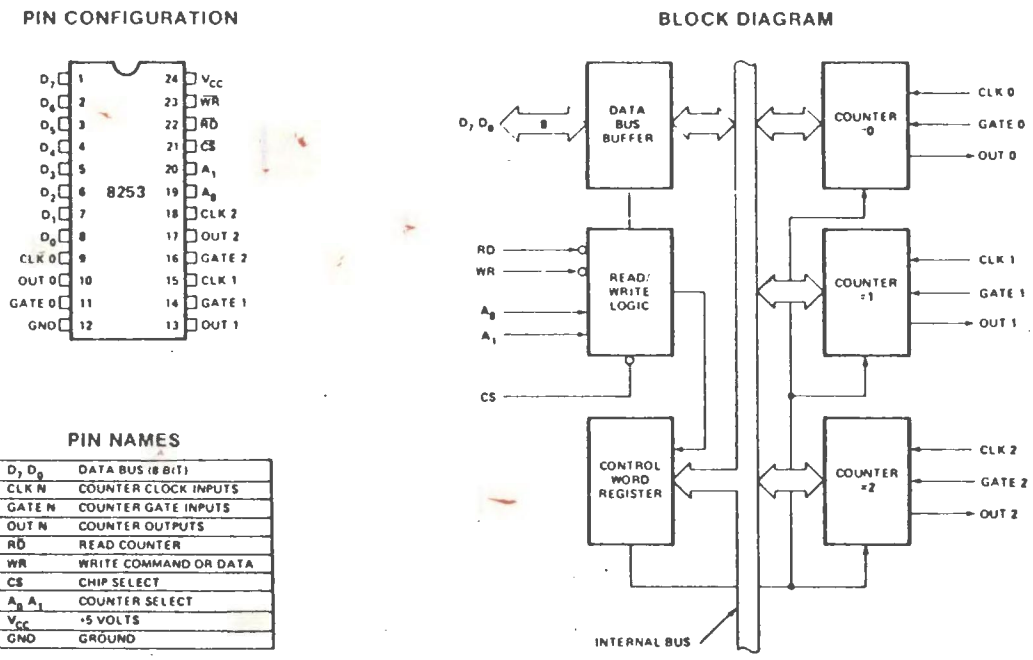


Figure 3.4: The pin configuration and block diagram of the 8253 Timer/Counter

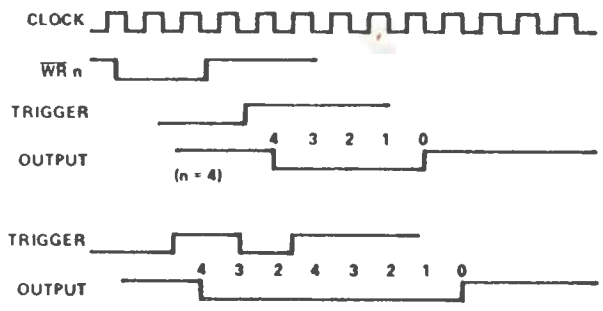


Figure 3.5: The timing diagram of the 8253, mode 1

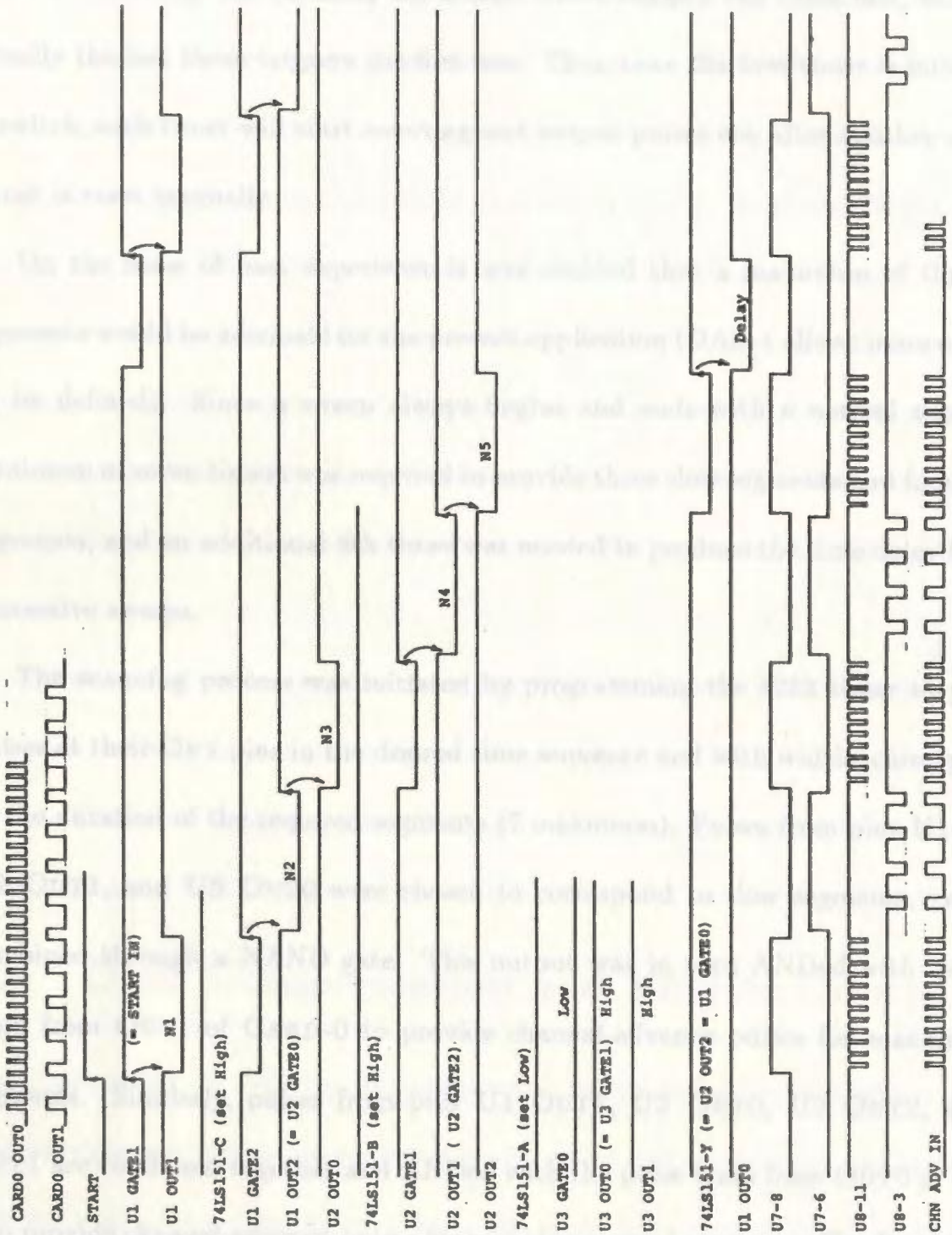


Figure 3.6: The timing diagram of the external clock

the second timer, and in turn, the second timer triggers the third one, and so on. Finally the last timer triggers the first one. Thus, once the first timer is initiated by a switch, each timer will start counting and output pulses one after another until the timer is reset manually.

On the basis of past experience it was decided that a maximum of three slow segments would be adequate for the present application (DAS–1 allows more segments to be defined). Since a sweep always begins and ends with a normal segment, a minimum of seven timers was required to provide three slow segments and four normal segments, and an additional 8th timer was needed to produce the time delay between successive sweeps.

The scanning process was initiated by programming the 8253 timer to produce pulses at their OUT pins in the desired time sequence and with widths corresponding to the duration of the required segments (7 maximum). Pulses from pins U1 OUT2, U2 OUT1, and U3 OUT0 were chosen to correspond to slow segments, and were combined through a NAND gate. This output was in turn ANDed with the pulse train from OUT1 of CARD-0 to provide channel-advance pulses for scanning slow segments. Similarly, pulses from pins U1 OUT1, U2 OUT0, U2 OUT2, and U3 OUT1 are combined together and ANDed with the pulse train from OUT0 of CARD-0 to provide channel-advance pulses for scanning normal segments. Finally, these two series of pulses with different frequencies are combined to form the required continuous sequence of channel-advance pulses for the entire scan.

Because the ramp segmentation requirements are variable, the circuitry must be flexible enough to accommodate a range of situations. The 74LS151 chip was used for this purpose. Four of its input pins are connected to U1 OUT1, U2 OUT0, U2 OUT2, and U3 OUT1, and its output is connected to U1 GATE0, the GATE input pin of the delay timer. One of these 8253 OUT pins is selected according to the position of the mechanical switch *sw2*. When *sw2* is set to position '0', a linear ramp is generated without segments. When *sw2* is set to position '1', '2', or '3', the results is a segmented ramp with one, two, or three slow segments. Fig. 3.6 shows a typical timing diagram for the external clock circuit. The are two slow segments in this example.

Switch *sw1* is used to start or reset the clock. When *sw1* is switched from RESET to START, a rising edge is sent to the START_IN pin of the MCS card to start the first sweep. The start signal for the subsequent sweeps is produced by the output from the delay timer OUT0 on U1.

3.3 Drift Compensation and Finesse Control

The requirement for drift compensation arises from instabilities throughout the system which result in time-dependent changes, either real or apparent, in the observed frequency shift for a given spectral line. The basic procedure, as previously outlined, involves the application of bias corrections to the three piezoelectric translators upon which the moving reflector of the Fabry-Perot is mounted. Drift compensation is effected by the application of a common correction to all three translators at once,

while finesse control requires a more complicated procedure which involves a cycle of test and correction voltages applied to the individual translators, and spanning several successive sweeps. The required total of four bias supplies was provided by the two pairs of DAC's which were present on the IBM Adapter cards (CARD-0 & CARD-1). These outputs were software controllable, and the necessary test/correction operations were performed during the deadtime between successive sweeps.

Branching to the appropriate software was achieved via the interrupt signal which is generated by the MCS board on completion of each sweep. This signal forces the 8086/8088 CPU to suspend execution of the main (MCS) program and pass control to an interrupt service routine (ISR). After the interrupt service routine is completed, control is returned to the point where the CPU previously left the main program. The ISR, which is listed in the Appendix, comprises a number of subroutines for performing routine tasks as part of the original MCS software. A new subroutine, named *stabilization*, was added to ISR in order to achieve active feedback control of the interferometer alignment.

The stabilization routine provided a number of different options depending on the status of the system. The choice of options was determined by checking three different flags. The first of these is the *drift_control* flag which is set from the keyboard when the operator requires drift control to be active. If only this flag is set, then drift corrections only will be performed. A separate *finesse_control* flag, which can likewise be set from the keyboard, is subsequently checked to determine whether the finesse

control feature should also be active in addition to drift control. It is noted in passing that finesse control by itself is not a realistic option, and is prohibited. A check is also made on the *sum_flag* which is automatically set when the *collect* mode is active (i.e., it is reset when the *auto* mode is active). The testing of this flag is necessary because the requirements for drift and finesse control are different for the two data acquisition modes. If only drift stabilization is required in the *auto* mode, then a correction to the drift control voltage is made after each sweep since only the current-sweep counts are retained in this mode, and a simple comparison of the (current) counts in the two drift windows is all that is required. If both drift and finesse control are required in the *auto* mode, then a cycle of four sweeps is necessary. At the end of the first sweep the counts in the two drift windows are compared, and a correction to the drift control voltage is made (as above). In addition, the total counts in the finesse window are stored for reference in the next sweep, and appropriate test voltages are applied to the three piezoelectric elements to achieve a tilt about the x direction. After the second sweep, the total count in the finesse window is compared with that for the first sweep and finesse control voltages are adjusted according to the result of this comparison. No drift correction is attempted at this point because of possible interference between the two types of correction. The next two steps in the cycle perform the finesse adjustment with respect to the y direction.

For the *collect* mode, the stabilization procedures are different because in this mode the counts accumulated during a given sweep are always summed with the

counts accumulated during previous sweeps. It is consequently necessary to subtract the total window counts for *two successive* sweeps in order to obtain the net counts collected during the most recent sweep. Thus, the number of sweeps needed to complete a cycle of drift and finesse control is doubled in the *collect* mode.

One of the problems addressed in the stabilization routine was the effect of statistical noise upon the drift and finesse controls. With respect to drift control, the principal effect is that the counts in the two windows are rarely identical even if the system is perfectly stable and no drifting has in fact occurred. This causes the drift control voltage to change unnecessarily after almost every sweep with the result that spectral lines can be significantly broadened. In order to avoid this, the routine was modified to prohibit corrections when the difference $N_1 - N_2$ between the two window counts, N_1 and N_2, was less than their statistical uncertainty. In the event of a no-correction condition, the value of $N_1 - N_2$ was allowed to accumulate over successive cycles of drift control until it eventually exceeded the statistical uncertainty, at which point a correction was made.

A similar procedure was employed with respect to finesse control. As an example, suppose that the finesse-window count increased after a given tilt test, thereby indicating improved alignment. As above, a correction was made only if the increment in the window count exceeded the statistical error, otherwise the finesse control voltages were reset to their value before the tilt test. In this case the corresponding tilt test in the next control cycle was chosen to be identical to the previous one as being the

best guess for improved alignment. On the other hand, if the finesse-window count was found to decrease after a given tilt test, and the no-correction criterion was satisfied, then the corresponding tilt test during the next control cycle was made in the opposite direction.

Another problem which required attention was that of DAC overflow and underflow. The 12-bit DAC's in question could only accommodate input (control) data in the range from 0 to 4095 corresponding to analog outputs from 0 to 10 V. A consistent decrease or increase in the calculated input data for a DAC could eventually result in the characteristic wraparound condition where the output voltage suddenly switches from one extreme to the other. Given that the maximum single test or correction step was constrained to a DAC input value of less than 50, this condition was avoided by restricting the input data to the range 100 to 4000 as controlled by software. If either limit of this range was reached for any one of the four DAC's, then data acquisition was terminated with an appropriate warning message being displayed on the monitor.

3.4 MCS Software Development

In addition to the modifications of the interrupt service routine as described in the previous section, it was necessary to make extensive changes to the MCS software. These changes were primarily designed to provide maximum (keyboard) control over the functioning of the system, with particular emphasis on the stabilization and segmented ramp features. This included extensive modifications of the monitor display routine.

The MCS software is coded in Microsoft Pascal Language as we_ as 8086/8088 Assembly Language. It consists of:

- a Pascal main program MCS.PAS,
- an additional Pascal module, CALCULAT.IMP, which provides most of the mathematical functions for manipulation of data,
- an Assembly Language module, CGA_901.ASM, which controls the graphics display,
- an Assembly Language module, SUPPORT.ASM, which provides support for interrupt service.

Each of these was compiled separately before linking into an executable program, MCS.EXE.

The body of the main program is relatively simple and straightforward. The program first performs a number of system initialization tasks, and then enters a repeat loop. It repeatedly searches the keyboard to see if any key has been pressed by the user. If no key has been pressed, the program simply updates the screen display and continues keyboard searching. If a valid command key has been pressed, the program responds by taking appropriate actions according to the current status of the system. The program repeats this loop until it receives a QUIT command.

In the present connection the following statements are of primary interest.

...

```
else if (get_cmd(cmd)) then begin
  do_cmd(cmd);
end
...
```

Here *get_cmd* is a function which reads the keyboard, checks for errors, and assigns an appropriate command to the variable *cmd* if no error is found. *do_cmd* is a procedure which executes the command.

The function *get_cmd* invokes another function *get_key_cmd* to identify which key has been pressed and assigns an appropriate command to *cmd_name* if the key corresponds to a valid MCS command. The function *get_key_cmd* in turn calls an 8086/8088 assembly language routine *get_key* to determine the current status of the keyboard. The assembly language routine *get_key* is a function which checks for a key pressed and returns *True* if a key is available, otherwise it returns *False*. It also returns the shift-key status, ASCII code and scan_code of the key defined by IBM PC BIOS if a key has been pressed.

To add new commands to the MCS system it was necessary to modify the functions *get_cmd* and *do_cmd*, and write new procedures corresponding to each new command. First of all, *cmd_name_type* was redefined to include the names of the new commands. *cmd_name_type* is an enumerated type which was defined as follows:

```
cmd_name_type = (cmd_update_display,
                 cmd_setup,
                 cmd_roi_mode,
                 cmd_display_mode,
                 ...
                 cmd_end_loop);
```

All MCS commands must be defined here. There were 56 commands in the original MCS program. The following commands were added to accommodate the segmented ramp and system stabilization features. Unique key combinations were assigned to each of these commands.

`cmd_menu_ramp` displays a menu for programming the external clock for segmented ramp operation.

`cmd_menu_window` displays a menu for setting up the *reference* channel, the *drift window* and *finesse window*.

`cmd_menu_help` displays a menu for on-line help.

`cmd_auto_collect` switches data acquisition *auto/collect* mode.

`cmd_marker_ref` sets the marker at the *reference* channel.

`cmd_set_marker_channel` sets the marker at a specified channel.

`cmd_disp_window` displays the *reference* channel and the *drift window* and *finesse window*.

`cmd_disp_segment` displays the channel numbers for the slow segments.

`cmd_disp_step` displays the test and correction steps for drift and finesse control.

`cmd_set_window` sets the *reference* channel and widths of the *drift window* and *finesse window*.

`cmd_set_step` sets the drift and finesse control test and correction steps.

`cmd_set_drift_control` switches the *drift_control* flag on/off.

`cmd_set_finesse_control` switches the *finesse_control* flag on/off.

`cmd_set_dwell_time` sets the dwell time per channel for the external clock.

`cmd_set_segment` defines up to three slow segments in a spectrum.

`cmd_delete_segment` deletes the slow segments which have been defined previously.

`cmd_set_segment_mult` sets the segmented ramp multiplier.

`cmd_start_ramp` starts the external clock.

`cmd_set_drift_volt` sets the initial drift control voltage.

`cmd_set_tilt_A_volt` sets the initial finesse control voltage *A*.

`cmd_set_tilt_B_volt` sets the initial finesse control voltage *B*.

`cmd_set_tilt_C_volt` sets the initial finesse control voltage *C*.

`cmd_vert_arrow_A` allows up/down arrow keys to increase/decrease finesse control voltage *A*.

`cmd_vert_arrow_B` allows up/down arrow keys to increase/decrease finesse control voltage *B*.

`cmd_vert_arrow_C` allows up/down arrow keys to increase/decrease finesse control voltage *C*.

`cmd_vert_arrow_D` allows up/down arrow keys to increase/decrease drift control voltage.

`cmd_drift_volt_up` increases drift control voltage.

`cmd_tilt_A_volt_up` increases finesse control voltage *A*.

`cmd_tilt_B_volt_up` increases finesse control voltage *B*.

`cmd_tilt_C_volt_up` increases finesse control voltage *C*.

`cmd_drift_volt_down` decreases drift control voltage.

`cmd_tilt_A_volt_down` decreases finesse control voltage *A*.

`cmd_tilt_B_volt_down` decreases finesse control voltage *B*.

`cmd_tilt_C_volt_down` decreases finesse control voltage *C*.

`cmd_help_info` displays the MCS system information.

`cmd_help_menu` displays information on the menu.

`cmd_help_f_key` displays information on the function keys.

`cmd_help_alt_f_key` displays information on the alt-function keys.

`cmd_help_key_pad` displays information on the keypad.

`cmd_help_cmd_key` displays information on the MCS command keys.

There are a number of procedures which actually execute these commands and they must appear before the main body of the program. To avoid editing a very large program, these procedures were edited separately and incorporated into the MCS

program during compilation via the Microsoft Pascal metacommand, $*include*. The following are the files which contain the procedures, they are listed in a order in which they appear in the main program. Most of the procedures were modified, and those with major modification are marked by †. Those files marked by ‡ are newly created to accommodate new commands.

general.pas contains a variety of functions and procedures used by various routines.

display.pas contains several procedures for displaying various information.

do_scrn.pas† draws the screen as shown in Fig. 3.1.

config.pas† contains two functions: *get_config* and *put_config*, they are used for saving and retrieving the MCS system configuration.

setup.pas† allows the user to set the system configuration.

update.pas† displays: (i) the number of sweeps completed, (ii) the counts accumulated in the current marker channel, (iii) a graphics plot of the data spectrum, and (iv) the DAC data for the drift and finesse control voltages.

This procedure is invoked most frequently by the procedure *do_cmd* since the function *get_cmd* always assigns *cmd_update_display* to the variable *cmd* if no key has been pressed or if an error has been found in getting a command.

fkeys.pas contains a number of procedures for adjusting the graphics display, such as changing the horizontal or vertical scale, and switching the display between

the MCS card and the Buffer.

marker.pas contains procedures for marker positioning functions.

roi.pas contains procedures for searching for a Region of Interest.

buffer.pas contains two procedures. The procedure *fill_buffer* transfers data spectrum from the MCS card to the Buffer, while the procedure *restore* transfers data from the Buffer to the MCS card.

presets.pas contains three procedures which allow the user to preset the *dwell_time*, *pass_length*, and *pass_count_preset*. The *dwell_time* determines the data acquisition time per channel in internal dwell_time clock mode, the *pass_length* sets the total number of channels in a spectrum, and the *pass_count_preset* sets the total number of sweeps desired.

type_dat.pas sends spectral data to a printer. In the full spectrum display mode, all data are printed. In expanded display mode, if the marker is inside an ROI then data in the ROI will be printed, otherwise spectral data that are currently displayed on the screen will be printed.

save.pas saves the spectrum currently in the Buffer to a disk file.

recall.pas retrieves data which were previously saved in a disk file.

compare.pas prompts the user for a spectrum file name, reads and displays that spectrum, and compares it with the spectrum in the Buffer.

start.pas† contains several procedures:

start — starts data acquisition,

stop — stops data acquisition,

clear — clears the current spectrum,

auto_collect — changes the *auto/collect* mode,

set_drift_control — switches drift control on/off,

set_finesse_control — switches finesse control on/off.

spawn.pas includes two procedures whereby the user can execute a DOS command or program without quitting the MCS program.

ramp.pas† contains procedures for programming the external clock:

set_dwell_time — sets the dwell time of the external clock,

set_segment_mult — sets the segmented ramp multiplier,

set_segment — allows up to three slow segments to be defined in a spectrum,

erase_segment — deletes the segments which have previously been defined,

start_ramp — starts the external clock.

window.pas† is used to set the reference peak channel, drift window width, and finesse window width.

step.pas† is used to set the drift control correction step, finesse control test step, and finesse control correction step.

volt.pas† provides procedures that allow the user to set the initial drift and finesse

control voltages, and to increase/decrease voltages step by step manually.

help.pas‡ provides on-line help for the MCS system.

get_cmd.pas† is used to detect the keyboard and get the MCS commands.

do_cmd.pas† cails corresponding procedures listed above to perform the tasks defined by the function *get_cmd*.

initial.pas† performs the system initialization. It invokes the function *get_config* to get the old system configuration from the disk file *sys_file*. If *sys_file* does not exist or an error occurs during retrieval of the old configuration, then this procedure sets the default configuration of the system.

main.pas is the main program. It uses the function *get_cmd* to get a command and invokes the procedure *do_cmd* to execute that command. It performs system initialization in the beginning of the program and saves the system configuration to a disk file before the program ends.

Chapter 4

EXPERIMENTAL RESULTS

This MCS data acquisition and control system has been tested to record Brillouin scattering spectra of quartz, and performance was compared with that of the Burleigh DAS-1 system.

4.1 Experimental Setup

The typical experimental setup for Brillouin scattering studies is shown in Fig. 4.1. The incident light was produced by a single-mode Argon ion laser (Spectra Physics 165) tuned at a wavelength of 514.5 nm. The laser beam was focused on the sample by a lens $L1$. It was reflected upwards by a mirror M. A He-Ne laser beam was used to define the optical axis of the spectrometer. These two laser beams crossed at the sample and defined the location of the scattering volume. The scattered light passed through lenses $L2$ and $L3$ before entering the Fabry-Perot interferometer (Burleigh RC 110) which was equipped with plates flat to $\lambda/200$ at $\lambda = 500$ nm having a reflectivity of 92% for triple-pass operation. Light coming through the Fabry-Perot was focused by a lens $L4$, passed through a pinhole P, and then collected by a photomultiplier

(ITT FW 130). The output pulses from the photomultiplier were passed through an amplifier discriminator (PAR 1120) before being collected by the MCS system.

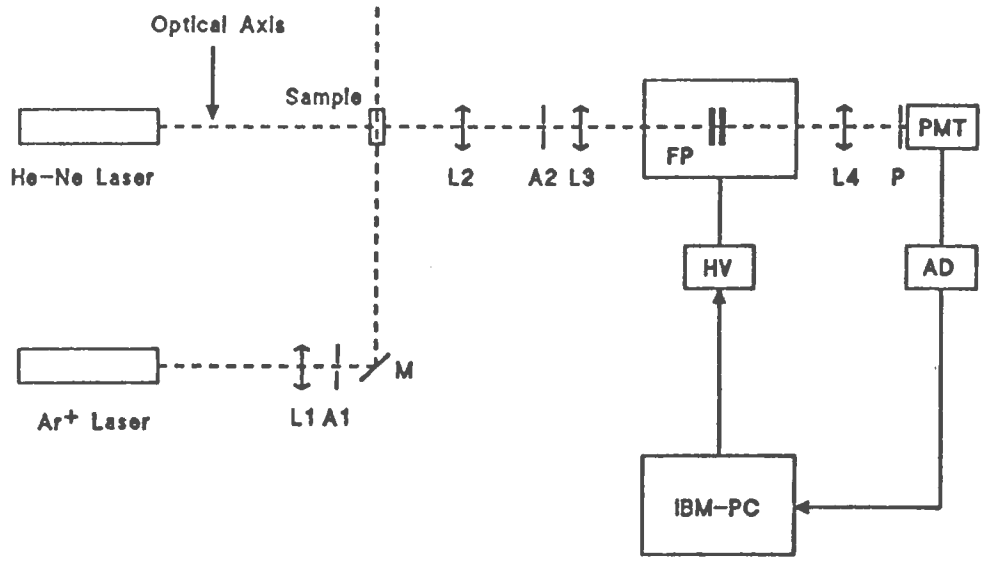


Figure 4.1: Block diagram of the experimental setup for Brillouin scattering. $A1$ – $A2$ apertures, $L1$ – $L4$ lenses, P pin-hole, M mirror, FP Fabry-Perot interferometer, PMT photomultiplier, AD amplifier-discriminator, HV high voltage amplifier

4.2 Drift and Finesse Stabilization

The Brillouin spectrum of fused silica (quartz) was used to assess the performance of the system. A typical spectrum is shown in Fig. 4.2 and consists of two pairs of symmetrically shifted components in addition to the strong central, or Rayleigh, component R. The components labelled L are due to scattering from longitudinal acoustic waves, and those labelled T are associated with scattering from transverse acoustic waves.

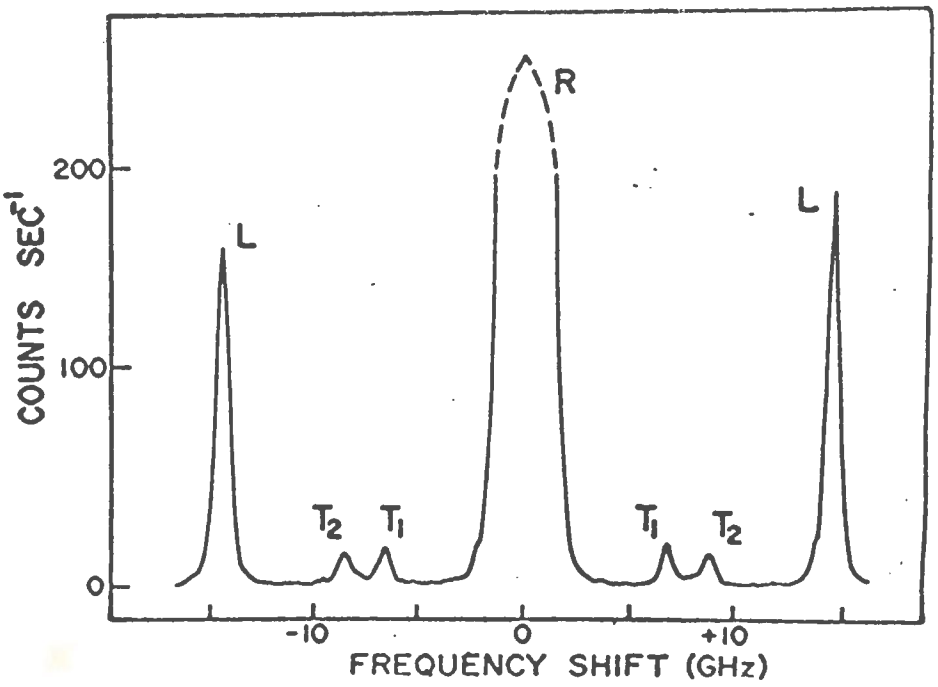


Figure 4.2: A typical Brillouin spectrum

Before each experiment was started, the Fabry-Perot interferometer was adjusted carefully to obtain the best alignment using the Rayleigh scattering from the sample. This was done in two steps. First the Fabry-Perot was set in single-pass mode in order to enhance the signal level. The drift control voltage V_D was adjusted to bring a selected Rayleigh peak to the preset reference channel. The other three finesse control voltages V_A, V_B, and V_C were then adjusted to maximize the peak height. The Fabry-Perot was then set to triple-pass operation.

Because the light transmitted in the triple-pass mode is weaker than for the single-pass mode, a slow segment encompassing the central peak was set whenever necessary to ensure adequate signal level. It was found that about 40 to 50 counts at the central peak were needed for the MCS system to work properly to maintain Fabry-Perot

alignment (c.f. 100 counts for the DAS-1). After further manual fine-tuning of the finesse, the automatic drift and finesse controls were turned on. The peak counts were seen to increase gradually within a few minutes and were subsequently maintained at the maximum level.

It is important that the MCS system be able to bring the Fabry-Perot back to the best alignment if for some reason misalignment has occurred. To test this, the drift control voltage was deliberately adjusted to position the central peak away from the preset reference channel. It was observed that the drift control voltage was automatically adjusted and the central peak was brought back to its previous position in a few sweeps, as long as the tail of the peak was initially within the drift windows. A similar behavior was observed after deliberate detuning of the finesse: the finesse was automatically maximized (over a somewhat longer period of time) as long as the central peak was initially within the finesse window. The results of further testing showed that the MCS system was capable of automatically maintaining optimum finesse indefinitely.

A series of spectra were taken to observe the behavior when the drift control and finesse control were turned off. Before the experiment was started, the Fabry-Perot interferometer was aligned carefully to produce the spectrum of Fig. 4.3 where the lower diagram shows the details of the essential features. The central channel was chosen at channel 520, with a drift window width of 8 channels and a finesse window width of 5 channels. The total number of channels was 1024, and the dwell time was

was 1 ms per channel. The segmented ramp was not used since the scattered light intensity was adequate.

At the beginning of the experiment, the count rate at the central channel was about 160 per sweep, and the FWHM of the central Rayleigh peak was 8 channels. The finesse of the Fabry-Perot did not degrade very much within the first two hours of the experiment. However, the spectral peak did drift to channel 518 with about 65 counts per sweep after two hours. The lines became noticeably broader after four hours, and the central peak drifted to channel 506 with 30 counts per sweep. The transverse peaks which were clearly seen at beginning of the experiment were obscured due to the drifting of the central peak. Twelve hours later, the central 'peak' had drifted to channel 486 with only 12 counts per sweep as shown in Fig. 4.4

Fig. 4.5 shows these eight spectra together. For the purpose of clarity, only the first 400 channels are shown. The data have been divided by the accumulation time for each spectrum, so that the vertical scale is average counts per second.

Experiments were also performed over the same time period for the conditions: (*a*) both drift and finesse control were turned on; and (*b*) drift control was on, but finesse control turned off. The results are shown in Fig. 4.6 and Fig. 4.7, respectively. The spectrum that was taken when both drift and finesse control were turned off is also shown in Fig. 4.8 for comparison. The vertical scales of the spectra are chosen the same. Although spectrum Fig. 4.7 is not as good as that of Fig. 4.6, the central peak is still at the same position, i.e., drift control was functioning well. The peak

heights of Fig. 4.7 are only 60% compared to Fig. 4.6, because of finesse control being turned off. The results show that the effects of drift and finesse control produce a much superior spectrum.

4.3 Segmented Ramp Scanning

Segmented ramp scanning was also tested on this MCS data acquisition system. Fig. 4.9 and Fig. 4.10 show the Brillouin spectrum of quartz observed by using a linear ramp and a segmented ramp, respectively. Both spectra used a dwell time of 1 ms per channel for normal, or fast, segments.

It took 2 hours to obtain the spectrum shown in Fig. 4.9 using linear ramp scanning. The total number of sweeps was 6540. The total counts at the transverse peak and longitudinal peak were 1800 and 10000, respectively, while the Rayleigh peak count was 860000. When the segmented ramp was used, it took only half an hour to get the spectrum of Fig. 4.10. Two regions of interest were set, namely, channel 543 to 564 for the transverse peak, and channel 700 to 733 for the longitudinal peak. The segment multiplier was 20, i.e., the time per channel spent in the region of interest was 20 ms, or twenty times longer than that in other normal regions. The total number of sweeps was 923, with a total count of 120000 at the Rayleigh peak, 260 at the transverse peak, and 1400 at the longitudinal peaks. But in the region of interest, the transverse and longitudinal peak counts were 4900 and 28000, respectively, i.e., twenty times higher than that in the normal region.

Since the scattered light from quartz in this experiment was strong enough for the

MCS system to obtain a spectrum without using the segmented ramp, the advantages of using this feature are not so obvious. However, if the transverse peak signal were extremely weak, then segmented ramp scanning is necessary to adequately define this peak within a reasonable period of time.

4.4 Suggested Further Modifications

The performance of the MCS system is comparable to that of the Burleigh DAS–1. With further modifications the performance of the system can be further improved.

1. Keyboard control of initial alignment in a dark laboratory is not as convenient as using switches and knobs as with the DAS–1. The latter should be introduced as optional controls.

2. The spectral graphic display of the MCS system works very well when the system is operating in the *collect* mode. However, it needs improving to meet the requirements for performing initial Fabry-Perot alignment. The *auto* mode is always selected for performing initial alignment, and a large segment multiplier is used when the signal level of the Brillouin lines is extremely weak. In this case, the spectral display on the MCS system is not as satisfactory as for the corresponding mode on the DAS–1, where the old spectrum is completely erased before starting the next sweep so that changes in the spectrum are clearly seen. By comparison, the MCS system simply *overwrites* the old data with the new, so that small changes in the Fabry-Perot finesse are not easily observed on the

monitor. Further modification of the 8086/8088 Assembly Language graphics routine or using a faster computer, e.g. with an 80386 CPU, can improve the performance of the spectral graphics display in these circumstances.

3. The external clock circuit for the segmented ramp is a separate unit installed in a small box. The switches *sw1* and *sw2* on the box need to be set to proper position when dwell time or segment multiplier are changed, which is somewhat inconvenient. This circuit may be built in the form of an IBM PC plug-in card, and the switches can be eliminated in favor of software control.

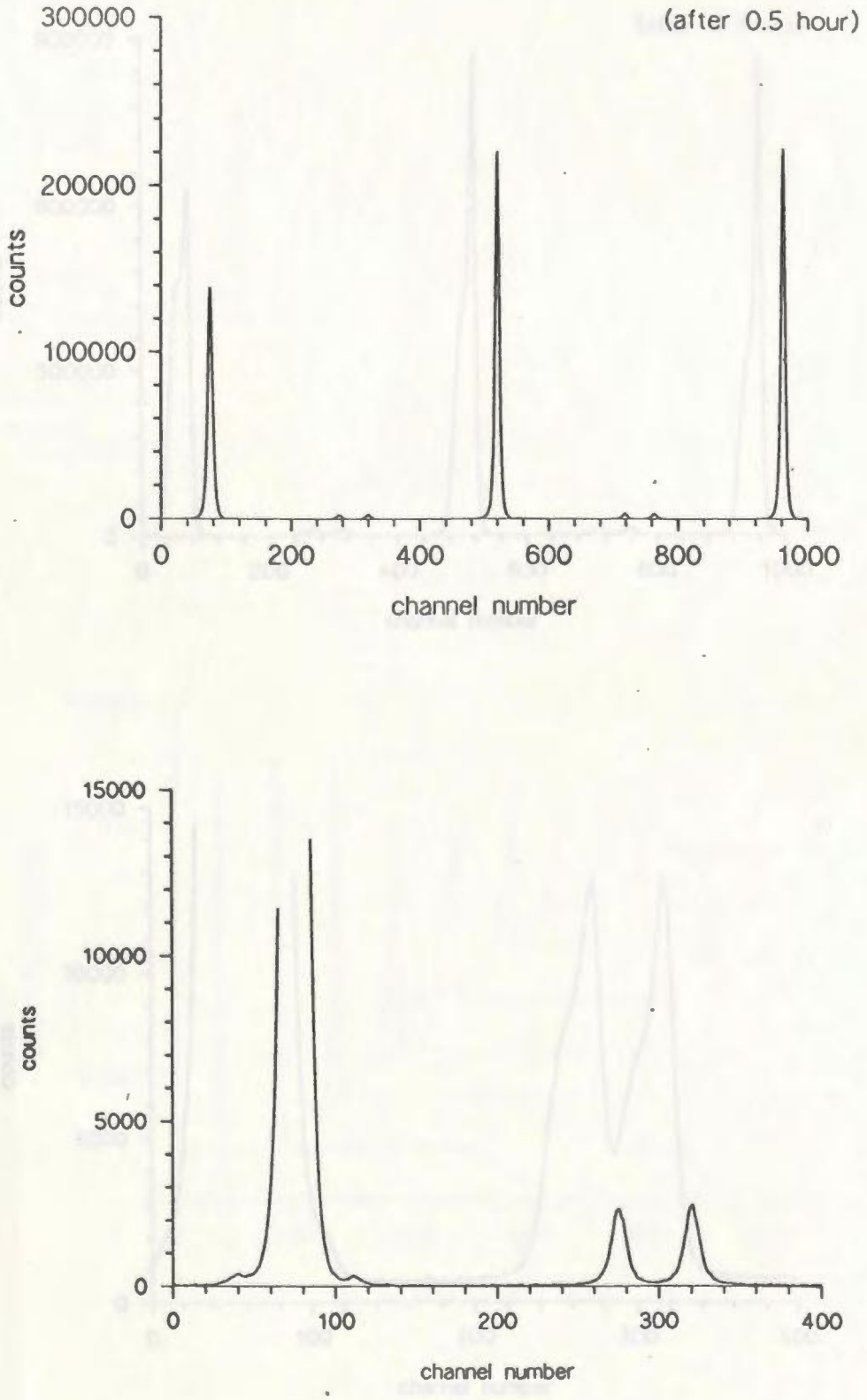


Figure 4.3: Brillouin spectrum of quartz after 0.5 hour

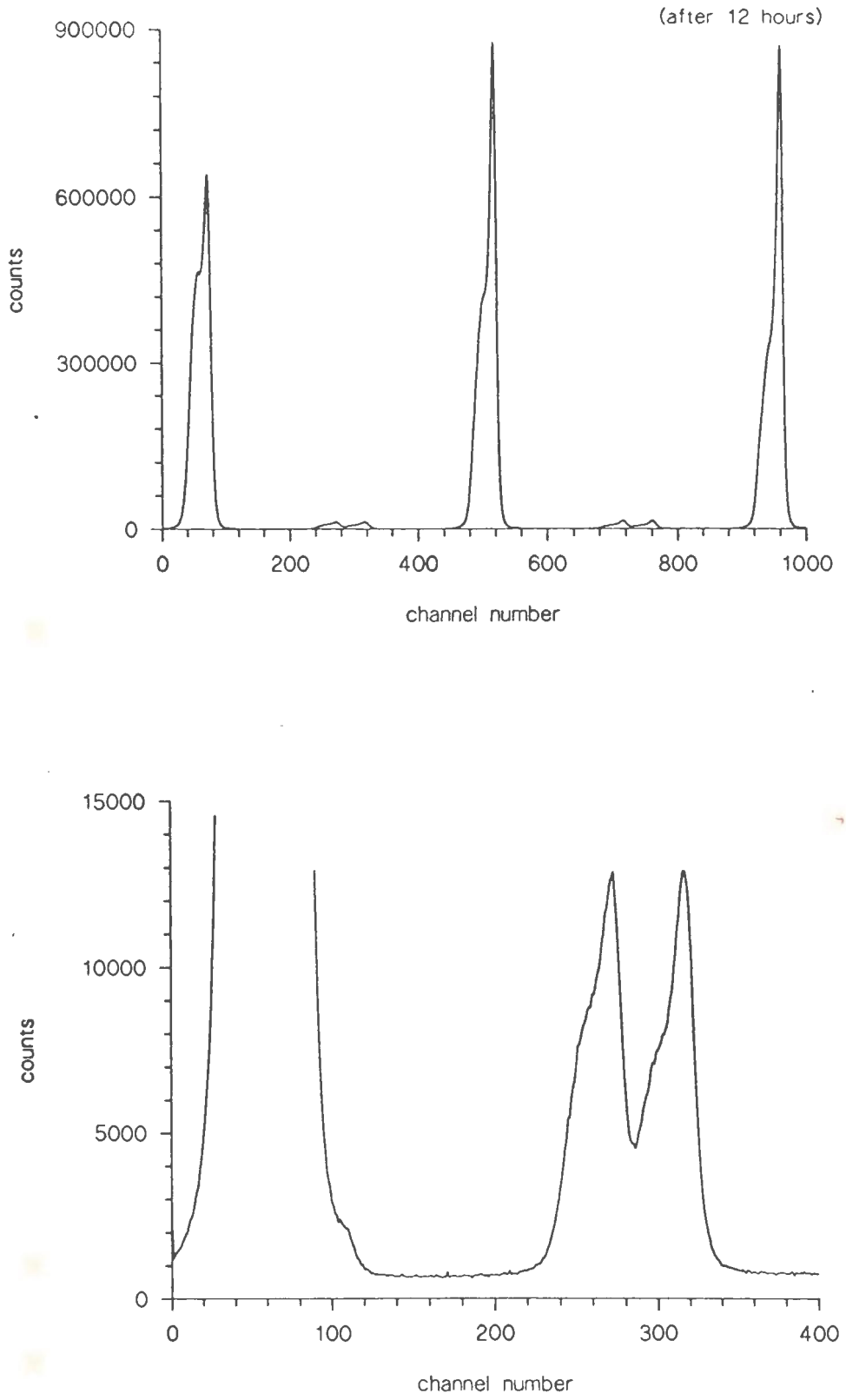


Figure 4.4: Brillouin spectrum of quartz after 12 hours

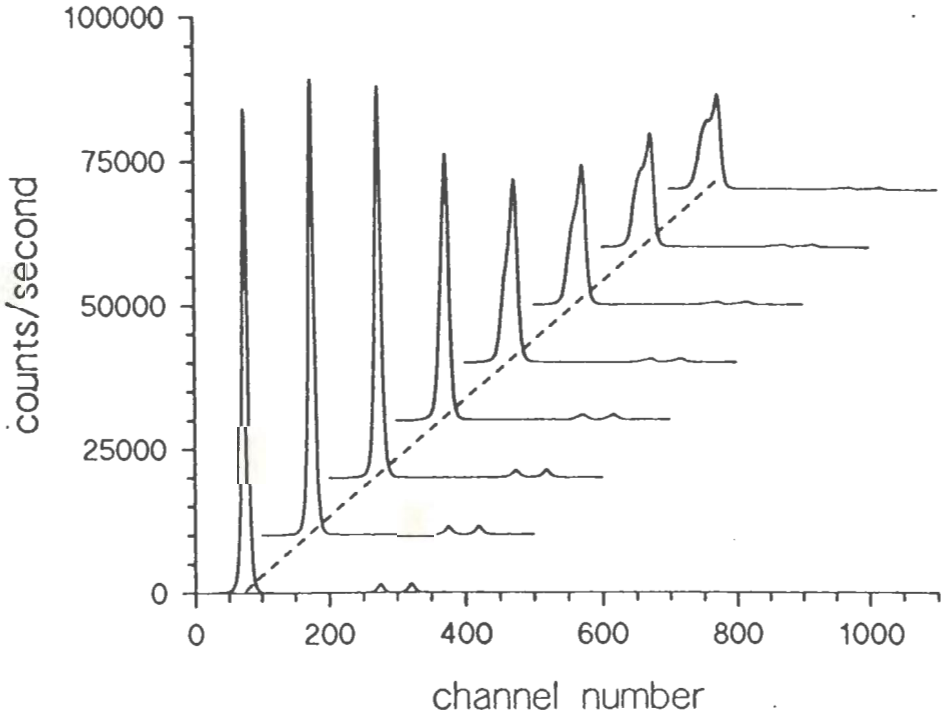


Figure 4.5: Average counts per second of the eight spectra

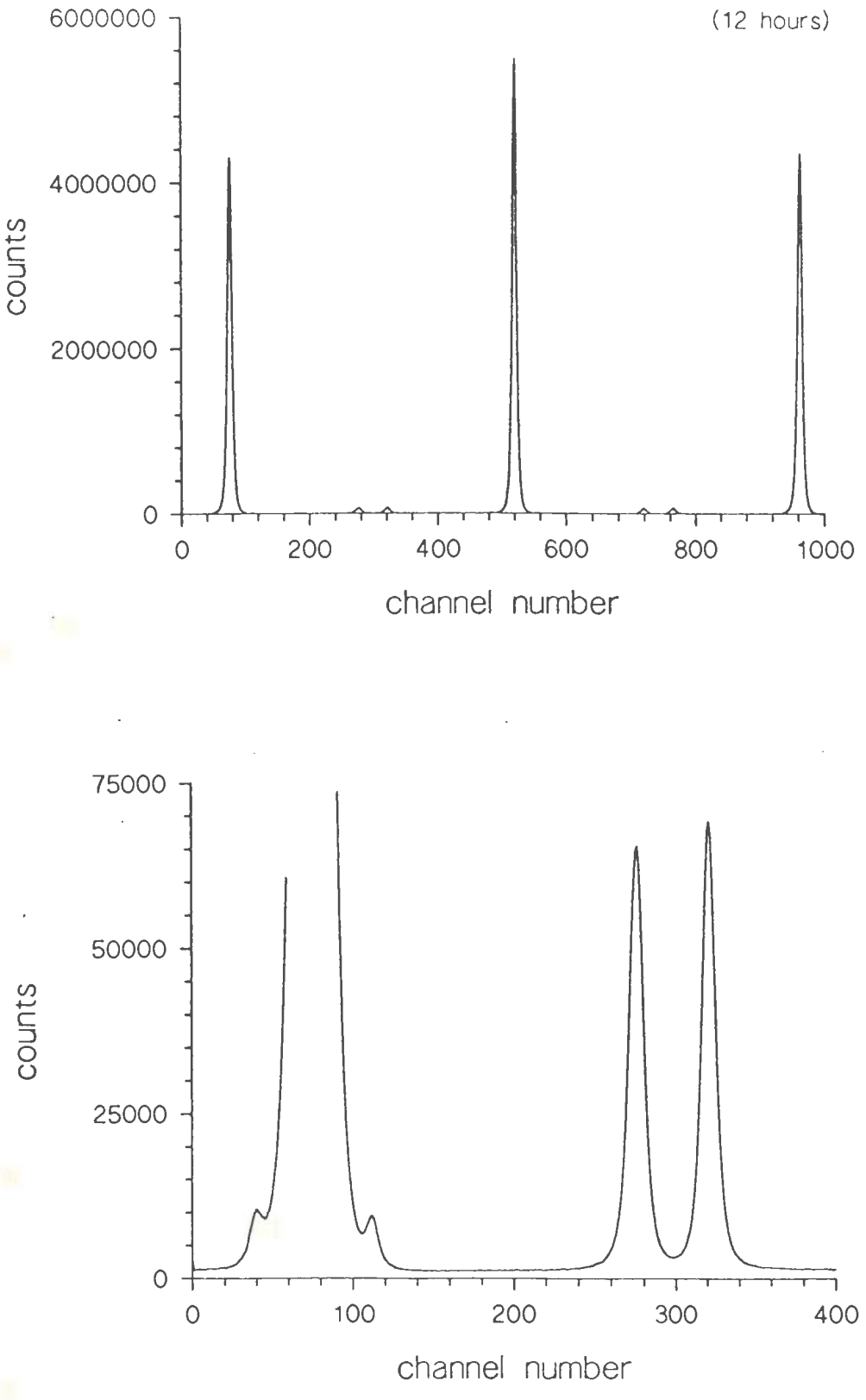


Figure 4.6: Brillouin spectrum of quartz, drift and finesse control ON

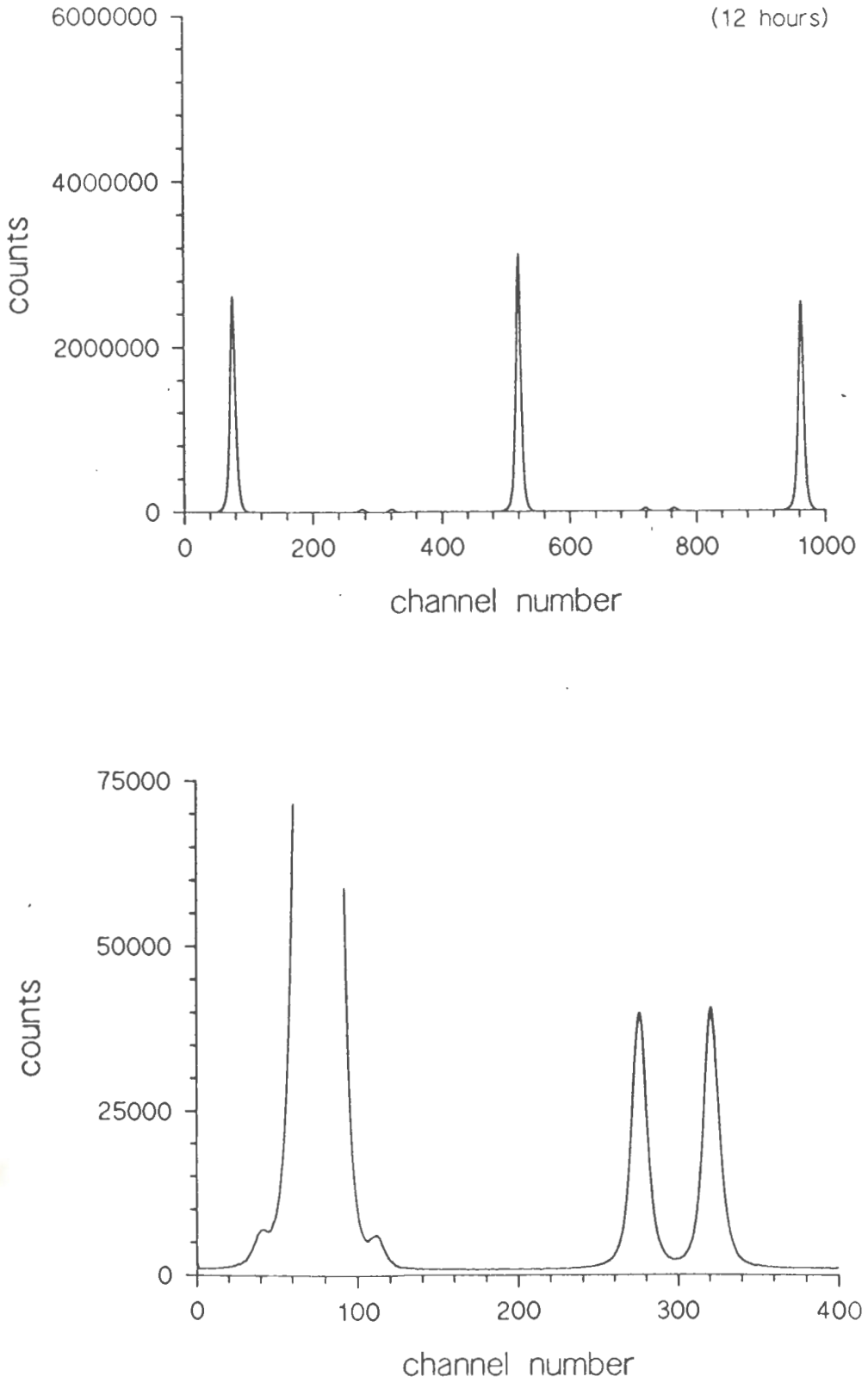
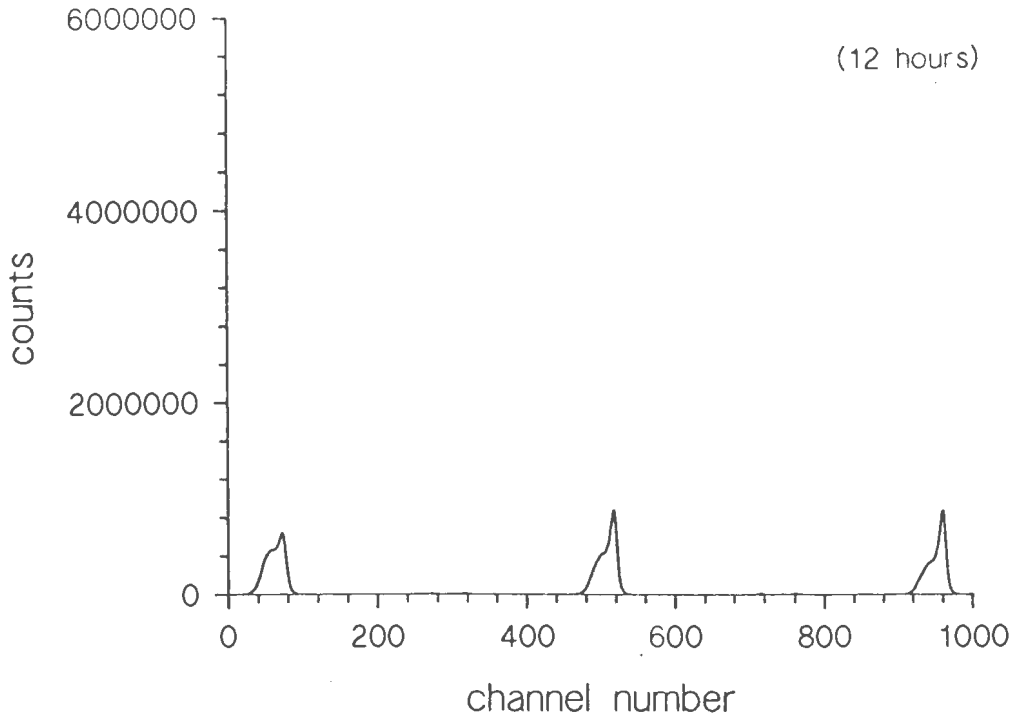


Figure 4.7: Brillouin spectrum of quartz, drift control ON, finesse control OFF



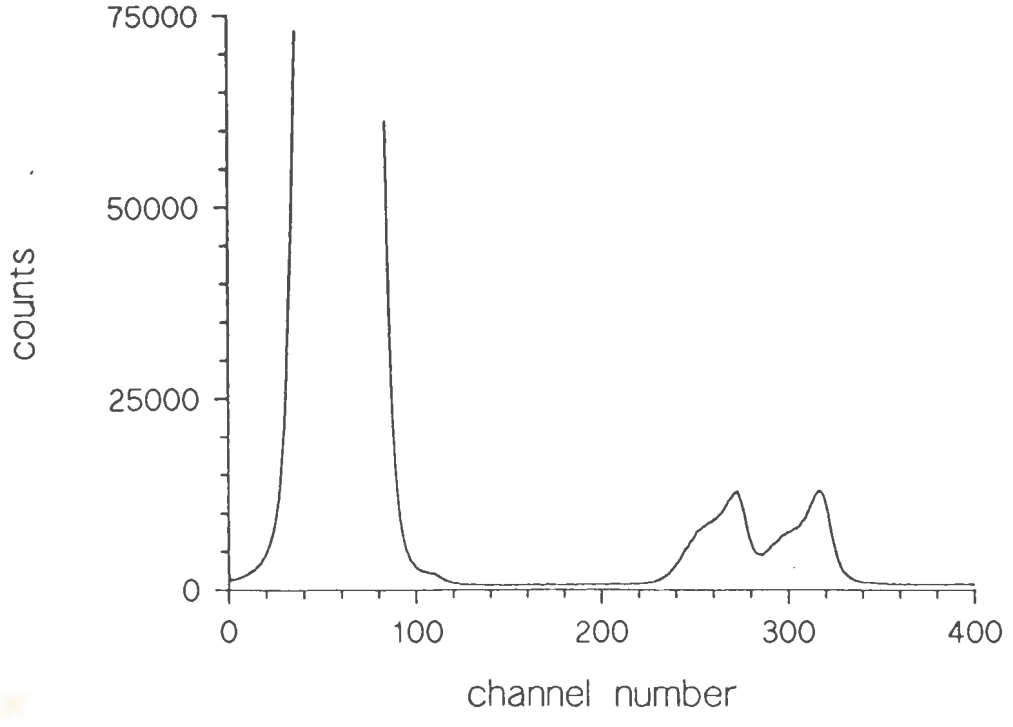


Figure 4.8: Brillouin spectrum of quartz, drift and finesse control OFF

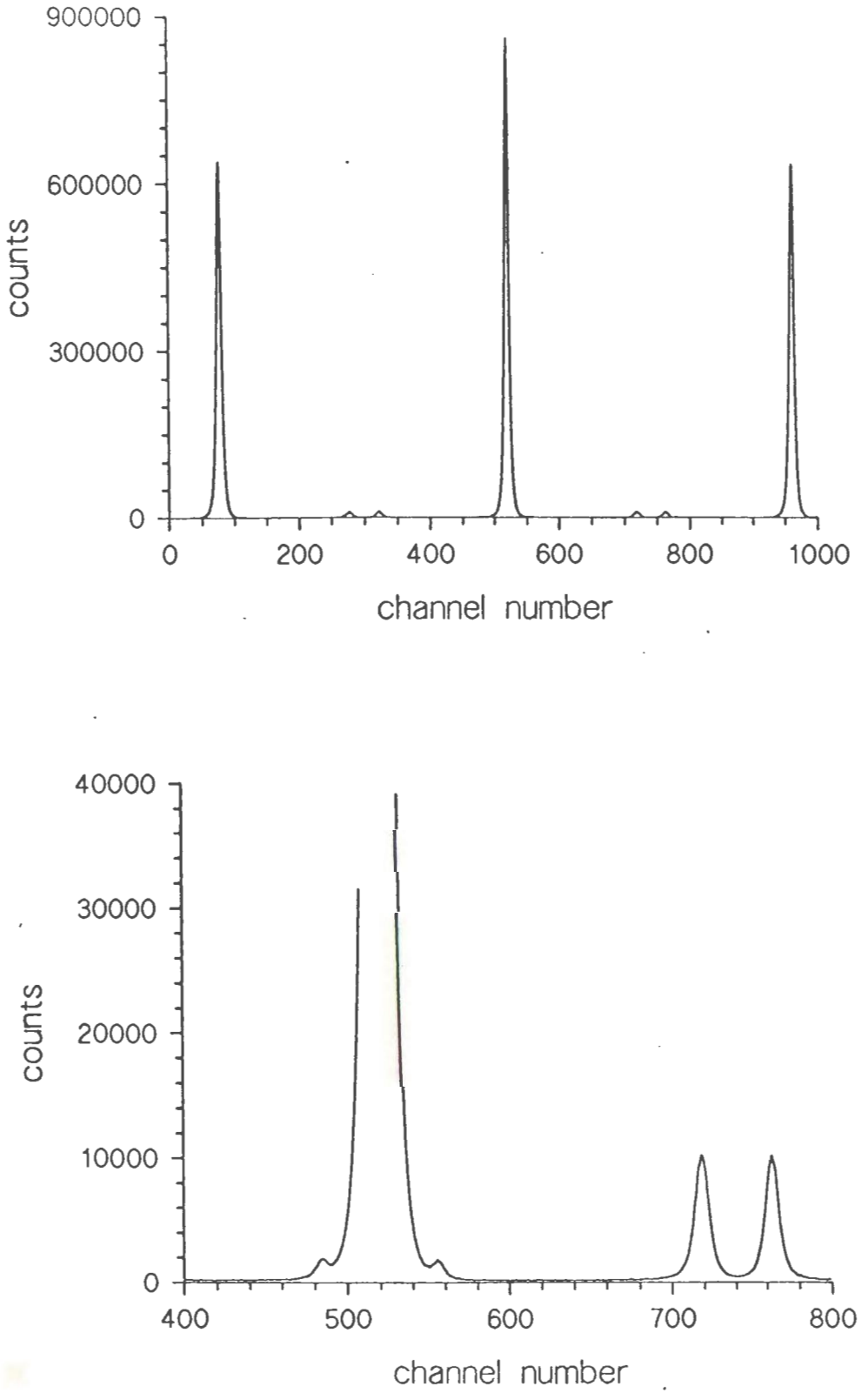


Figure 4.9: Brillouin spectrum of quartz, linear ramp

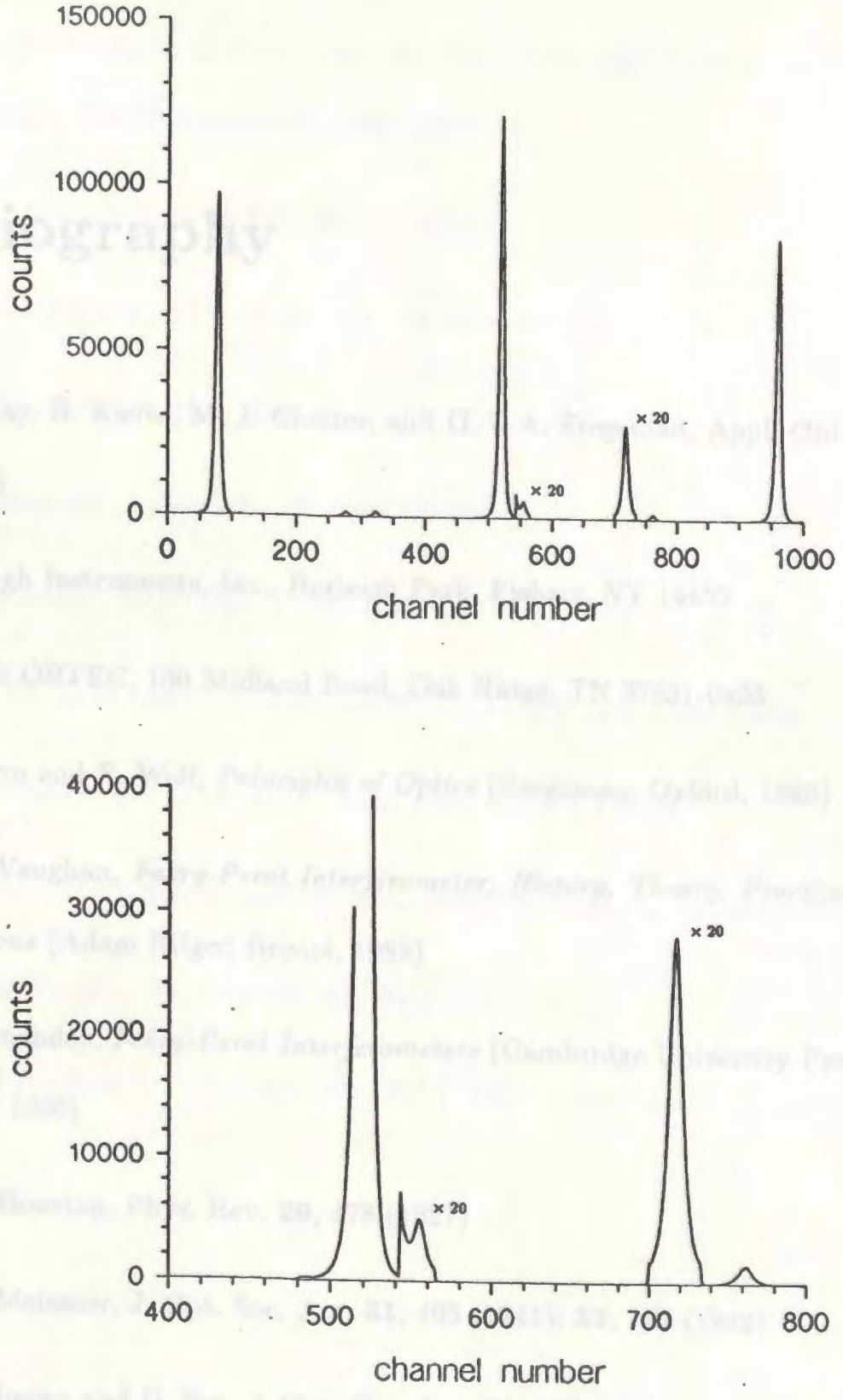


Figure 4.10: Brillouin spectrum of quartz, segmented ramp

Bibliography

[1] W. May, H. Kiefte, M. J. Clouter, and G. I. A. Stegeman, Appl. Opt. **17**, 1603 (1978)

[2] Burleigh Instruments, Inc., Burleigh Park, Fishers, NY 14453

[3] EG&G ORTEC, 100 Midland Road, Oak Ridge, TN 37831-0895

[4] M. Born and E. Wolf, *Principles of Optics* (Pergamon, Oxford, 1980)

[5] J. M. Vaughan, *Fabry-Perot Interferometer. History, Theory, Practice and Applications* (Adam Hilger, Bristol, 1988)

[6] G. Hernandez, *Fabry-Perot Interferometers* (Cambridge University Press, Cambridge, 1986)

[7] W. V. Houston, Phys. Rev. **29**, 478 (1927)

[8] K. W. Meissner, J. Opt. Soc. Am. **31**, 405 (1941); **32**, 185 (1942)

[9] P. Hariharan and D. Sen, J. Opt. Soc. Am. **51**, 398 (1961)

[10] J. R. Sandercock, Opt. Commun. **2**, 73 (1970)

[11] J. R. Sandercock, in *Proc. 2nd Int. Conf. on Light Scattering in Solids*, M. Balkanski, Ed. (Flammarion, Paris, 1971)

[12] J. R. Sandercock, RCA Rev. **36**, 89 (1975)

[13] J. R. Sandercock, Phys. Rev. Lett. **28**, 237 (1972)

[14] J. R. Sandercock, Phys. Rev. Lett. **29**, 1735 (1972)

[15] G. Hernandez, Appl. Opt. **17**, 3088 (1978)

[16] J. V. Ramsay, Appl. Opt. **1**, 411 (1962)

[17] R. A. McLaren and G. I. A. Stegeman, Appl. Opt. **12**, 1396 (1973)

[18] G. Hernandez and O. A. Mills, Appl. Opt. **12**, 126 (1973)

[19] Q. H. Lao, P. E. Schoen, and B. Chu, Rev. Sci. Instrum. **47**, 418 (1976)

[20] D. Bechtle, Rev. Sci. Instrum. **47**, 493 (1976)

[21] D. Bechtle, Rev. Sci. Instrum. **47**, 1377 (1976)

Appendix A

MCS Program Listing

This listing is not complete. It includes only those files which have either been extensively revised or introduced as completely new.

A.1 mcs.pas

```
{$title:'913 Control Program'}
{$linesize:120}
{$debug-}
{$symtab-}
{$floatcalls+}

{$message: 'including suputldd interface'}      {$include: 'suputldd.int'}
{$message: 'including filhanff interface'}      {$include: 'filhanff.int'}
{$message: 'including memorymm interface'}      {$include: 'memorymm.int'}
{$message: 'including calculat interface'}      {$include: 'calculat.int'}

{-----------------------------------------}
program mcs(input,output);
   uses suputldd,filhanff,memorymm,calculat;
{-----------------------------------------}


{
 12-19-86 Start with Maestro ver 4.05b and modify for the 913 mcs card.
          First change the variables and data structures.
 1-2-87   Worked on fkeys --- nothing tested yet and some stubs
 1-4-87   More changes to data structures, 913->buffer routines
 1-7-87   Start on presets, changes to get_cmd, do_cmd and presets
 1-9-87   Changed data structures from arrays over cur_913 to records.
          Will have to redo some of the other work above.
 1-17-87  Changed the buffer to sys[0].
```

```
 1-18-87  Program compiles now with many stubs. Activate start first.
          Added the isr routines for int3 support in file support.asm.
 1-20-87  Add stop, clear, save and recall spectrum.
 1-23-87  Add roi save and recall, type_dat, and compare. Not much left
          but calc functions now.
 2-24-87  All calc functions except report done. Need library lookup still
          and parse but ready for a lot of testing now.
 3-27-87  Everything is done. Made change to sys structure for handling
          external dwell with fast times.
 4-06-87  Minor change in fill_buffer so calib routine cycles correctly.
 6-05-87  Modified asm routines to eliminate lodsw, which doesn't work on
          the new IBM model 30. Ready for release - name Ver 1.1.
 11-06-87 Fix to allow pass preset of four byte integer. Ver 1.2.
 1-19-88  Fix loading of pass length problem in start.pas. Ver 1.3

 7-06-90  add drift and finesse stabilization, external clock for
          segmented ramp scanning, and provide help.  (ST.)
}



{$message:'input the number 1 for Enhanced Graphics Adaptor, 0 for other'}
{$inconst:ega}
{$if ega $then}
  {$message:'compiling EGA version of MCS.PAS'}
{$else}
  {$message:'compiling non-EGA version of MCS.PAS'}
{$end}

{$message: 'defining constant'}              {$include: 'const.inc'}
{$message: 'defining type'}                  {$include: 'type.inc'}
{$message: 'defining variables'}             {$include: 'var.inc'}
{$message: 'defining external procedures'}   {$include: 'proc.inc'}

{$message: 'including procedures ...'}
{$message: 'including general'}          {$include: 'general.pas'}
{$message: 'including display'}          {$include: 'display.pas'}
{$message: 'including do_scrn'}          {$include: 'do_scrn.pas'}
{$message: 'including config'}           {$include: 'config.pas'}
{$message: 'including setup'}            {$include: 'setup.pas'}
{$message: 'including update'}           {$include: 'update.pas'}
{$message: 'including fkeys'}            {$include: 'fkeys.pas'}
{$message: 'including marker'}           {$include: 'marker.pas'}
{$message: 'including roi'}              {$include: 'roi.pas'}
{$message: 'including buffer'}           {$include: 'buffer.pas'}
```

```
{$message: 'including presets'}          {$include: 'presets.pas'}
{$message: 'including type_dat'}         {$include: 'type_dat.pas'}
{$message: 'including save'}             {$include: 'save.pas'}
{$message: 'including recall'}           {$include: 'recall.pas'}
{$message: 'including compare'}          {$include: 'compare.pas'}
{$message: 'including start'}            {$include: 'start.pas'}
{$message: 'including spawn'}            {$include: 'spawn.pas'}
{$message: 'including ramp'}             {$include: 'ramp.pas'}
{$message: 'including window'}           {$include: 'window.pas'}
{$message: 'including step'}             {$include: 'step.pas'}
{$message: 'including volt'}             {$include: 'volt.pas'}
{$message: 'including help'}             {$include: 'help.pas'}
{$message: 'including get_cmd1'}         {$include: 'get_cmd1.pas'}
{$message: 'including get_cmd2'}         {$include: 'get_cmd2.pas'}
{$message: 'including get_cmd3'}         {$include: 'get_cmd3.pas'}
{$message: 'including do_cmd'}           {$include: 'do_cmd.pas'}
{$message: 'including initial'}          {$include: 'initial.pas'}

{$message: 'beginning main program'}     {$include: 'main.pas'}
{$message: '--- end of MCS.PAS ---'}
```

A.2 ramp.pas

```
{-- programming external clock for segmented ramp  July 1990, ST. --}

  procedure set_dwell_time;
    const
      delay_const = 100e3; {100mS}

    var
      dwell_int4 : integer4;
      num_string : lstring(20);

    begin {set_dwell_time}
      prompt('Enter external clock dwell time: (10..65535 uS) ');
      read_string(num_string,upper(num_string));
      if (decode(num_string,dwell_int4)) then begin
        if ((dwell_int4 >= 10) and (dwell_int4 <= 65535)) then begin

          {-- in order to write dwell_time into sys_file.913 --}
          sys[cur_913].dwell_time := dwell_int4;

          ramp_count_dwell := wrd(dwell_int4);  {convert to word type}
          eval(encode(message, ramp_count_dwell:5));
```

```
          write_string(message, r_dwell_disp, c_dwell_disp);

          {-- set delay time >= 100 mS, note: /word_tpye is not allowed}
          ramp_count_delay := round(delay_const /dwell_int4);
          if (ramp_count_delay < 1) then ramp_count_delay := 1;

          prompt('please re-start the external clock');
        end
        else begin {dwell_int4 <10 or >65535}
          prompt('Value not in range -- aborting set dwell time');
        end;
      end
      else begin
        prompt('Error in the input, aborting set dwell time');
      end;
    end; {set_dwell_time}

{---------------------------------------}
  procedure set_segment_mult;
    var
      temp_mult          : integer;
      num_string         : lstring(20);

    begin
      prompt('Enter segment multiplier : (2..999) ');
      read_string(num_string,upper(num_string));
      if (decode(num_string, temp_mult)) then begin
        if ((temp_mult >= 2) and (temp_mult <= 999)) then begin
          ramp_count_mult := temp_mult;
          eval(encode(message, ramp_count_mult:-3));
          write_string(message, r_mult_disp, c_mult_disp);
          white_line;
        end
        else begin {temp_mult <=0 or >999}
          prompt('Value not in range -- aborting set multiplier');
        end;
      end
      else begin
        prompt('Error in the input, aborting set multiplier');
      end;
    end; {set_segment_mult}

{---------------------------------------}
  procedure remind_switch;
    var
```

```
    message2 : lstring(10);
  begin
    message := 'Please set switch to ';
    message2 := '''Linear''';
    if (seg1_start <> 0) then message2 := '''1''';
    if (seg2_start <> 0) then message2 := '''2''';
    if (seg3_start <> 0) then message2 := '''3''';
    concat(message, message2);
    concat(message, ' on the Ramp-Clock-Box');
    prompt(message);
  end; {remind_switch}

{---------------------------------------}
  procedure set_segment;
    var
      get_new_segment  : boolean;
      temp_start,
      temp_end         : integer;
      num_string       : lstring(20);

    function get_start_end : boolean;
      var
        max_ch,
        exchange : integer;
      begin
        max_ch := sys[cur_913].pass_length - 1;
        prompt('Enter segment start channel : ');
        read_string(num_string,upper(num_string));
        if (decode(num_string,temp_start)) then begin
          if ((temp_start > 0) and (temp_start < max_ch)) then begin
            {go on to get_end}
          end
          else begin
            prompt('Value not in range -- aborting set segment');
            get_start_end := false;
            return;
          end;
        end {if decode}
        else begin
          prompt('Error in the input, aborting set segment');
          get_start_end := false;
          return;
        end; {not decode}

        prompt('Enter segment end channel : ');
```

```
    read_string(num_string,upper(num_string));
    if (decode(num_string,temp_end)) then begin
      if ((temp_end > 0) and (temp_end < max_ch)) then begin
        if (temp_start < temp_end) then begin
          get_start_end := true;
        end
        else if (temp_start > temp_end) then begin
          exchange := temp_start;
          temp_start := temp_end;
          temp_end := exchange;
          get_start_end := true;
        end
        else begin
prompt('start-channel = end-channel! -- aborting set segment');
          get_start_end := false;
        end;
      end
      else begin
        prompt('Value not in range -- aborting set segment');
        get_start_end := false;
      end;
    end {if decode}
    else begin
      prompt('Error in the input, aborting set segment');
      get_start_end := false;
    end;
  end; {get_start_end}

begin {set_segment}
  get_new_segment := false;

  if (seg1_start = 0) then begin {no segment defined}
    if (get_start_end) then begin
      seg1_start := temp_start;
      seg1_end   := temp_end;
      get_new_segment := true;
    end;
  end {seg1 = 0}
  else if (seg2_start = 0) then begin {only seg1 defined}
    if (get_start_end) then begin
      if (temp_start > seg1_end +1) then begin
        seg2_start := temp_start;
        seg2_end   := temp_end;
        get_new_segment := true;
      end
```

```
    else if (temp_end < seg1_start -1) then begin
      seg2_start := seg1_start;
      seg2_end   := seg1_end;
      seg1_start := temp_start;
      seg1_end   := temp_end;
      get_new_segment := true;
    end
    else begin
      prompt('segment overlap!  -- aborting set segment');
    end;
  end; {get_start_end}
end {seg2 = 0}
else if (seg3_start = 0) then begin
  if (get_start_end) then begin
    if (temp_start > seg2_end +1) then begin
      seg3_start := temp_start;
      seg3_end   := temp_end;
      get_new_segment := true;
    end
    else if ( (temp_start > seg1_end +1)
          and (temp_end < seg2_start -1) ) then begin
      seg3_start := seg2_start;
      seg3_end   := seg2_end;
      seg2_start := temp_start;
      seg2_end   := temp_end;
      get_new_segment := true;
    end
    else if (temp_end < seg1_start -1) then begin
      seg3_start := seg2_start;
      seg3_end   := seg2_end;
      seg2_start := seg1_start;
      seg2_end   := seg1_end;
      seg1_start := temp_start;
      seg1_end   := temp_end;
      get_new_segment := true;
    end
    else begin
      prompt('segment overlap!  -- aborting set segment');
    end;
  end; {get_start_end}
end {seg3 = 0}
else begin {3 segments are set}
  prompt('You can''t have more than three segments');
end;
```

```
      if (get_new_segment) then begin
        disp_segment;
        remind_switch;
      end;
    end; {set_segment}

{-----------------------------------------}
  procedure delete_segment;
    var
      deleted     : boolean;
      temp_delete : integer;
      num_string : lstring(20);

    begin
      deleted := false;

      if (seg1_start = 0) then begin
        {no segment exist}
      end
      else begin {seg1 <> 0}
        prompt('Enter segment number to be deleted : ');
        read_string(num_string,upper(num_string));
        if (decode(num_string,temp_delete)) then begin
          if (temp_delete = 1) then begin
            seg1_start := seg2_start;
            seg1_end   := seg2_end;
            seg2_start := seg3_start;
            seg2_end   := seg3_end;
            seg3_start := 0;
            seg3_end   := 0;
            deleted := true;
          end
          else if (temp_delete = 2) then begin
            seg2_start := seg3_start;
            seg2_end   := seg3_end;
            seg3_start := 0;
            seg3_end   := 0;
            deleted := true;
          end
          else if (temp_delete = 3) then begin
            seg3_start := 0;
            seg3_end   := 0;
            deleted := true;
          end
          else begin {temp_delete <> 1,2,3}
```

```
            prompt('Value not in range -- aborting delete segment');
          end;
        end {if deccde}
        else begin
          prompt('Error in the input, aborting delete segment');
        end;
      end; {seg1 <> 0}

      if (deleted) then begin
        disp_segment;
        remind_switch;
      end;
    end;{delete_segment}

{-------------------------------------------------------------}
  procedure start_ramp;

{
  Jan. 9, 1990
  program 8253 Timer/Counter on card0, chip1, chip2, chip3
  generate pulses of two frequences for segmented ramp voltage

  Card_0 counter 0 and counter 1 generate clock T1, T2 (ie. dwell, mult)
  Card_0 Binary_output_port interfacing to 8253 chips
  B0...B7 ----- D0...D7
  B8  --------- A0
  B9  --------- A1
  B10 --------- WR
  B11 --------- CS_chip1 (delay, n1, n2)
  B12 --------- CS_chip2 (n3, n4, n5)
  B13 --------- CS_chip3 (n6, n7    )
  B14 --------- (NC.)
  B15 --------- (NC.)

  example:

  card0, counter0
    write mode:         out(card0_counter_ctrl, c0_mode1_ctrl_byte);
    write count:        out(card0_c0_lo, count_lo);
                        out(card0_c0_hi, count_hi);

  chip1, counter1
    write mode:         out(card0_binary_out_lo, c1_mode1_ctrl_byte);
                        out(card0_binary_out_hi, chip1_ctrl_reg + wr_high);
                        out(card0_binary_out_hi, chip1_ctrl_reg + wr_low);
```

```
                        out(card0_binary_out_hi, chip1_ctrl_reg + wr_high);
    write count:        out(card0_binary_out_lo, count_lo);
                        out(card0_binary_out_hi, chip1_c1 + wr_high);
                        out(card0_binary_out_hi, chip1_c1 + wr_low);
                        out(card0_binary_out_hi, chip1_c1 + wr_high);
                        out(card0_binary_out_lo, count_hi);
                        out(card0_binary_out_hi, chip1_c1 + wr_high);
                        out(card0_binary_out_hi, chip1_c1 + wr_low);
                        out(card0_binary_out_hi, chip1_c1 + wr_high);
}

    const
      card0_dev_num_reg_lo = 16#C2E2; {reg_hi not used}
      card0_binary_out_lo = 16#22E2;
      card0_binary_out_hi = 16#22E3;  {dev_number of Binary_out_reg =8}

      card0_c0_lo = 16#82E2;
      card0_c0_hi = 16#82E3;
      card0_c1_lo = 16#92E2;
      card0_c1_hi = 16#92E3;
      card0_counter_ctrl = 16#B2E2;   {reg_hi not used}

      c0_mode1_ctrl_byte = 2#00110010;
      c1_mode1_ctrl_byte = 2#01110010;
      c2_mode1_ctrl_byte = 2#10110010;
      c0_mode2_ctrl_byte = 2#00110100;
      c1_mode2_ctrl_byte = 2#01110100;
      c2_mode2_ctrl_byte = 2#10110100;

      chip1_c0   = 2#00110000;  {B11 (CS_chip1) = low}
      chip1_c1   = 2#00110001;
      chip1_c2   = 2#00110010;
      chip1_ctrl = 2#00110011;

      chip2_c0   = 2#00101000;  {B12 (CS_chip2) = low}
      chip2_c1   = 2#00101001;
      chip2_c2   = 2#00101010;
      chip2_ctrl = 2#00101011;

      chip3_c0   = 2#00011000;  {B13 (CS_chip3) = low}
      chip3_c1   = 2#00011001;
      chip3_c2   = 2#00011010;
      chip3_ctrl = 2#00011011;

      wr_low  = 2#00000000;
```

```
  wr_high = 2#00000100;


procedure write_mode;
  procedure write_card0_mode(ctrl_byte : byte);
    begin
      out(card0_counter_ctrl, ctrl_byte);
    end; {write_card0_mode}

  procedure write_chip_mode(ctrl_reg, ctrl_byte : byte);
    begin
      out(card0_binary_out_lo, ctrl_byte);
      out(card0_binary_out_hi, ctrl_reg + wr_high);
      out(card0_binary_out_hi, ctrl_reg + wr_low);
      out(card0_binary_out_hi, ctrl_reg + wr_high);
    end; {write_chip_mode}

  begin {write_mode}
    write_card0_mode(c0_mode2_ctrl_byte);
    write_card0_mode(c1_mode2_ctrl_byte);
    write_chip_mode(chip1_ctrl, c0_mode1_ctrl_byte);
    write_chip_mode(chip1_ctrl, c1_mode1_ctrl_byte);
    write_chip_mode(chip1_ctrl, c2_mode1_ctrl_byte);
    write_chip_mode(chip2_ctrl, c0_mode1_ctrl_byte);
    write_chip_mode(chip2_ctrl, c1_mode1_ctrl_byte);
    write_chip_mode(chip2_ctrl, c2_mode1_ctrl_byte);
    write_chip_mode(chip3_ctrl, c0_mode1_ctrl_byte);
    write_chip_mode(chip3_ctrl, c1_mode1_ctrl_byte);
  end; {write_mode}

procedure write_count;
  procedure write_card0_dwell(count:word);
    begin
      out(card0_c0_lo, lobyte(count));
      out(card0_c0_hi, hibyte(count));
    end;

  procedure write_card0_mult(count:integer);
    begin
      out(card0_c1_lo, lobyte(count));
      out(card0_c1_hi, hibyte(count));
    end;

  procedure write_chip_count(chip_counter:byte; count:integer);
    begin
```

```
      out(card0_binary_out_lo, lobyte(count));
      out(card0_binary_out_hi, chip_counter + wr_high);
      out(card0_binary_out_hi, chip_counter + wr_low);
      out(card0_binary_out_hi, chip_counter + wr_high);
      out(card0_binary_out_lo, hibyte(count));
      out(card0_binary_out_hi, chip_counter + wr_high);
      out(card0_binary_out_hi, chip_counter + wr_low);
      out(card0_binary_out_hi, chip_counter + wr_high);
    end;

  procedure set_cs_wr_high;
    begin
      out(card0_binary_out_lo, 2#00000000);{data line=low}
      out(card0_binary_out_hi, 2#00111100);{cs, wr =high}
    end;

  begin {write_count}
    write_card0_dwell(ramp_count_dwell);
    write_card0_mult (ramp_count_mult);
    write_chip_count(chip1_c0, ramp_count_delay);
    write_chip_count(chip1_c1, ramp_count_1);
    write_chip_count(chip1_c2, ramp_count_2);
    write_chip_count(chip2_c0, ramp_count_3);
    write_chip_count(chip2_c1, ramp_count_4);
    write_chip_count(chip2_c2, ramp_count_5);
    write_chip_count(chip3_c0, ramp_count_6);
    write_chip_count(chip3_c1, ramp_count_7);
    set_cs_wr_high;
  end; {write_count}

procedure get_count;
  begin
    ramp_count_1 := sys[cur_913].pass_length;   {linear}
    ramp_count_2 := 1;
    ramp_count_3 := 1;
    ramp_count_4 := 1;
    ramp_count_5 := 1;
    ramp_count_6 := 1;
    ramp_count_7 := 1;

    if (seg1_end <> 0) then begin
      ramp_count_1 := seg1_start;
      ramp_count_2 := seg1_end - seg1_start + 1;
      ramp_count_3 := sys[cur_913].pass_length
       - ramp_count_1 - ramp_count_2;
```

```
      end; {seg1 <> 0}

      if (seg2_end <> 0) then begin
        ramp_count_3 := seg2_start - seg1_end - 1;
        ramp_count_4 := seg2_end - seg2_start + 1;
        ramp_count_5 := sys[cur_913].pass_length
         - ramp_count_1 - ramp_count_2
         - ramp_count_3 - ramp_count_4;
      end; {seg2 <> 0}

      if (seg3_end <> 0) then begin
        ramp_count_5 := seg3_start - seg2_end - 1;
        ramp_count_6 := seg3_end - seg3_start + 1;
        ramp_count_7 := sys[cur_913].pass_length
         - ramp_count_1 - ramp_count_2
         - ramp_count_3 - ramp_count_4
         - ramp_count_5 - ramp_count_6;
      end; {seg3 <> 0}
    end; {get_count}

  begin {start_ramp}
    if (not sys[cur_913].external_start) then begin
      prompt('Please choose external start by setup-config.');
    end
    else if (not sys[cur_913].external_dwell) then begin
      prompt('Please choose external dwell by setup-config.');
    end
    else begin {ext-start and ext-dwell}
      out (Card0_dev_num_reg_lo, 8);    {enable binary port}
      write_mode;
      get_count;
      write_count;
      out (Card0_dev_num_reg_lo, 9);    {enable dac port}
      prompt('Please press RESET and START key on the Ramp-Clock-Box');
    end;
  end; {start_ramp}
```

A.3 window.pas

```
{set reference channel, drift window, and finesse window  ST.}

  procedure set_window;
    var
      temp_ref,
```

```
  temp_dr,
  temp_fi,
  total_chns,
  index              : integer;
  num_string         : lstring(20);

function get_ref : boolean;
  begin
    get_ref := false;
    prompt('Enter reference peak channel:');
    read_string(num_string,upper(num_string));
    if (decode(num_string,temp_ref)) then begin
      total_chns := sys[cur_913].pass_length;
      if ((temp_ref > 1) and (temp_ref < total_chns)) then begin
        get_ref := true;
      end {0 < temp_ref < total_chns}
      else begin
        prompt('Value not in range -- aborting set window');
      end; {temp_ref out of range}
    end {if decode}
    else begin
      prompt('Error in the input, aborting set window');
    end;
  end; {get_ref}

function get_drift_window : boolean;
  var
     dr_left, dr_right : integer;
  begin
    get_drift_window := false;
    prompt('Enter drift window width: (on each side of the ref-peak)');
    read_string(num_string,upper(num_string));
    if (decode(num_string,temp_dr)) then begin
      if ((temp_dr > 0) and (temp_dr < 100)) then begin
        dr_left  := temp_ref - temp_dr;
        dr_right := temp_ref + temp_dr;
        total_chns := sys[cur_913].pass_length;

        if ((dr_left > 0) and (dr_right < total_chns)) then begin
          get_drift_window := true;
        end
        else begin
          prompt('drift window out of range -- aborting set window');
        end;
      end {0 < temp_dr < 100}
```

```
      else begin
        prompt('Value not in range -- aborting set window');
      end;
    end {if decode}
    else begin
      prompt('Error in the input, aborting set window');
    end;
  end; {get_drift_window}

function get_finesse_window : boolean;
  var
     fi_left, fi_right : integer;
  begin
    get_finesse_window := false;
    prompt('Enter finesse window width: (centered on the ref-peak)');
    read_string(num_string,upper(num_string));
    if (decode(num_string,temp_fi)) then begin
      if ((temp_fi > 0) and (temp_fi < 100)) then begin
        temp_fi := (temp_fi div 2) * 2 + 1; {odd number}
        fi_left  := temp_ref - temp_fi;
        fi_right := temp_ref + temp_fi;
        total_chns := sys[cur_913].pass_length;

        if ((fi_left > 0) and (fi_right < total_chns)) then begin
          get_finesse_window := true;
        end
        else begin
          prompt('finesse window out of range -- aborting set window');
        end;
      end {0 < temp_fi < 100}
      else begin
        prompt('Value not in range -- aborting set window');
      end;
    end {if decode}
    else begin
      prompt('Error in the input, aborting set window');
    end;
  end; {get finesse window}

procedure erase_window_mark;
  var
    left, right, index : integer;
  begin
    left  := ref_channel - dr_window_width;
    right := ref_channel + dr_window_width;
```

```
    for index := left to right do begin
      adcam1^[index] := adcam1^[index] and 16#ffffff;
    end;
  end;

procedure mark_drift_window;
  var
    left, right, index : integer;
  begin
    left  := ref_channel - dr_window_width;
    right := ref_channel + dr_window_width;
    for index := left to right do begin
      adcam1^[index] := adcam1^[index] or 16#80000000;
    end;
  end;

begin {set_window}
  drift_control := false;
  finesse_control := false;
  sweep_count := 0;
  disp_drift_control;
  disp_finesse_control;

  disp_window;
  erase_window_mark;

  if (get_ref) then begin
    eval(encode(message,temp_ref:4));
    write_string(message, r_ref_disp, c_ref_disp);
  end
  else begin
    disp_window;              {old windows}
    mark_drift_window;
    return;
  end; {if get_ref}

  if (get_drift_window) then begin
    eval(encode(message,temp_dr:2));
    write_string(message, r_dr_wd_disp, c_dr_wd_disp);
  end
  else begin
    disp_window;
    mark_drift_window;
    return;
  end; {if get_drift_window}
```

```
    if (get_finesse_window) then begin
      eval(encode(message,temp_fi:2));
      write_string(message, r_fi_wd_disp, c_fi_wd_disp);
    end
    else begin
      disp_window;
      mark_drift_window;
      return;
    end; {if get finesse window}

    ref_channel := temp_ref;
    dr_window_width := temp_dr;
    fi_window_width := temp_fi;
    mark_drift_window;
    white_line;
  end; {set_window}
```

A.4 step.pas

```
{set drift control step, finesse test and corection steps.  July 1990, ST.}

  procedure set_step;
    var
      temp_dr_step,
      temp_fi_test,
      temp_fi_corr      : integer;
      num_string        : lstring(20);

    function get_drift_step : boolean;
      begin
        get_drift_step := false;
        prompt('Enter drift control voltage step: (1..40)');
        read_string(num_string,upper(num_string));
        if (decode(num_string,temp_dr_step)) then begin
          if ((temp_dr_step > 0) and (temp_dr_step <= 40)) then begin
            get_drift_step := true;
          end
          else begin
            prompt('Value not in range -- aborting set step');
          end;
        end
        else begin
          prompt('Error in the input, aborting set step');
```

```
    end;
  end; {set_drift_step}


function get_finesse_test : boolean;
  begin
    get_finesse_test := false;
    prompt('Enter finesse test step (1..40)');
    read_string(num_string,upper(num_string));
    if (decode(num_string,temp_fi_test)) then begin
      if ((temp_fi_test > 0) and (temp_fi_test <= 40)) then begin
        get_finesse_test := true;
      end
      else begin
        prompt('Value not in range -- aborting set step');
      end;
    end
    else begin
      prompt('Error in the input, aborting set step');
    end;
  end; {get_finesse_test}

function get_finesse_corr : boolean;
  begin
    get_finesse_corr := false;
    prompt('Enter finesse correction step (1..40)');
    read_string(num_string,upper(num_string));
    if (decode(num_string,temp_fi_corr)) then begin
      if ((temp_fi_corr > 0) and (temp_fi_corr <= 40)) then begin
        get_finesse_corr := true;
      end
      else begin
        prompt('Value not in range -- aborting set step');
      end;
    end
    else begin
      prompt('Error in the input, aborting set step');
    end;
  end; {get_finesse_corr}

begin {set step}
  drift_control := false;
  finesse_control := false;
  sweep_count := 0;
  disp_drift_control;
```

```
    disp_finesse_control;
    disp_step;

    if (get_drift_step) then begin
      eval(encode(message,temp_dr_step:2));
      write_string(message, r_dr_step_disp, c_dr_step_disp);
    end
    else begin
      disp_step;
      return;
    end;

    if (get_finesse_test) then begin
      eval(encode(message,temp_fi_test:2));
      write_string(message, r_Fi_test_disp, c_Fi_test_disp);
    end
    else begin
      disp_step;
      return;
    end;

    if (get_finesse_corr) then begin
      eval(encode(message,temp_fi_corr:2));
      write_string(message, r_Fi_corr_disp, c_Fi_corr_disp);
    end
    else begin
      disp_step;
      return;
    end;

    drift_corr_step   := temp_dr_step;
    finesse_test_step := temp_fi_test;
    finesse_corr_step := temp_fi_corr;
    x_test_data := finesse_test_step;
    y_test_data := finesse_test_step;
    x_corr_data := finesse_corr_step;
    y_corr_data := finesse_corr_step;
    white_line;
  end; {set step}
```

A.5 volt.pas

```
procedure set_drift_volt;
```

```
    var
      temp_data          : integer;
      num_string         : lstring(20);

    begin {set_drift_volt}

      prompt('Enter drift control DAC data (100 ... 4000) : ');
      read_string(num_string,upper(num_string));
      if (decode(num_string,temp_data)) then begin
        if ((temp_data >= 100) and (temp_data <= 4000)) then begin
          dac_data_drift := temp_data;
          out (Card0_DAC_ctrl_reg_Hi, DAC_channel_0);
          out (Card0_DAC_data_reg_Lo, lobyte(dac_data_drift));
          out (Card0_DAC_data_reg_Hi, hibyte(dac_data_drift));
          white_line;
        end {(temp_data >= 100) and (temp_data <= 4000)}
        else begin
          prompt('Value not in range -- aborting set Drift voltage');
        end;
      end {if decode}
      else begin
        prompt('Error in the input, aborting set drift control voltage');
      end;
    end; {set_drift_volt}

{--------------------------------}
  procedure set_tilt_A_volt;

    var
      temp_data          : integer;
      num_string         : lstring(20);

    begin {set_tilt_A_volt}

      prompt('Enter finesse tilt_A voltage (100 ... 4000) : ');
      read_string(num_string,upper(num_string));
      if (decode(num_string,temp_data)) then begin
        if ((temp_data >= 100) and (temp_data <= 4000)) then begin
          dac_data_tilt_A := temp_data;
          out (Card1_DAC_ctrl_reg_Hi, DAC_channel_0);
          out (Card1_DAC_data_reg_Lo, lobyte(dac_data_tilt_A));
          out (Card1_DAC_data_reg_Hi, hibyte(dac_data_tilt_A));
          white_line;
        end
        else begin
```

```
          prompt('Value not in range -- aborting set Tilt_A voltage');
        end;
      end {if decode}
      else begin
        prompt('Error in the input, aborting set tilt_A voltage');
      end;
    end; {set_tilt_A_volt}

{-------------------------------}
  procedure set_tilt_B_volt;

    var
      temp_data          : integer;
      num_string         : lstring(20);

    begin {set_tilt_B_volt}

      prompt('Enter finesse tilt_B voltage (100 ... 4000) : ');
      read_string(num_string,upper(num_string));
      if (decode(num_string,temp_data)) then begin
        if ((temp_data >= 100) and (temp_data <= 4000)) then begin
          dac_data_tilt_B := temp_data;
          out (Card0_DAC_ctrl_reg_Hi, DAC_channel_1);
          out (Card0_DAC_data_reg_Lo, lobyte(dac_data_tilt_B));
          out (Card0_DAC_data_reg_Hi, hibyte(dac_data_tilt_B));
          white_line;
        end
        else begin
          prompt('Value not in range -- aborting set Tilt_B voltage');
        end;
      end {if decode}
      else begin
        prompt('Error in the input, aborting set tilt_B voltage');
      end;
    end; {set_tilt_B_volt}

{-------------------------------}
  procedure set_tilt_C_volt;

    var
      temp_data          : integer;
      num_string         : lstring(20);

    begin {set_tilt_C_volt}
```

```
      prompt('Enter finesse tilt_C voltage (100 ... 4000) : ');
      read_string(num_string,upper(num_string));
      if (decode(num_string,temp_data)) then begin
        if ((temp_data >= 100) and (temp_data <= 4000)) then begin
          dac_data_tilt_C := temp_data;
          out (Card1_DAC_ctrl_reg_Hi, DAC_channel_1);
          out (Card1_DAC_data_reg_Lo, lobyte(dac_data_tilt_C));
          out (Card1_DAC_data_reg_Hi, hibyte(dac_data_tilt_C));
          white_line;
        end
        else begin
          prompt('Value not in range -- aborting set Tilt_C voltage');
        end;
      end {if decode}
      else begin
        prompt('Error in the input, aborting set tilt_C voltage');
      end;
    end; {set_tilt_C_volt}

{-------------------------------}
  procedure clear_volt_arrow;
    begin
      write_string(' ', r_arrow_A, c_arrow_A);
      write_string(' ', r_arrow_B, c_arrow_B);
      write_string(' ', r_arrow_C, c_arrow_C);
      write_string(' ', r_arrow_D, c_arrow_D);
    end;

{-------------------------------}
  procedure vert_arrow_A;
    begin
      clear_volt_arrow;
      if (vert_arrow_mode = volt_A) then begin
        vert_arrow_mode := vert_scale;
      end
      else begin
        vert_arrow_mode := volt_A;
        write_string(chr(18),r_arrow_A, c_arrow_A);  {up/dn arrow}
      end;
    end; {vert_arrow_A}

{-------------------------------}
  procedure vert_arrow_B;
    begin
      clear_volt_arrow;
```

```
      if (vert_arrow_mode = volt_B) then begin
        vert_arrow_mode := vert_scale;
      end
      else begin
        vert_arrow_mode := volt_B;
        write_string(chr(18),r_arrow_B, c_arrow_B);
      end;
    end; {vert_arrow_B}

{-------------------------------}
  procedure vert_arrow_C;
    begin
      clear_volt_arrow;
      if (vert_arrow_mode = volt_C) then begin
        vert_arrow_mode := vert_scale;
      end
      else begin
        vert_arrow_mode := volt_C;
        write_string(chr(18),r_arrow_C, c_arrow_C);
      end;
    end; {vert_arrow_C}

{-------------------------------}
  procedure vert_arrow_D;
    begin
      clear_volt_arrow;
      if (vert_arrow_mode = volt_D) then begin
        vert_arrow_mode := vert_scale;
      end
      else begin
        vert_arrow_mode := volt_D;
        write_string(chr(18),r_arrow_D, c_arrow_D);
      end;
    end; {vert_arrow_D}

{-------------------------------}
  procedure drift_volt_up;
    var
      temp_data : integer;

    begin
      temp_data := dac_data_drift + drift_corr_step;
      if (temp_data <= 4000) then begin
        dac_data_drift := temp_data;
        out (Card0_DAC_ctrl_reg_Hi, DAC_channel_0);
```

```
        out (Card0_DAC_data_reg_Lo, lobyte(dac_data_drift));
        out (Card0_DAC_data_reg_Hi, hibyte(dac_data_drift));
      end
      else begin
        prompt('drift volt out of range ! ');
        write(chr(7));
      end;
    end;

{--------------------------------}
  procedure drift_volt_down;
    var
      temp_data : integer;

    begin
      temp_data := dac_data_drift - drift_corr_step;
      if  (temp_data >= 100 ) then begin
        dac_data_drift := temp_data;
        out (Card0_DAC_ctrl_reg_Hi, DAC_channel_0);
        out (Card0_DAC_data_reg_Lo, lobyte(dac_data_drift));
        out (Card0_DAC_data_reg_Hi, hibyte(dac_data_drift));
      end
      else begin
        prompt('drift volt out of range ! ');
        write(chr(7));
      end;
    end;

{--------------------------------}
  procedure tilt_A_volt_up;
    var
      temp_data : integer;

    begin
      temp_data := dac_data_tilt_A + finesse_corr_step;
      if (temp_data <= 4000) then begin
        dac_data_tilt_A := temp_data;
        out (Card1_DAC_ctrl_reg_Hi, DAC_channel_0);
        out (Card1_DAC_data_reg_Lo, lobyte(dac_data_tilt_A));
        out (Card1_DAC_data_reg_Hi, hibyte(dac_data_tilt_A));
      end
      else begin
        prompt('tilt A volt out of range ! ');
        write(chr(7));
      end;
```

```
    end;

{--------------------------------}
  procedure tilt_A_volt_down;
    var
      temp_data : integer;

    begin
      temp_data := dac_data_tilt_A - finesse_corr_step;
      if (temp_data >= 100) then begin
        dac_data_tilt_A := temp_data;
        out (Card1_DAC_ctrl_reg_Hi, DAC_channel_0);
        out (Card1_DAC_data_reg_Lo, lobyte(dac_data_tilt_A));
        out (Card1_DAC_data_reg_Hi, hibyte(dac_data_tilt_A));
      end
      else begin
        prompt('tilt A volt out of range ! ');
        write(chr(7));
      end;
    end;

{--------------------------------}
  procedure tilt_B_volt_up;
    var
      temp_data : integer;

    begin
      temp_data := dac_data_tilt_B + finesse_corr_step;
      if (temp_data <= 4000) then begin
        dac_data_tilt_B := temp_data;
        out (Card0_DAC_ctrl_reg_Hi, DAC_channel_1);
        out (Card0_DAC_data_reg_Lo, lobyte(dac_data_tilt_B));
        out (Card0_DAC_data_reg_Hi, hibyte(dac_data_tilt_B));
      end
      else begin
        prompt('tilt B volt out of range ! ');
        write(chr(7));
      end;
    end;

{--------------------------------}
  procedure tilt_B_volt_down;
    var
      temp_data : integer;
```

```
    begin
      temp_data := dac_data_tilt_B - finesse_corr_step;
      if (temp_data >= 100) then begin
        dac_data_tilt_B := temp_data;
        out (Card0_DAC_ctrl_reg_Hi, DAC_channel_1);
        out (Card0_DAC_data_reg_Lo, lobyte(dac_data_tilt_B));
        out (Card0_DAC_data_reg_Hi, hibyte(dac_data_tilt_B));
      end
      else begin
        prompt('tilt B volt out of range ! ');
        write(chr(7));
      end;
    end;

{-------------------------------}
  procedure tilt_C_volt_up;
    var
      temp_data : integer;

    begin
      temp_data := dac_data_tilt_C + finesse_corr_step;
      if (temp_data <= 4000) then begin
        dac_data_tilt_C := temp_data;
        out (Card1_DAC_ctrl_reg_Hi, DAC_channel_1);
        out (Card1_DAC_data_reg_Lo, lobyte(dac_data_tilt_C));
        out (Card1_DAC_data_reg_Hi, hibyte(dac_data_tilt_C));
      end
      else begin
        prompt('tilt C volt out of range ! ');
        write(chr(7));
      end;
    end;

{-------------------------------}
  procedure tilt_C_volt_down;
    var
      temp_data : integer;

    begin
      temp_data := dac_data_tilt_C - finesse_corr_step;
      if (temp_data >= 100) then begin
        dac_data_tilt_C := temp_data;
        out (Card1_DAC_ctrl_reg_Hi, DAC_channel_1);
        out (Card1_DAC_data_reg_Lo, lobyte(dac_data_tilt_C));
        out (Card1_DAC_data_reg_Hi, hibyte(dac_data_tilt_C));
```

```
      end
      else begin
        prompt('tilt C volt out of range ! ');
        write(chr(7));
      end;
    end;
```

A.6 get_cmd2.pas

```
{---if in local mode get the next command---}
{---uses IBM bios definitions of the keyboard scan codes---}

    function get_key_cmd(var cmd : cmd_type) : boolean;

      var
        shift : boolean;
        scan_code,key : byte;
        num_string : lstring(20);
        temp_int : integer;
        temp_long : integer4;
        temp_real : real;

      begin
        get_key_cmd := true;
        {always returns true and updates display if nothing else}

        if (not get_key(shift,scan_code,key)) then begin
          cmd.cmd_name := cmd_update_display;
          return;
        end; {if not get key}

{a key has been pressed, identify the key and get a cmd name for do_cmd}
        if (not shift) then begin
          white_line;
          case scan_code of

            f1 : cmd.cmd_name := cmd_start;
            f2 : cmd.cmd_name := cmd_stop;
            f3 : cmd.cmd_name := cmd_clear;
            f4 : cmd.cmd_name := cmd_auto_collect;
            f5 : cmd.cmd_name := cmd_set_drift_control;
            f6 : cmd.cmd_name := cmd_set_finesse_control;
            f7 : cmd.cmd_name := cmd_vert_arrow_D;
            f8 : cmd.cmd_name := cmd_vert_arrow_B;
```

```
    f9 : cmd.cmd_name := cmd_vert_arrow_A;
    f10: cmd.cmd_name := cmd_vert_arrow_C;

    a_f1 : cmd.cmd_name := cmd_disp_fast;
    a_f2 : cmd.cmd_name := cmd_roi_mode;
    a_f3 : cmd.cmd_name := cmd_MCB_buffer;
    a_f4 : cmd.cmd_name := cmd_full_expand;
    a_f5 : cmd.cmd_name := cmd_disp_info;
{   a_f6 :                                      }
    a_f7 : cmd.cmd_name := cmd_set_drift_volt;
    a_f8 : cmd.cmd_name := cmd_set_tilt_B_volt;
    a_f9 : cmd.cmd_name := cmd_set_tilt_A_volt;
    a_f10. cmd.cmd_name := cmd_set_tilt_C_volt;

    c_f1 : begin
      cmd.cmd_name := cmd_set_mcb;
      cmd.int_num := 1;
    end;

    c_f2 : begin
      cmd.cmd_name := cmd_set_mcb;
      cmd.int_num := 2;
    end;

    c_f3 : begin
      cmd.cmd_name := cmd_set_mcb;
      cmd.int_num := 3;
    end;

    c_f4 : begin
      cmd.cmd_name := cmd_set_mcb;
      cmd.int_num := 4;
    end;

    c_f5 : begin
      cmd.cmd_name := cmd_set_mcb;
      cmd.int_num := 5;
    end;

    c_f6 : begin
      cmd.cmd_name := cmd_set_mcb;
      cmd.int_num := 6;
    end;

    c_f7 : begin
```

```
  cmd.cmd_name := cmd_set_mcb;
  cmd.int_num := 7;
end;

c_f8 : begin
  cmd.cmd_name := cmd_set_mcb;
  cmd.int_num := 8;
end;

up : begin
  case (vert_arrow_mode) of
    vert_scale : cmd.cmd_name := cmd_vert_up;
    volt_A :     cmd.cmd_name := cmd_tilt_A_volt_up;
    volt_B :     cmd.cmd_name := cmd_tilt_B_volt_up;
    volt_C :     cmd.cmd_name := cmd_tilt_C_volt_up;
    volt_D :     cmd.cmd_name := cmd_drift_volt_up;
    otherwise    cmd.cmd_name := cmd_update_display;
  end;
end;

down : begin
  case (vert_arrow_mode) of
    vert_scale : cmd.cmd_name := cmd_vert_down;
    volt_A :     cmd.cmd_name := cmd_tilt_A_volt_down;
    volt_B :     cmd.cmd_name := cmd_tilt_B_volt_down;
    volt_C :     cmd.cmd_name := cmd_tilt_C_volt_down;
    volt_D :     cmd.cmd_name := cmd_drift_volt_down;
    otherwise    cmd.cmd_name := cmd_update_display;
  end;
end;

left      : cmd.cmd_name := cmd_marker_left;
right     : cmd.cmd_name := cmd_marker_right;
pgdn      : cmd.cmd_name := cmd_marker_fast_left;
pgup      : cmd.cmd_name := cmd_marker_fast_right;
home      : cmd.cmd_name := cmd_marker_home;
end_key   : cmd.cmd_name := cmd_marker_end;
back_space: cmd.cmd_name := cmd_marker_ref;

equal : begin
  prompt('Enter marker channel (0 .. 4095):');
  read_string(num_string,upper(num_string));
  if(decode(num_string,temp_int)) then begin
    cmd.cmd_name := cmd_set_marker_channel;
    cmd.int_num := temp_int;
```

```
        end{if}
        else begin
prompt('Error in the input, aborting set marker channel');
          cmd.cmd_name := cmd_update_display;
        end;{else}
      end;

      minus : cmd.cmd_name := cmd_display_compress;
      plus : cmd.cmd_name := cmd_display_expand;

      ins : cmd.cmd_name := cmd_insert_roi;
      del : cmd.cmd_name := cmd_delete_roi;

      space_bar, return_code : begin
        white_line;
        cmd.cmd_name := cmd_update_display;
      end;

      a_1, key_1 : begin
        case menu of
          main: begin
            cmd.cmd_name := cmd_menu_presets;
          end;{main}

          help: begin
            cmd.cmd_name := cmd_help_info;
          end;{help}

          presets: begin
            cmd.cmd_name := cmd_set_window;
          end;{presets}

          ramp: begin
            if (sys[cur_913].external_dwell) then begin
              cmd.cmd_name := cmd_set_dwell_time;
            end
            else begin {internal dwell clock}
    prompt('Enter internal clock dwell time in microseconds: ');
              read_string(num_string,upper(num_string));
              if(decode(num_string,temp_real)) then begin
                if (temp_real < 65536) then begin
                  cmd.cmd_name := cmd_preset_dwell_time;
                  cmd.long_num := round4(temp_real);
                end
                else begin
```

```
            prompt('Error in the input, aborting PRESET');
            cmd.cmd_name := cmd_update_display;
          end;{else}
        end
        else begin
          prompt('Error in the input, aborting PRESET');
          cmd.cmd_name := cmd_update_display;
        end;
      end; {internal dwell clock}
    end;{ramp}

    calc: begin
      cmd.cmd_name := cmd_calib;
    end;{calc}

    i_o: begin
      cmd.cmd_name := cmd_save;
      cmd.strng := null;
      get_name(cmd.strng,'mcs');
    end;{i_o}

    utils: begin
      prompt('MCB -> Buffer, (Y/N) ? ');
      if (confirm) then begin
        cmd.cmd_name := cmd_fill_buffer;
      end;
    end;{utils}

    otherwise begin
      prompt('That key is not defined');
      cmd.cmd_name := cmd_update_display;
    end;
  end;{case menu of}
end; {case a_1}

a_2, key_2 : begin
  case menu of
    main: begin
      cmd.cmd_name := cmd_menu_ramp;
    end;{main}

    help: begin
      cmd.cmd_name := cmd_help_menu;
    end;{help}
```

```
        presets: begin
          cmd.cmd_name := cmd_set_step;
        end;{presets}

        ramp: begin
          if (not sys[cur_913].external_dwell) then begin
prompt('Please reset configuation, select external clock');
            cmd.cmd_name := cmd_update_display;
          end
          else if (not sys[cur_913].external_start) then begin
prompt('Please reset configuation, select external start trigger');
            cmd.cmd_name := cmd_update_display;
          end
          else begin
            cmd.cmd_name := cmd_set_segment_mult;
          end;
        end;

        calc: begin
          cmd.cmd_name := cmd_roi_area;
        end;{calc}

        i_o: begin
          cmd.cmd_name := cmd_recall;
          cmd.strng := null;
          get_name(cmd.strng,'mcs');
        end;{i_o}

        utils: begin
          prompt('Exit to DOS, (Y/N) ? ');
          if (confirm) then begin
            cmd.cmd_name := cmd_dos;
          end
          else begin
            cmd.cmd_name := cmd_update_display;
          end;
        end;{utils}

        otherwise begin
          prompt('That key is not defined');
          cmd.cmd_name := cmd_update_display;
        end;
      end;{case menu of}
    end; {case a_2}
```

```
        a_3, key_3 : begin
          case menu of
            main: begin
              cmd.cmd_name := cmd_menu_calc;
            end;{main}

            help: begin
              cmd.cmd_name := cmd_help_f_key;
            end;{help}

            presets: begin
    prompt('Enter number of channels per sweep (4..4096) : ');
              read_string(num_string,upper(num_string));
              if(decode(num_string,temp_int)) then begin
                cmd.cmd_name := cmd_preset_pass_length;
                cmd.int_num := temp_int;
              end{if}
              else begin
                prompt('Error in the input, aborting PRESET');
                cmd.cmd_name := cmd_update_display;
              end;{else}
            end;

            ramp: begin
              if (not sys[cur_913].external_dwell) then begin
      prompt('Please reset configuation, select external clock');
                cmd.cmd_name := cmd_update_display;
              end
              else if (not sys[cur_913].external_start) then begin
prompt('Please reset configuation, select external start trigger');
                cmd.cmd_name := cmd_update_display;
              end
              else begin
                cmd.cmd_name := cmd_set_segment;
              end;
            end;

            calc: begin
              cmd.cmd_name := cmd_peak_info;
            end;{calc}

            i_o: begin
              cmd.cmd_name := cmd_save_roi;
              cmd.strng := null;
              get_name(cmd.strng,'roi');
```

```
            end;{i_o}

            utils: begin
              cmd.cmd_name := cmd_user;
prompt('Enter an executable command, as though at the dos prompt');
              read_string(cmd.strng,upper(cmd.strng));
            end;{utils}

            otherwise begin
              prompt('That key is not defined');
              cmd.cmd_name := cmd_update_display;
            end;
          end;{case menu of}
        end; {case a_3}

        a_4, key_4 : begin
          case menu of
            main: begin
              cmd.cmd_name := cmd_menu_i_o;
            end;{main}

            help: begin
              cmd.cmd_name := cmd_help_alt_f_key;
            end;{help}

            presets: begin
              prompt('Enter total number of sweeps preset : ');
              read_string(num_string,upper(num_string));
              if(decode(num_string,temp_long)) then begin
                cmd.cmd_name := cmd_preset_pass_count;
                cmd.long_num := temp_long;
              end{if}
              else begin
                prompt('Error in the input, aborting PRESET');
                cmd.cmd_name := cmd_update_display;
              end;{else}
            end;

            ramp: begin
              if (not sys[cur_913].external_dwell) then begin
        prompt('Please reset configuation, select external clock');
                cmd.cmd_name := cmd_update_display;
              end
              else if (not sys[cur_913].external_start) then begin
  prompt('Please reset configuation, select external start trigger');
```

```
        cmd.cmd_name := cmd_update_display;
      end
      else begin
        cmd.cmd_name := cmd_delete_segment;
      end;
    end;

    calc: begin
      prompt('Data divided by the number of sweeps, (Y/N) ? ');
      if (confirm) then begin
        cmd.cmd_name := cmd_normalize;
      end
      else begin
        cmd.cmd_name := cmd_update_display;
      end;
    end;{calc}

    i_o: begin
      cmd.cmd_name := cmd_recall_roi;
      cmd.strng := null;
      get_name(cmd.strng,'roi');
    end;{i_o}

    utils: begin
      get_cmd_array;
      if(have_cmds) then begin
        prompt('Executing external command file');
        cmd_err := NO_ERR;
        cmd_mode := disk_file;
        cmd.cmd_name := cmd_update_display;
      end;{if}
    end;{utils}

    otherwise begin
      prompt('That key is not defined');
      cmd.cmd_name := cmd_update_display;
    end;
  end;{case menu of}
end; {case a_4}

a_5, key_5 : begin
  case menu of
    main: begin
      cmd.cmd_name := cmd_menu_utils;
    end;{main}
```

```
            help: begin
              cmd.cmd_name := cmd_help_keypad;
            end;{help}

            presets: begin
              prompt('Setup system configuration, (Y/N) ? ');
              if (confirm) then begin
                cmd.cmd_name := cmd_setup;
              end
              else begin
                cmd.cmd_name := cmd_update_display;
              end;
            end;{presets}

            ramp: begin
              if (not sys[cur_913].external_dwell) then begin
        prompt('Please reset configuation, select external clock');
                cmd.cmd_name := cmd_update_display;
              end
              else if (not sys[cur_913].external_start) then begin
prompt('Please reset configuation, select external start trigger');
                cmd.cmd_name := cmd_update_display;
              end
              else begin
                cmd.cmd_name := cmd_start_ramp;
              end;
            end;

            calc: begin
              cmd.cmd_name := cmd_smooth;
            end;{calc}

            i_o: begin
                cmd.cmd_name := cmd_print;
            end;{i_o}

            utils: begin
              cmd.cmd_name := cmd_compare;
            end;{utils}

            otherwise begin
              prompt('That key is not defined');
              cmd.cmd_name := cmd_update_display;
            end;
```

```
  end;{case menu of}
end; {case a_5}

a_6, key_6 : begin
  case menu of
    main: begin
      cmd.cmd_name := cmd_menu_help;
    end;{main}

    help: begin
      cmd.cmd_name := cmd_help_cmd_key;
    end;{help}

    calc: begin
      cmd.cmd_name := cmd_total_sum;
    end;{calc}

    utils: begin
      prompt('Buffer -> MCB, (Y/N) ? ');
      if (confirm) then begin
        cmd.cmd_name := cmd_restore;
      end;
    end;{utils}

    otherwise begin
      prompt('That key is not defined');
      cmd.cmd_name := cmd_update_display;
    end;
  end;{case menu of}
end; {case a_6}

a_7, key_7 : begin
  case menu of
    calc: begin
      cmd.cmd_name := cmd_report;
      prompt('Enter report filename (PRN for printer) : ');
      read_string(cmd.strng,upper(cmd.strng));
    end;{calc}

    otherwise begin
      prompt('That key is not defined');
      cmd.cmd_name := cmd_update_display;
    end;
  end;{case menu of}
end; {case a_7}
```

```
a_9, key_9 : begin
  case menu of
    main: begin
      prompt('Quitting program ... (Y/N) ? ');
      if (confirm) then begin
        cmd.cmd_name := cmd_quit;
      end
      else begin
        cmd.cmd_name := cmd_update_display;
      end;
    end;

    otherwise begin
      prompt('That key is not defined');
      cmd.cmd_name := cmd_update_display;
    end;
  end;{case menu of}
end; {case a_9}

a_0, key_0 : begin
  case menu of
    main: begin
      prompt(Copyright);
      reverse_chars(r_copyright,c_copyright,14);
      cmd.cmd_name := cmd_update_display;
    end;

    otherwise begin
      cmd.cmd_name := cmd_menu_main;
    end;
  end;{case menu of}
end; {case a_0}

a_m : cmd.cmd_name := cmd_menu_main;
a_p : cmd.cmd_name := cmd_menu_presets;
a_r : cmd.cmd_name := cmd_menu_ramp;
a_c : cmd.cmd_name := cmd_menu_calc;
a_i : cmd.cmd_name := cmd_menu_i_o;
a_u : cmd.cmd_name := cmd_menu_utils;
a_h : cmd.cmd_name := cmd_menu_help;

otherwise begin
  prompt('That key is not defined');
  cmd.cmd_name := cmd_update_display;
```

```
        end; {otherwise}
      end; {case scan_code of}
    end {if not shift key}
    else begin {shift key}
      white_line;
      case scan_code of
        left  : cmd.cmd_name := cmd_index_left;  {shift left arrow}
        right : cmd.cmd_name := cmd_index_right; {shift right arrow}
        otherwise begin
          prompt('That key is not defined');
          cmd.cmd_name := cmd_update_display;
        end; {otherwise}
      end; {case scan_code}
    end; {else shift key}

  end; {function get_key_cmd}
```

A.7 do_cmd.pas

```
{----- Perform the command passed by one of the cmd getter routines -----}
{ added new commands for drift control, clock, and help.  July 1990, ST. }

{
 Input error checking is done by get_cmd. Error checking that depends on
 the state of the mca machine, such as commands legal only when in the
 calc mode  or when a specific mca must be selected, is done by do_cmd.
 Errors in implementing the command - such as an illegal file name or out
 of  range numeric input - are reported by the specific procedure called
 to  perform the command. The global variable cmd_err (integer) will be
 set to  give some indication of the error that has occurred :

 cmd_err = NO_ERR = 0;
           0  Successful execution

           MODE_ERR = 1
           1  Aborted by procedure do_cmd, illegal in the current mode

           PARM_ERR = 2
           2  Aborted by a called procedure, usually a parameter error

           EXEC_ERR = 3
           3  Aborted by a called procedure, an execution or machine error
              such as disk full, communication with Adcam error, etc.
```

```
          4 - 9 Reserved

          ADCAM_ERR
          10 Adcam refuses the command - example : trying to start an
             active segment

 Get_cmd should always check cmd_err when in disk_file or remote modes
 and  halt with error message.
}

  procedure do_cmd(cmd : cmd_type);

    begin

      case cmd.cmd_name of

        cmd_update_display      : update_display;
        cmd_setup               : f1_setup_display;
        cmd_roi_mode            : f2_roi;
        cmd_full_expand         : f3_full_expand;
        cmd_MCB_buffer          : f4_adcam_buffer;
        cmd_vert_down           : f5_vert_down;
        cmd_vert_up             : f6_vert_up;
        cmd_display_compress    : f7_compress;
        cmd_display_expand      : f8_expand;
        cmd_marker_left         : f9_left;
        cmd_marker_right        : f10_right;
        cmd_marker_fast_left    : f9_fast_left;
        cmd_marker_fast_right   : f10_fast_right;
        cmd_marker_home         : marker_home;
        cmd_marker_end          : marker_end;
        cmd_set_marker_channel  : set_marker_channel(cmd.int_num);
        cmd_marker_ref          : marker_ref;

        cmd_set_mcb : begin
          cmd_err := NO_ERR;
          if(cmd.int_num = 0) then begin {display the buffer}
            if(plot_913) then begin
              f4_adcam_buffer;
            end;{if}
          end{else if}
          else begin
            if(not plot_913) then begin
              f4_adcam_buffer;
            end;
```

```
        switch_913(cmd.int_num);
      end;{else}
    end;

    cmd_index_left : begin
      cmd_err := NO_ERR;
      index_left;
    end;

    cmd_index_right : begin
      cmd_err := NO_ERR;
      index_right;
    end;

    cmd_insert_roi : begin
      cmd_err := NO_ERR;
      insert_roi;
    end;

    cmd_delete_roi : begin
      cmd_err := NO_ERR;
      delete_roi;
    end;

    cmd_start : begin
      cmd_err := NO_ERR;
      if(not plot_913) then begin
prompt('START applies to MCS only, please select the desired MCB');
        cmd_err := MODE_ERR;
      end{if}
      else begin
        start;
      end;{else}
    end;

    cmd_wait : begin
      cmd_err := NO_ERR;
      if(not plot_913) then begin
prompt('WAIT applies to MCS only, please select the desired MCB');
        cmd_err := MODE_ERR;
      end{if}
      else begin
        waiting := true;
      end;{else}
    end;
```

```
      cmd_stop : begin
        cmd_err := NO_ERR;
        if(not plot_913) then begin
prompt('STOP applies to MCS only, please select the desired MCB');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          stop;
        end;{else}
      end;

      cmd_clear : begin
        cmd_err := NO_ERR;
        if(not plot_913) then begin
prompt('ERASE applies to MCS only, please select the desired MCB');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          clear;
        end;{else}
      end;

      cmd_menu_main : begin
        menu := main;
        disp_menu;
      end;

      cmd_menu_help : begin
        menu := help;
        disp_menu;
      end;

      cmd_menu_presets : begin
        if(not plot_913) then begin
          f4_adcam_buffer;
prompt('PRESET applies to MCB only, switching you to MCB');
        end;
        menu := presets;
        disp_menu;
      end;

      cmd_menu_ramp : begin
        if(not plot_913) then begin
          f4_adcam_buffer;
```

```
prompt('RAMP applies to MCB only, switching you to MCB');
        end;
        menu := ramp;
        disp_menu;
        disp_segment;
      end;

      cmd_menu_calc : begin
        if(plot_913) then begin
          f4_adcam_buffer;
prompt('CALC applies to Buffer only, switching you to Buffer');
        end;{if}
        menu := calc;
        disp_menu;
      end;

      cmd_menu_i_o : begin
        if(plot_913) then begin
          f4_adcam_buffer;
prompt('I/O applies to Buffer only, switching you to Buffer');
        end;{if}
        menu := i_o;
        disp_menu;
      end;

      cmd_menu_utils : begin
        menu := utils;
        disp_menu;
      end;

      cmd_preset_dwell_time : begin
        cmd_err := NO_ERR;
        if(not plot_913) then begin
prompt('PRESET applies to MCS only, please select the desired MCB');
          cmd_err := MODE_ERR;
        end{if}
        else if (sys[cur_913].external_dwell) then begin
          prompt('Can''t set dwell in external dwell clock mode');
          cmd_err := MODE_ERR;
        end{else if}
        else begin
          preset_dwell_time(cmd.long_num);
          disp_presets;
        end;{else}
      end;
```

```
      cmd_preset_pass_length : begin
        cmd_err := NO_ERR;
        if(not plot_913) then begin
prompt('PRESET applies to MCS only, please select the desired MCB');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          preset_pass_length(cmd.int_num);
          disp_presets;
        end;{else}
      end;

      cmd_preset_pass_count : begin
        cmd_err := NO_ERR;
        if(not plot_913) then begin
prompt('PRESET applies to MCS only, please select the desired MCB');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          preset_pass_count(cmd.long_num);
          disp_presets;
        end;{else}
      end;

      cmd_calib : begin
        cmd_err := NO_ERR;
        if(plot_913) then begin
prompt('Calc functions apply to Buffer only, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          calibrate;
        end;{else}
      end;

      cmd_roi_area : begin
        cmd_err := NO_ERR;
        if(plot_913) then begin
prompt('Calc functions apply to Buffer only, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          roi_sum;
        end;{else}
```

```
      end;

      cmd_peak_info : begin
        cmd_err := NO_ERR;
        if(plot_913) then begin
prompt('Calc functions apply to Buffer only, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          peak_info;
        end;{else}
      end;

      cmd_normalize : begin
        cmd_err := NO_ERR;
        if(plot_913) then begin
prompt('Calc functions apply to Buffer only, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          normalize;
        end;{else}
      end;

      cmd_smooth : begin
        cmd_err := NO_ERR;
        if(plot_913) then begin
prompt('Calc functions apply to Buffer only, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          smooth;
        end;{else}
      end;

      cmd_total_sum : begin
        cmd_err := NO_ERR;
        if(plot_913) then begin
prompt('Calc functions apply to Buffer only, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          total_sum;
        end;{else}
      end;
```

```
      cmd_report : begin
        cmd_err := NO_ERR;
        if(plot_913) then begin
prompt('Calc functions apply to Buffer only, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          report(cmd.strng);
        end;{else}
      end;

      cmd_save : begin
        cmd_err := NO_ERR;
        if (plot_913) then begin
prompt('SAVE works only from the Buffer, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          save_spectrum(cmd.strng);
        end;{else}
      end;

      cmd_recall : begin
        cmd_err := NO_ERR;
        if (plot_913) then begin
prompt('RECALL works only to the Buffer, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          recall_spectrum(cmd.strng);
        end;{else}
      end;

      cmd_save_roi : begin
        cmd_err := NO_ERR;
        save_roi(cmd.strng);
      end;

      cmd_recall_roi : begin
        cmd_err := NO_ERR;
        recall_roi(cmd.strng);
      end;

      cmd_recall_calib : begin
```

```
        cmd_err := NO_ERR;
        if (plot_913) then begin
prompt('CALIBRATE works only to the Buffer, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          recall_calibration(cmd.strng);
        end;{else}
      end;

      cmd_describe_detector : begin
        cmd_err := NO_ERR;
        if (plot_913) then begin
prompt('Can only describe the Buffer, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          buf_det_desc := cmd.strng;
        end;{else}
      end;

      cmd_describe_sample : begin
        cmd_err := NO_ERR;
        if (plot_913) then begin
prompt('Can only describe the Buffer, please select the Buffer');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          buf_smp_desc := cmd.strng;
        end;{else}
      end;

      cmd_fill_buffer : begin
        cmd_err := NO_ERR;
        if(not plot_913) then begin
          prompt('Please select the desired MCB');
          cmd_err := MODE_ERR;
        end{if}
        else begin
          fill_buffer;
        end;{else}
      end;

      cmd_restore : begin
        cmd_err := NO_ERR;
```

```
          if(not plot_913) then begin
            prompt('Please select the desired MCB');
            cmd_err := MODE_ERR;
          end{if}
          else begin
            restore;
            disp_presets;
          end;{else}
        end;

        cmd_print : begin
          cmd_err := NO_ERR;
          if(plot_913) then begin
  prompt('PRINT works only from the Buffer, select the Buffer.');
            cmd_err := MODE_ERR;
          end{if}
          else begin
            type_dat;
          end;{else}
        end;

        cmd_dos : begin
          cmd_err := NO_ERR;
          dos;
        end;

        cmd_user : begin
          cmd_err := NO_ERR;
          spawn_child(cmd.strng);
        end;

        cmd_extern : begin
          cmd_err := NO_ERR;
        end;

        cmd_compare : begin
          cmd_err := NO_ERR;
          if(plot_913 or (not page)) then begin
prompt('Compare only functions on the Buffer, in expanded view');
            cmd_err := MODE_ERR;
          end{if}
          else begin
            compare;
          end;{else}
        end;
```

```
cmd_quit : begin
  done := true;
end;

cmd_disp_window          : disp_window;
cmd_disp_segment         : disp_segment;
cmd_disp_step            : disp_step;
cmd_disp_info            : disp_info;
cmd_disp_fast            : disp_fast;

cmd_set_window           : set_window;
cmd_set_step             : set_step;
cmd_set_drift_control    : set_drift_control;
cmd_set_finesse_control  : set_finesse_control;
cmd_auto_collect         : auto_collect;

cmd_set_dwell_time       : set_dwell_time;
cmd_set_segment_mult     : set_segment_mult;
cmd_set_segment          : set_segment;
cmd_delete_segment       : delete_segment;
cmd_start_ramp           : start_ramp;

cmd_set_drift_volt       : set_drift_volt;
cmd_set_tilt_A_volt      : set_tilt_A_volt;
cmd_set_tilt_B_volt      : set_tilt_B_volt;
cmd_set_tilt_C_volt      : set_tilt_C_volt;
cmd_vert_arrow_A         : vert_arrow_A;
cmd_vert_arrow_B         : vert_arrow_B;
cmd_vert_arrow_C         : vert_arrow_C;
cmd_vert_arrow_D         : vert_arrow_D;
cmd_drift_volt_up        : drift_volt_up;
cmd_tilt_A_volt_up       : tilt_A_volt_up;
cmd_tilt_B_volt_up       : tilt_B_volt_up;
cmd_tilt_C_volt_up       : tilt_C_volt_up;
cmd_drift_volt_down      : drift_volt_down;
cmd_tilt_A_volt_down     : tilt_A_volt_down;
cmd_tilt_B_volt_down     : tilt_B_volt_down;
cmd_tilt_C_volt_down     : tilt_C_volt_down;

cmd_help_info            : help_info;
cmd_help_menu            : help_menu;
cmd_help_f_key           : help_f_key;
cmd_help_alt_f_key       : help_alt_f_key;
cmd_help_keypad          : help_keypad;
```

```
      cmd_help_cmd_key        : help_cmd_key;

      otherwise prompt('Invalid command');

    end; {case}
  end; {procedure do_cmd}
```

A.8 main.pas

```
{---------------------- main -------------------------}
  begin {main}

    old_video_mode := get_video_mode;
    initial;
    install_isr;
    disp_menu;
    prompt('EG&G Ortec  ACE-MCS Software  Release 1.3');

    repeat {forever}
      if (waiting) then begin
        if (not sys[cur_913].active) then begin
          waiting := false; {interrupt service routine set active = 0}
          update_display;
        end{if not active}
        else if (get_key(shift,scan_code,key)) then begin
          if (key = esc) then begin
            prompt('User ended wait command.');
            waiting := false;
          end{if (key = esc)}
          else if (key = 0) then begin
            case scan_code of
              f3 : f3_full_expand;
              f5 : f5_vert_down;
              f6 : f6_vert_up;
              f7 : f7_compress;
              f8 : f8_expand;
              f9 : f9_left;
              f10 : f9_right;
              a_f9 : f9_fast_left;
              a_f10 : f9_fast_right;
              home : marker_home;
              end_key : marker_end;
              otherwise prompt('Key not defined while WAITING');
            end; {case}
```

```
        end; {else if (key = 0)}
      end{else if get_key}
      else begin
        update_display;
      end; {else}
    end{if waiting}
    else if (get_cmd(cmd)) then begin
      do_cmd(cmd);
    end{if get_cmd}
    else begin
      {do nothing just a stub}
    end; {else}
  until done;

  if (not put_config) then begin
prompt('Error writing data -- is disk full?...<RETURN> to continue');
    pause; {waiting for enter key to be pressed}
  end; {if not put_config}
  video_mode(olde_video_mode);
  restore_isr;
  err_exit(errorlevel);

end. {main}
```

Appendix B

Interrupt Service Routine

```
;-------------------------------------------------------------------
                extrn   num_913s :word  ;number of installed 913's
                extrn   cur_913  :word  ;the currently addressed 913
                extrn   sys      :byte  ;array[0..max_913s] of data strucs

;               ---------------------------------------
                extrn   switch_flag             :byte   ;pascal boolean
                extrn   drift_control           :byte
                extrn   finesse_control         :byte
                extrn   update_drift            :byte
                extrn   update_finesse          :byte

                extrn   sweep_count             :word   ;pascal integer

                extrn   ref_channel             :word
                extrn   dr_window_width         :word
                extrn   fi_window_width         :word

                extrn   drift_corr_step         :word

                extrn   x_test_data             :word
                extrn   x_corr_data             :word
                extrn   y_test_data             :word
                extrn   y_corr_data             :word

                extrn   dac_data_drift          :word   ;card0 dac0
                extrn   dac_data_tilt_B         :word   ;card0 dac1
                extrn   dac_data_tilt_A         :word   ;card1 dac0
                extrn   dac_data_tilt_C         :word   ;card1 dac1
```

```
                extrn   dr_window_counts         :word   ;for pascal display
                extrn   fi_window_counts         :word
                extrn   dr_window_differ         :word
                extrn   fi_window_differ         :word

                extrn   err_code                 :word

;-----------------------------------------------------------------
dgroup          group _data
_data           segment word public     'DATA'
_data           ends

                assume  cs:support,ds:dgroup

support         segment para public 'code'
                page 40,132

;-------------- 1990/3/4(ST.)
drift_corr_done db      0               ;if drift correction have been done,
                                        ; clear accumulated drift counts
left_lo         dw      0               ;left window counts
left_hi         dw      0
right_lo        dw      0               ;right window counts
right_hi        dw      0
center1_lo      dw      0               ;finesse window counts before test
center1_hi      dw      0
center2_lo      dw      0               ; after test
center2_hi      dw      0

old_left_lo     dw      0               ;used in collect mode
old_left_hi     dw      0
old_right_lo    dw      0
old_right_hi    dw      0
old_center1_lo  dw      0
old_center1_hi  dw      0
old_center2_lo  dw      0
old_center2_hi  dw      0

;---------------------------
data_seg        dw      ?               ;store pascal dgroup segment
old_int_off     dw      ?               ;save area to restore the int vector
old_int_seg     dw      ?
old_int_mask    db      ?               ;save area for old int mask

active_offs     equ     0               ;offset of active flag
```

```
cbyte_offs      equ     1               ;offset of cbyte within structure
sum_offs        equ     5               ;operating mode: sum or oscilloscope
length_offs     equ     10              ;offset of pass length
pass_offs       equ     12              ;offset of pass count
pass_pre_offs   equ     16              ;offset of pass count preset
abort_offs      equ     60              ;offset of abort flag

;define constant ------------------------------------------
err_drift               equ     1    ;card0_dac0 data overflow
err_tilt_A              equ     2    ;card1_dac0
err_tilt_B              equ     3    ;card0_dac1
err_tilt_C              equ     4    ;card1_dac1
err_left                equ     5    ;net left window counts overflow
err_right               equ     6    ;    right
err_center1             equ     7    ;    ...
err_center2             equ     8    ;    ...
err_sweep               equ     9    ;sweep number out of range

adcam1_seg              equ    0d000h ;address of MCB spectrum memory
adcam1_offset           equ     0000h ; segment:offset = d000:0000

card0_base_addr         equ     02E2h ;AD/DA card_0 base address
card1_base_addr         equ     06E2h ;AD/DA card_1 base address
dev_num_reg_offset      equ    0C000h ;device number register offset
dac_ctrl_reg_offset     equ     1000h ;DAC control register offset
dac_data_reg_offset     equ     3000h ;DAC data register offset

card0_dev_num_reg       equ    0C2E2h
card0_dev_num_reg_Hi    equ    0C2E3h ; not used
card0_dac_ctrl_reg      equ     12E2h ; not used
card0_dac_ctrl_reg_Hi   equ     12E3h
card0_dac_data_reg      equ     32E2h
card0_dac_data_reg_Hi   equ     32E3h

card1_dev_num_reg       equ    0C6E2h
card1_dev_num_reg_Hi    equ    0C6E3h ; not used
card1_dac_ctrl_reg      equ     16E2h ; not used
card1_dac_ctrl_reg_Hi   equ     16E3h
card1_dac_data_reg      equ     36E2h
card1_dac_data_reg_Hi   equ     36E3h

dac_dev_num             equ     9
dac_channel_0           equ     0     ;2 channels on crad_0
dac_channel_1           equ     1
```

```
;to use card_0 DAC channel #0:
; 1. set dac_device_number (9) to card0_dev_num_reg,
;        dev_num_reg_Hi not used
; 2. set dac_channel_0 (0) to card_0_dac_ctrl_reg_Hi,
;        ctrl_reg_Lo not used
; 3. load DAC data double word to card0_dac_data_reg
;
;    000h = 0.000v
;    FFFh = 9.998v       1 LSB = 2.4414mv
;
;step 1 have been done by PASCAL procedure initial

;-----------------------------------------------
;install an interrupt service routine for the 913
                public  install_isr
install_isr     proc    far

                cli                     ;don't bother me
                push    es
                push    ds              ;pascal dgroup segment

;save the old interrupt vector to restore when program exits
                mov     al,0bh          ;irq3 + const offset of 8 = 11
                mov     ah,35h          ;get vector service from dos
                int     21h             ;dos call es:bx has vector
                mov     cs:old_int_off,bx
                mov     cs:old_int_seg,es

                mov     dx,offset isr   ;need address of isr in ds:dx
                push    cs
                pop     ds              ;ds has this code segment
                mov     ah,25h          ;install int vector service
                mov     al,0bh          ;irq3 + const offset of 8 = 11
                int     21h             ;dos call

                pop     ds              ;restore data addressability
                pop     es

                mov     cs:data_seg,ds  ;store dgroup seg for isr

                in      al,21h          ;interrupt controller mask
                mov     cs:old_int_mask,al      ;restore when we leave
                and     al,11110111b    ;clear irq3 mask
                out     21h,al          ;restore mask
```

```
;1989/9/22(ST.) ---------------
                mov     al,20h
                out     20h,al          ;reset 8259 interrupt controller
;               ---------------

                sti                     ;ready for interrupt
                ret
install_isr     endp

;--------------------------------
;restore irq3 to its value as stored by install_isr
                public  restore_isr
restore_isr     proc    far
                push    ds

;need address of isr in ds:dx
                mov     dx,cs:old_int_off
                mov     ds,cs:old_int_seg
                mov     ah,25h          ;install int vector service
                mov     al,0bh          ;irq3 + const offset of 8 = 11
                int     21h             ;dos call

                mov     al,cs:old_int_mask      ;as we found it
                out     21h,al                  ;restore mask
                pop     ds

                ret
restore_isr     endp


;--------------------
;This is a pascal procedure to terminate the program and return an error
;level. Use as the last statement in the program.
                public  err_exit
err_exit        proc    far
                push    bp                      ;save caller's frame pointer
                mov     bp,sp                   ;setup to get variables
                mov     al,byte ptr [bp+6]      ;get err_level
                mov     ah,4ch                  ;terminate with error level
                int     21h                     ;dos function call
                pop     bp                      ;never get to here
                ret     2                       ; but just in case
err_exit        endp
```

```
;----------------------------------------
;revised procedure marker from cga_901.asm
;to draw or clear a smaller marker line
;don't want to touch cga_901, so put it here
;pascal declaration: procedure marker_small(cursor_line:integer);extern;

                public  marker_small
marker_small    proc    far
                push    bp              ;save the frame pointer
                mov     bp,sp           ;prepare to pick up relative channel
                mov     bx,[bp+6]       ;get the channel offset
                push    es              ;save es segment
                mov     ax,0b800h       ;video segment
                mov     es,ax           ;ready to write to video
                mov     cl,bl           ;save the low order 3 bits
                and     cl,07h          ;clear the top five bits to get count
                shr     bx,1
                shr     bx,1
                shr     bx,1            ;divide by 8 to get the byte number
                mov     al,80h          ;put a one in the msb
                shr     al,cl           ;mov the one to the correct offset
                add     bx,1376         ;skip the function key and top border

;----- was      mov     cx, 66          ;66 points each even and odd rows
;----- skip upper part of the marker line (ST. 1990/5/28)
                add     bx,5120         ;skip upper 64 points (5120=64*80)
                mov     cx,2            ;2 points below sepctrum base line

marker_loop:    xor     es:[bx],al      ;mark or clear a dot in the column
                xor     es:[bx+8112],al ;mark or clear preceding odd row
                add     bx,80           ;skip a full screen row
                loop    marker_loop     ;66 times is the full display area
                pop     es
                pop     bp
                ret     2               ;clean the stack
marker_small    endp


;-----------------------------------------------------
;               Interrupt Service Routine
;-----------------------------------------------------
;This routine assumes it knows the length and offsets within a data
; structure in the pascal code. Any changes in the definition of
; that structure or even the compiler may require modifications.
                public  isr
```

```
isr             proc    near

                sti                     ;OK to interrupt me.

                push    ds              ;might be another isr's dseg
                mov     ds,cs:data_seg  ;point to pascal's dgroup

                push    ax
                push    bx
                push    cx
                push    dx

                mov     cx,num_913s     ;how many installed?
;---------------
isr_loop:
;---------------
                mov     bx,cx           ;currently serviced 913
                mov     al,bl           ;select 913
                dec     al              ;index 0-7 instead of 1-8
                mov     dx,292h         ;point to 913 address port
                out     dx,al

;----- check if sweep completed on this 913 (may have more than one) -----
                mov     dx,294h         ;913 status and control register
                in      al,dx           ;read his status
                test    al,10000000b
                jnz     pass_completed  ;bit 7 = 1, pass completed
                jmp     isr_loop_end    ;bit 7 = 0, pass not completed
pass_completed:
                mov     al,81h          ;first set the control register
                out     dx,al           ; bits 0 and 7 (stop? ST.)

;----- bx point to the sys[cur_913] start address -----
                shl     bx,1            ;multiply by 64 - structure length
                shl     bx,1
                shl     bx,1
                shl     bx,1
                shl     bx,1
                shl     bx,1
                add     bx,offset dgroup:sys ;now points to structure start

;----- assume we're thru -----
                mov     byte ptr [bx + active_offs],0

;----- check user hardware stop -----
```

```
;we will not restart if user stop is active
; and we must clear the active flag
                in      al,dx                   ;read 913 status register
                test    al,00000001b            ;isolate bit 0
                jnz     isr_loop_end            ;bit 0=1, usr hardware stop

;----- increase pass_count -----
                mov     ax,[bx + pass_offs]     ;pick up pass_count_low
                inc     ax                      ;bump it
                mov     [bx + pass_offs],ax     ;and put it away
                jnz     no_overflow             ;pass_count_low over 64k?
                inc     word ptr [bx + pass_offs + 2]
no_overflow:

;----- compare pass_count with pass_count_preset -----
                cmp     ax,[bx + pass_pre_offs]       ;pass_count_lo equal?
                jne     not_equal
                mov     ax,[bx + pass_offs + 2]       ;pass_count_high ?
                cmp     ax,[bx + pass_pre_offs + 2]
                jne     not_equal
                jmp     isr_loop_end    ;don't restart if preset equalled
not_equal:

;----- check abort_flag for user conditional stop -----
                cmp     byte ptr [bx + abort_offs], 1   ;conditional stop?
                je      isr_loop_end                    ;yes, don't restart

;----- check switch_flag -----
                cmp     switch_flag, 1                  ;switch auto/sum?
                jne     no_switch
                mov     switch_flag, 0
                xor     byte ptr [bx + sum_offs], 1     ;sum = not(sum)
                xor     byte ptr [bx + cbyte_offs], 00001000b  ;cbyte bit 3
                mov     word ptr [bx + pass_offs], 1    ;pass_count = 1
                mov     word ptr [bx + pass_offs + 2], 0
                mov     sweep_count, 0
                jmp     start_next_sweep                ;skip drift control
no_switch:

;----- check drift_control -----
check_drift_control:
                cmp     drift_control, 1
                je      drift_ctrl_on
drift_ctrl_off:
                mov     sweep_count, 0
```

```
                jmp     start_next_sweep
drift_ctrl_on:
                call    stabilization
                cmp     err_code, 0
                je      start_next_sweep        ;err code = 0, no err
                mov     sweep_count, 0          ;err code <> 0, sth wrong
                jmp     isr_loop                ;skip start
;----- start next sweep -----
start_next_sweep:
                mov     byte ptr [bx + active_offs],1   ;still running
                mov     al,[bx + cbyte_offs]            ;from pascal array
                out     dx,al                           ;done!

;--------------
isr_loop_end:   dec     cx                      ;do all existing 913's
                jz      isr_loop_done           ;usually only one
                jmp     isr_loop
isr_loop_done:
;--------------

                mov     ax,cur_913      ;must restore for main program
                dec     ax              ;1..8 maps to 0..7
                mov     dx,292h         ;address port
                out     dx,al           ;as we found it

                pop     dx
                pop     cx
                pop     bx
                cli
                mov     al,20h
                out     20h,al          ;tell 8259 we are thru
                pop     ax
                pop     ds              ;clean up

                iret
isr             endp
;--------------------------------

;----------------------------
stabilization   proc    near
;----------------------------
                push    ax
                push    bx
                push    cx
                push    dx
```

```
                push    es
                push    si

                mov     ax, adcam1_seg          ;segment of adcam1 = d000h
                mov     es, ax                  ;es has this segment
                                                ;ds has PASCAL data segment
                                                ;bx point to sys[ ]

                mov     al, [bx + sum_offs]
                shl     al, 1
                add     al, finesse_control     ;al = sum *2 +finesse_ctrl

                cmp     al, 00000011b
                je      sum_drift_finesse

                cmp     al, 00000010b
                je      sum_drift_only

                cmp     al, 00000001b
                je      auto_drift_finesse
                jne     auto_drift_only

sum_drift_finesse:
                call    sum8
                jmp     stabilization_end

sum_drift_only:
                call    sum2
                jmp     stabilization_end

auto_drift_finesse:
                call    auto4
                jmp     stabilization_end

auto_drift_only:
                call    auto1
                jmp     stabilization_end

stabilization_end:
                pop     si
                pop     es
                pop     dx
                pop     cx
                pop     bx
                pop     ax
```

```
                ret
;-----------------------------
stabilization   endp
;-----------------------------


;-----------------------------
auto4           proc    near
;-----------------------------
;auto mode:
;drift stabilization and finesse optimization
;4 sweeps a cycle

                inc     sweep_count          ;count + 1 after each sweep

                cmp     sweep_count, 1
                je      auto4_1
                cmp     sweep_count, 2
                je      auto4_2
                cmp     sweep_count, 3
                je      auto4_3
                cmp     sweep_count, 4
                je      auto4_4                     ;it must be 1..4
auto4_err:
                mov     err_code, err_sweep     ;never get to here
                ret

auto4_1:        jmp     auto4_sweep1
auto4_2:        jmp     auto4_sweep2
auto4_3:        jmp     auto4_sweep3
auto4_4:        mov     sweep_count, 0          ;4 sweeps per cylce
                jmp     auto4_sweep4
;-----------------------------
auto4_sweep1:
; drift correction
; finesse x_test_tilt
                call    calc_left
                call    calc_right
                call    calc_center1

                call    drift_correction
                call    x_test_tilt
                mov     update_drift, 0         ;tell pascal update display
                ret
;-----------------------------
```

```
auto4_sweep2:
; finesse x_tilt_corection
                call    calc_center2
                call    x_correction
                mov     update_finesse, 0
                ret
;---------------------------
auto4_sweep3:
; drift correction
; finesse y_test_tilt
                call    calc_left
                call    calc_right
                call    calc_center1

                call    drift_correction
                call    y_test_tilt
                mov     update_drift, 0
                ret
;---------------------------
auto4_sweep4:
; finesse y_tilt_corection
                call    calc_center2
                call    y_correction
                mov     update_finesse, 0
                ret
;-------------------------------
auto4           endp
;-------------------------------


;-------------------------------
auto1           proc    near
;-------------------------------
;auto mode:
;drift stabilization only, no finesse optimization
;1 sweep a cycle
                call    calc_left
                call    calc_right

                call    drift_correction
                mov     update_drift, 0

                mov     sweep_count, 0          ;1 sweep per cylce
                ret
;-------------------------------
```

```
auto1           endp
;-------------------------------


;-------------------------------
sum8            proc    near
;-------------------------------
;sum mode:
;drift stabilization and finesse optimization
;8 sweeps a cycle
                inc     sweep_count          ;count + 1 after each sweep

                cmp     sweep_count, 1
                je      sum8_1
                cmp     sweep_count, 2
                je      sum8_2
                cmp     sweep_count, 3
                je      sum8_3
                cmp     sweep_count, 4
                je      sum8_4
                cmp     sweep_count, 5
                je      sum8_5
                cmp     sweep_count, 6
                je      sum8_6
                cmp     sweep_count, 7
                je      sum8_7
                cmp     sweep_count, 8
                je      sum8_8                       ;must be 1..8
sum8_err:
                mov     err_code, err_sweep     ;never get to here
                ret

sum8_1:         jmp     sum8_sweep1
sum8_2:         jmp     sum8_sweep2
sum8_3:         jmp     sum8_sweep3
sum8_4:         jmp     sum8_sweep4
sum8_5:         jmp     sum8_sweep5
sum8_6:         jmp     sum8_sweep6
sum8_7:         jmp     sum8_sweep7
sum8_8:         mov     sweep_count, 0       ;8 sweeps per cylce
                jmp     sum8_sweep8
;---------------------------
sum8_sweep1:
                call    calc_left
                call    calc_right
```

```
                call    calc_center1

                call    save_left
                call    save_right
                call    save_center1
                ret
;---------------------------
sum8_sweep2:
                call    calc_left
                call    calc_right
                call    calc_center1

                call    net_left
                call    net_right
                call    net_center1

                call    drift_correction
                call    x_test_tilt
                mov     update_drift, 0
                ret
;---------------------------
sum8_sweep3:
                call    calc_center2
                call    save_center2
                ret
;---------------------------
sum8_sweep4:
; finesse x correction
                call    calc_center2
                call    net_center2

                call    x_correction
                mov     update_finesse, 0
                ret
;---------------------------
sum8_sweep5:
;sweep 5 is the same as sweep 1
                call    calc_left
                call    calc_right
                call    calc_center1

                call    save_left
                call    save_right
                call    save_center1
                ret
```

```
;----------------------------
sum8_sweep6:
;similar to sweep2 except y_test_tilt
                call    calc_left
                call    calc_right
                call    calc_center1

                call    net_left
                call    net_right
                call    net_center1

                call    drift_correction
                call    y_test_tilt
                mov     update_drift, 0
                ret
;----------------------------
sum8_sweep7:
;sweep7 is the same as sweep 3
                call    calc_center2
                call    save_center2
                ret
;----------------------------
sum8_sweep8:
;similar to sweep4 except y_correction
                call    calc_center2
                call    net_center2

                call    y_correction
                mov     update_finesse, 0
                ret
;-----------------------------
sum8            endp
;-----------------------------


;-----------------------------
sum2            proc    near
;-----------------------------
;sum mode:
;drift stabilization, no finesse optimization
;2 sweeps per cycle
                inc     sweep_count          ;count + 1 after each sweep

                cmp     sweep_count, 1
                je      sum2_1
```

```
                cmp     sweep_count, 2              ;it must be 1..2
                je      sum2_2
sum2_err:
                mov     err_code, err_sweep     ;never get to here
                ret

sum2_1:         jmp     sum2_sweep1
sum2_2:         mov     sweep_count, 0      ;2 sweeps per cylce
                jmp     sum2_sweep2
;---------------------------
sum2_sweep1:
                call    calc_left
                call    calc_right

                call    save_left
                call    save_right
                ret
;---------------------------
sum2_sweep2:
                call    calc_left
                call    calc_right

                call    net_left
                call    net_right

                call    drift_correction
                mov     update_drift, 0
                ret
;-----------------------------
sum2            endp
;-----------------------------


;---------------------------
calc_count      proc    near
;---------------------------
calc_left:
                mov     si, ref_channel
                sub     si, dr_window_width     ;1st channel of left window
                shl     si, 1
                shl     si, 1                   ;4 bytes data per channel

                xor     ax, ax
                xor     dx, dx
                mov     cx, dr_window_width
left_sum_loop:
```

```
                add     ax, es:[si]
                adc     dx, es:[si+2]
                add     si, 4
                loop    left_sum_loop

                and     dh, 01111111b           ;MSB is ROI mark
                mov     cs:left_lo, ax
                mov     cs:left_hi, dx
                ret
;-------------------
calc_right:
                mov     si, ref_channel
                inc     si                      ;1st channel of right window
                shl     si, 1
                shl     si, 1

                xor     ax, ax
                xor     dx, dx
                mov     cx, dr_window_width
right_sum_loop:
                add     ax, es:[si]
                adc     dx, es:[si+2]
                add     si, 4
                loop    right_sum_loop

                and     dh, 01111111b
                mov     cs:right_lo, ax
                mov     cs:right_hi, dx
                ret
;-------------------
calc_center1:
                mov     si, ref_channel
                mov     ax, fi_window_width
                shr     ax, 1                   ;finesse window div 2
                sub     si, ax                  ;ref_ch - (window div 2)
                shl     si, 1                   ; = 1st channel of fi window
                shl     si, 1

                xor     ax, ax
                xor     dx, dx
                mov     cx, fi_window_width
center1_sum_loop:
                add     ax, es:[si]
                adc     dx, es:[si+2]
                add     si, 4
```

```
                loop    center1_sum_loop

                and     dh, 01111111b
                mov     cs:center1_lo, ax
                mov     cs:center1_hi, dx
                ret
;-------------------
calc_center2:
                mov     si, ref_channel
                mov     ax, fi_window_width
                shr     ax, 1
                sub     si, ax
                shl     si, 1
                shl     si, 1

                xor     ax, ax
                xor     dx, dx
                mov     cx, fi_window_width
center2_sum_loop:
                add     ax, es:[si]
                adc     dx, es:[si+2]
                add     si, 4
                loop    center2_sum_loop

                and     dh, 01111111b
                mov     cs:center2_lo, ax
                mov     cs:center2_hi, dx
                ret
;-----------------------------
calc_count      endp
;-----------------------------


;-----------------------------
save_count      proc    near
;-----------------------------
save_left:
                mov     ax, cs:left_lo
                mov     cs:old_left_lo, ax
                mov     ax, cs:left_hi
                mov     cs:old_left_hi, ax
                ret
;-------------------
save_right:
                mov     ax, cs:right_lo
```

```
                mov     cs:old_right_lo, ax
                mov     ax, cs:right_hi
                mov     cs:old_right_hi, ax
                ret
;-------------------
save_center1:
                mov     ax, cs:center1_lo
                mov     cs:old_center1_lo, ax
                mov     ax, cs:center1_hi
                mov     cs:old_center1_hi, ax
                ret
;-------------------
save_center2:
                mov     ax, cs:center2_lo
                mov     cs:old_center2_lo, ax
                mov     ax, cs:center2_hi
                mov     cs:old_center2_hi, ax
                ret
;---------------------------
save_count      endp
;---------------------------


;---------------------------
net_count       proc    near
;---------------------------
net_left:
                mov     ax, cs:left_lo
                mov     dx, cs:left_hi
                sub     ax, cs:old_left_lo
                sbb     dx, cs:old_left_hi
                mov     cs:left_lo, ax
                mov     cs:left_hi, dx
                ret
;-------------------
net_right:
                mov     ax, cs:right_lo
                mov     dx, cs:right_hi
                sub     ax, cs:old_right_lo
                sbb     dx, cs:old_right_hi
                mov     cs:right_lo, ax
                mov     cs:right_hi, dx
                ret
;-------------------
net_center1:
```

```
                mov     ax, cs:center1_lo
                mov     dx, cs:center1_hi
                sub     ax, cs:old_center1_lo
                sbb     dx, cs:old_center1_hi
                mov     cs:center1_lo, ax
                mov     cs:center1_hi, dx
                ret
;-------------------
net_center2:
                mov     ax, cs:center2_lo
                mov     dx, cs:center2_hi
                sub     ax, cs:old_center2_lo
                sbb     dx, cs:old_center2_hi
                mov     cs:center2_lo, ax
                mov     cs:center2_hi, dx
                ret
;-------------------------------------
net_count       endp
;-------------------------------------


;-------------------------------------
drift_correction        proc    near
;-------------------------------------
; drift = right window counts - left window counts
; if (drift > 0) then begin
;   if (drift > 255) then begin
;     dac0 increase;
;   end
;   else begin
;     if (drift*drift > counts) then dac0 increase;
;     else no correction;
;   end;
; end
; else if (drift < 0) then begin
;   drift = - drift
;   if (drift > 255) then begin
;     dac0 decrease;
;   end
;   else begin
;     if (drift*drift > counts) then dac0 decrease;
;     else no correction;
;   end;
; end
; else begin
;   no correction
```

```
; end;


;normally, 0 < window counts < 0000FFFFh,  ie. hi_word = 0
                cmp     cs:left_hi, 0
                jnz     left_overflow
                cmp     cs:right_hi, 0
                jnz     right_overflow
                jz      drift_no_overflow

left_overflow:  mov     err_code, err_left
                ret
right_overflow: mov     err_code, err_right
                ret

;-------------------
drift_no_overflow:
;calculate counts differences, using only lo_word since hi_word = 0
                cmp     cs:drift_corr_done, 0 ;correction done last time?
                jz      calc_current_drift    ;no,
                mov     dr_window_differ, 0   ;yes, clear accumulated drift
                mov     cs:drift_corr_done, 0

calc_current_drift:
                mov     dx, cs:right_lo       ;dx =  right
                add     dx, cs:left_lo        ;dx = (right + left)
                rcr     dx, 1                 ;dx = (right + left) / 2
                mov     dr_window_counts, dx

                mov     ax, cs:right_lo
                sub     ax, cs:left_lo        ;ax = right - left
                add     ax, dr_window_differ  ;accumulate drift counts
                mov     dr_window_differ, ax

                cmp     ax, 0                 ;drift to the right or left?
                jg      drift_right           ;drift > 0
                jl      drift_left            ;drift < 0
                ret                           ;drift = 0

drift_right:
                cmp     ax, 255               ;drift > 255 ?
                ja      volt_D_up             ;yes, make correction
                mul     al                    ;ax = al*al = differ^2
                cmp     ax, dx                ;drift^2 >= counts ?
                jae     volt_D_up             ;yes, drift >= sqr(counts)
```

```
                ret                             ;no,  drift < error

drift_left:
                neg     ax                      ;ax = positive
                cmp     ax, 255                 ;ax > 255 ?
                ja      volt_D_down             ;yes
                mul     al                      ;ax = al*al = differ^2
                cmp     ax, dx
                jae     volt_D_down
                ret                             ;drift < error

;---------------
volt_D_up:
;dac_data_drift = dac_data_drift + drift_corr_step
                mov     al, 0                           ;select channel 0
                mov     dx, card0_dac_ctrl_reg_Hi
                out     dx, al

                mov     ax, dac_data_drift
                add     ax, drift_corr_step     ;go up by drift_corr_step
                cmp     ax, 0FFFh
                jg      drift_overflow          ;drift volt > 10v

                mov     dac_data_drift, ax      ;drift volt < 10v
                mov     dx, card0_dac_data_reg
                out     dx, ax
                mov     cs:drift_corr_done, 1
                ret

;---------------
volt_D_down:
;dac_data_drift = dac_data_drift - drift_corr_step
                mov     al, 0                   ;select channel 0
                mov     dx, card0_dac_ctrl_reg_Hi
                out     dx, al

                mov     ax, dac_data_drift
                sub     ax, drift_corr_step     ;go down by drift_corr_step
                cmp     ax, 000
                jl      drift_overflow          ;drift volt < 0v

                mov     dac_data_drift, ax      ;drift volt => 0v
                mov     dx, card0_dac_data_reg
                out     dx, ax
                mov     cs:drift_corr_done, -1
```

```
                ret

drift_overflow: mov     err_code, err_drift
                ret
;----------------------------
drift_correction  endp
;----------------------------


;----------------------------
x_test_tilt     proc    near
;----------------------------
                call    check_dac
                cmp     err_code, 0
                jne     x_test_tilt_end         ;dac data over range

;--- x_test ---
;dac_data_tilt_A := dac_data_tilt_A + x_test_data
;dac_data_tilt_C := dac_data_tilt_C - x_test_data
                mov     al, 0                           ;select channel 0
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_A
                add     ax, x_test_data
                mov     dac_data_tilt_A, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax

                mov     al, 1                           ;select channel 1
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_C
                sub     ax, x_test_data
                mov     dac_data_tilt_C, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax
x_test_tilt_end:
                ret

;--------------
x_test_back:
;dac_data_tilt_A := dac_data_tilt_A - x_test_data
;dac_data_tilt_C := dac_data_tilt_C + x_test_data
                mov     al, 0                           ;select channel 0
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
```

```
                mov     ax, dac_data_tilt_A
                sub     ax, x_test_data
                mov     dac_data_tilt_A, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax

                mov     al, 1                           ;select channel 1
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_C
                add     ax, x_test_data
                mov     dac_data_tilt_C, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax
x_test_back_end:
                ret
;----------------------------
x_test_tilt     endp
;----------------------------


;-------------------------------------
x_correction    proc    near
;-------------------------------------
;x_test_back
;if (center differ > 0) then begin
;   if (center differ > statistical error) then begin
;      correct_x;
;      finesse_correction := 1;
;   end;
;end
;else if (center differ < 0) then begin
;   x_test_data := - x_test_data;               ;reverse tilt direction
;   x_corr_data := - x_corr_data;
;   if (|center differ| > statistical error) then begin
;      correct_x;
;      finesse_correction := -1;
;   end;
;end
;else begin
;  x_test_data := - x_test_data;                ;reverse tilt direction
;  x_corr_data := - x_corr_data;
;  finesse_correction := 0;
;end;
;
```

```
;dac over range check has been done in tilt_test
;therefore dac over range will not occur during correction
; because normally correction step is smaller than test step

                call    x_test_back             ;back from test tilt

;normally, 0 < window counts < 0000FFFFh,  ie. hi_word = 0
                cmp     cs:center1_hi, 0
                jnz     x_c1_overflow
                cmp     cs:center2_hi, 0
                jnz     x_c2_overflow
                jz      x_calc_center_differ

x_c1_overflow:  mov     err_code, err_center1
                ret
x_c2_overflow:  mov     err_code, err_center2
                ret

;calculate center differ, using only lo_word since hi_word = 0
x_calc_center_differ:
                mov     dx, cs:center1_lo
                add     dx, cs:center2_lo       ;dx = (center1 + center2)
                rcr     dx, 1                   ;dx = (center1 + center2)/2
                mov     fi_window_counts, dx

                mov     ax, cs:center2_lo
                sub     ax, cs:center1_lo       ;ax = center differ
                mov     fi_window_differ, ax

                cmp     ax, 0
                jg      x_test_right            ;differ > 0
                jl      x_test_wrong            ;differ < 0

x_test_no_differ:
                neg     x_test_data             ;differ = 0
                neg     x_corr_data             ;change dirction next time
                ret                             ;no correction

x_test_wrong:                                   ;ax =  differ  < 0
                neg     ax                      ;ax = |differ| > 0
                neg     x_test_data             ;change dirction next time
                neg     x_corr_data

x_test_right:                                   ;ax =|differ|, dx =center1
```

```
;note: statistical error = sqrt(N)
;if (differ) > sqrt(center counts) then correction
;ie. if (differ * differ) > (center counts) then correction
;since maximum counts = 65535, so maximum error = sqr(counts) = 256
; so if (differ) > 255 then correction
                cmp     ax, 255                 ;ax = differ > 255 ?
                ja      do_x_corr               ;yes
                                                ;no
                mul     al                      ;differ^2 = al*al -> ax
                cmp     ax, dx                  ;differ >= error?
                jae     do_x_corr               ;yes
                ret                             ;no

do_x_corr:
;dac_data_tilt_A := dac_data_tilt_A + x_corr_data
;dac_data_tilt_C := dac_data_tilt_C - x_corr_data
                mov     al, 0                           ;select channel 0
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_A
                add     ax, x_corr_data
                mov     dac_data_tilt_A, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax

                mov     al, 1                           ;select channel 1
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_C
                sub     ax, x_corr_data
                mov     dac_data_tilt_C, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax
                ret
;-----------------------------
x_correction    endp
;-----------------------------


;-----------------------------
y_test_tilt     proc    near
;-----------------------------
                call    check_dac
                cmp     err_code, 0
                jne     y_test_tilt_end         ;dac data over range
```

```
;--- y_test ---
;dac_data_tilt_A := dac_data_tilt_A + y_test_data
;dac_data_tilt_C := dac_data_tilt_C + y_test_data
;dac_data_tilt_B := dac_data_tilt_B - y_test_data *2
                mov     al, 0                           ;select card1_dac0
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_A
                add     ax, y_test_data
                mov     dac_data_tilt_A, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax

                mov     al, 1                           ;select card1_dac1
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_C
                add     ax, y_test_data
                mov     dac_data_tilt_C, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax

                mov     al, 1                           ;select card0_dac1
                mov     dx, card0_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_B
                sub     ax, y_test_data
                sub     ax, y_test_data
                mov     dac_data_tilt_B, ax
                mov     dx, card0_dac_data_reg
                out     dx, ax
y_test_tilt_end:
                ret

;----------------------------
y_test_back:
;dac_data_tilt_A := dac_data_tilt_A - y_test_data
;dac_data_tilt_C := dac_data_tilt_C - y_test_data
;dac_data_tilt_B := dac_data_tilt_B + y_test_data *2
                mov     al, 0                           ;select card1_dac0
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_A
                sub     ax, y_test_data
                mov     dac_data_tilt_A, ax
```

```
                mov     dx, card1_dac_data_reg
                out     dx, ax

                mov     al, 1                           ;select card1_dac1
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_C
                sub     ax, y_test_data
                mov     dac_data_tilt_C, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax

                mov     al, 1                           ;select card0_dac1
                mov     dx, card0_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_B
                add     ax, y_test_data
                add     ax, y_test_data
                mov     dac_data_tilt_B, ax
                mov     dx, card0_dac_data_reg
                out     dx, ax
y_test_back_end:
                ret
;-----------------------------
y_test_tilt     endp
;-----------------------------

;-------------------------------------
y_correction    proc    near
;-------------------------------------
                call    y_test_back

;normally, 0 < window counts < 0000FFFFh,  ie. hi_word = 0
                cmp     cs:center1_hi, 0
                jnz     y_c1_overflow
                cmp     cs:center2_hi, 0
                jnz     y_c2_overflow
                jz      y_calc_center_differ

y_c1_overflow:  mov     err_code, err_center1
                ret
y_c2_overflow:  mov     err_code, err_center2
                ret

y_calc_center_differ:
```

```
                mov     dx, cs:center1_lo
                add     dx, cs:center2_lo       ;dx = (center1 + center2)
                rcr     dx, 1                   ;dx = (center1 + center2)/2
                mov     fi_window_counts, dx

                mov     ax, cs:center2_lo
                sub     ax, cs:center1_lo       ;ax = center differ
                mov     fi_window_differ, ax

                cmp     ax, 0
                jg      y_test_right            ;differ > 0
                jl      y_test_wrong            ;differ < 0

y_test_no_differ:
                neg     y_test_data             ;differ = 0
                neg     y_corr_data             ;change dirction next time
                ret                             ;differ = 0

y_test_wrong:
                neg     ax                      ;ax = |differ|
                neg     y_test_data             ;change tilt dirction
                neg     y_corr_data

y_test_right:
                cmp     ax, 255                 ;differ > 255 ?
                ja      do_y_corr               ;yes
                                                ;no
                mul     al                      ;differ^2 = al*al -> ax
                cmp     ax, dx                  ;differ >= error?
                jae     do_y_corr               ;yes,
                ret                             ;no,

do_y_corr:
;dac_data_tilt_A := dac_data_tilt_A + y_corr_data
;dac_data_tilt_C := dac_data_tilt_C + y_corr_data
;dac_data_tilt_B := dac_data_tilt_B - y_corr_data *2
                mov     al, 0                           ;select card1_dac0
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_A
                add     ax, y_corr_data
                mov     dac_data_tilt_A, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax
```

```
                mov     al, 1                           ;select card1_dac1
                mov     dx, card1_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_C
                add     ax, y_corr_data
                mov     dac_data_tilt_C, ax
                mov     dx, card1_dac_data_reg
                out     dx, ax

                mov     al, 1                           ;select card0_dac1
                mov     dx, card0_dac_ctrl_reg_Hi
                out     dx, al
                mov     ax, dac_data_tilt_B
                sub     ax, y_corr_data
                sub     ax, y_corr_data
                mov     dac_data_tilt_B, ax
                mov     dx, card0_dac_data_reg
                out     dx, ax
y_correction_end:
                ret
;---------------------------
y_correction    endp
;---------------------------

;----------------------------
check_dac       proc    near
;----------------------------
;check dac over range
                mov     ax, dac_data_tilt_A
                cmp     ax, 4000
                jg      A_overflow
                cmp     ax, 100
                jl      A_overflow

                mov     ax, dac_data_tilt_B
                cmp     ax, 4000
                jg      B_overflow
                cmp     ax, 100
                jl      B_overflow

                mov     ax, dac_data_tilt_C
                cmp     ax, 4000
                jg      C_overflow
                cmp     ax, 100
                jl      C_overflow
```

```
                ret                                 ;no overflow
;--------------
A_overflow:     mov     err_code, err_tilt_A
                ret
B_overflow:     mov     err_code, err_tilt_B
                ret
C_overflow:     mov     err_code, err_tilt_C
                ret
;-----------------------------
check_dac       endp
;-----------------------------

;-------------------------------------------------
support         ends
;-------------------------------------------------
                end
```