Brief description of the Bomem data acquisition and processing using FTS

The data is written in a binary file format knows as Fourier Transform System, or FTS, format, which is transportable across different platforms. Each FTS record, a stored interferogram or spectrum (transformed interferogram), consists of two parts. The first part is a 512-byte header segment which contains integer and character data characterizing the scan. The data portion of the scan is written as a series of 512 byte blocks of data, where the actual number of blocks containing the data is accessible through one of the header variables. This file format allows both interferogram and spectrum data to be accessed via a series of FTS system subroutines and utility programmes (both C and Fortran). The definition of the header parameters may be found in Appendix 1.

Although the FTS file format is binary and thus not readable directly, it can be transported between VMS, Unix and PC machines. The system subroutines are organized in several programmes according to functionality. In addition, the system dependent subroutines are isolated in a single suite, and are available for the three aforementioned operating systems.

The current data acquisition programme for the Bomem interferometer, TMAIN.EXP, utilizes the original machine language DMA interface routines from Bomem. It is designed to operate with either the SEQ001 (IRQ=3, DAQ=1) or SEQ005 (IRQ=3, DAQ=1, and DACK=1) parallel interface PC cards[1], and will acquire interferograms and output them to disk in the FTS format. Bomem now maintains that the drive system of the interferometer has no hysteresis between the forward and reverse scans. However, for early models this may not be the case. For that reason, the acquisition programme treats each interferogram as a separate forward and reverse pair of scans and outputs them separately. After Fourier transforming each scan-pair into spectra separately, the spectra may be combined without hysteresis effects.

To allow for automated operation, the acquisition programme reads a control file which provides it with information as to when to acquire data, how many interferograms to coherently add up before outputting the result to disk, and where to output the data. It should be noted that each interferogram would consist of a scan-pair. Thus, requesting 5 interferograms to be added coherently before writing to disk will result in 5 forward scans and 5 reverse scans being acquired and added separately, and the scan-pair written as separate FTS records (each with its own header) to disk. In addition, a comment for the experiment, and the latitude, longitude and altitude of the experiment will be read from the control file and encoded into the FTS header record. A complete description and format of the ASCII control file is given in Appendix 2.

When the acquisition programme is run, it opens a data file for output. A specified number of interferograms (each consisting of a forward and reverse scan-pair) will be stored in the file whose name will consist of the year, the day of the year, and a counter (A0-Z0, A1-Z1...A9-Z9) when the file was opened, with the extension

---

[1] Note, some users have experienced difficulties with the SEQ005 board at a resolution of 2 cm^{-1}. Apparently an extra byte is generated from the board which overflows the DOS 64K memory boundary. If this occurs, the highest resolution one should use is 4 cm^{-1}.

".IFG". For example, 0249A0.IFG would be the first file opened on day 249 of the year 2000. These file names, as well as the interferogram sequence number, will be encoded into the FTS header record to provide tractability to the original file.

The DMA interface is limited to a 64K-byte DOS memory segment. However, the data processing of the acquired data requires arrays which exceed this DOS segment limit. For that reason, the acquisition software must switch between 64K-segmented mode to service the DMA request, and a linear address space to process and write out the interferograms. Thus, the current version of the software runs under DOS version 4.0, or 6.22 and higher. Some users have experienced difficulties on machines with a 486 architecture, so an 80386 (with a 387 co-processor) machine is recommended. A thesis detailing this software is included as Appendix 3. In order to switch between the 64K and linear address mode, the Phar-Lap DOS extender RUN22B.EXE is used. In order for this extender to function properly, it requires control over the memory. Thus, the CONFIG.SYS file used must invoke HIMEM.SYS and EMM386.EXE as follows:

DEVICE=<path>HIMEM.SYS

DEVICE=<path>EMM386.EXE RAM

Where <path> is the DOS path to these system programmes.

To begin data acquisition, the DOS extender, RUN22B.EXE, must be loaded along with the data acquisition programme, TMAIN.EXP. The name of the data acquisition control file is given as a command line parameter. Thus, to begin data acquisition, utilize the command:

RUN22B TMAIN control-file-name

The programme will load, open an output file, and check to see if the instrument resolution is different from the previous scan it has processed. Since this will always the case for the first scan, it will give the message:

Resolution Mismatch

and then begin to follow the commands in the control file. If the programme is being commanded to pause during the present time interval, it will give the message:

/Waiting ...

where the initial / character will rotate. If it is being commanded to take data during this time interval, it will message:

Acquiring scan XX of YY

where XX is the present scan being acquired and YY is the total number of scans to be acquired. Once again, each interferogram to be coherently added consists of 2 scan-pairs. Thus, the number, YY, displayed will be twice as large as the number specified in the control file for interferograms to be added. Upon completion of the specified number of interferograms to be added, the programme will output the scans to the data file in FTS format. It will also display for each scan a row of information giving 1) the scan number, 2) the interferogram number, 3) the starting time of the first scan in the addition, 4) the ending time of the last scan in the addition, 5) the number of samples from the start to the centre of the scan, 6) the number of samples from the centre to the end of the scan, and 7) the centre amplitude in volts.

Appendix 1

Header Parameter Definition

SDL/93-032

FOURIER TRANSFORM SPECTROSCOPY (FTS) HEADER PARAMETERS

Chuck Harris

17 September 1993

FOREWORD

This document describes the header variable definitions for the Fourier Transform Spectroscopy (FTS) software developed by the Space Dynamics Laboratory at Utah State University (SDL/USU), and used with most SDL/USU programs to transform interferograms into spectra.

1. FTS HEADER VARIABLE DEFINITIONS

1.1 INTRODUCTION

This document describes the header variable definitions for the Fourier Transform Spectroscopy (FTS) software written by Chuck Harris, and used with most SDL/USU programs to transform interferograms into spectra.

The FTS header is stored at the beginning of the scan in a data file and relates experimental configuration and data reduction methods from one data processing stage to the next. The parameters stored in the header are necessary for understanding how the data were collected and processed.

The header is 512 bytes and is stored in binary format as 1- or 4-byte variables. The header variables are listed in Table 1. This table also contains the byte number and parameter name for the variable.

2. 1-BYTE VARIABLES

2.1 *QQDSRC* FHDR(1..32) Source of Current Scan

The *QQDSRC* variable is a string of 32 ASCII characters that indicates the source of data used to generate the current scan data. Intended as a bookkeeping device, it enables users to trace a string of entries to a lab notebook entry or to the original data. However, for increased flexibility, this string is not automatically updated by the FTS programs; therefore, the user must exercise caution. To be compatible with FORTRAN-77 strings, the string must be filled out to the end with spaces.

2.2 *QQRMRK* FHDR(33..64) Remark Concerning Current Scan

The *QQRMRK* variable is a string of 32 ASCII characters that provides relative comments about the data. It is intended to be used as a note-taking device. To be compatible with FORTRAN-77 strings, the string must be filled out to the end with spaces.

2.3 *QQSYSV* I1FHDR(1) FTS System Version Used

The *QQSYSV* variable is the FTS system version number. It tracks the programs and conventions used to create the current data. It should be updated when major changes are made to the scan header layout or to the data processing procedure.

2.4 *QQDTYP* I1FHDR(2) Scan Data Word Type

The *QQDTYP* variable records the types of numbers used to store the scan data. The possible values for the data words are:

0 : Integer*2
1 : Integer*4
2 : Real*4
(3 Real*8) - not on files

Note: complex data are stored as two data words. In FTS version 2, real data types are not allowed in data arrays, but not in files. Integers are stored in order from most significant bit (MSB) to least significant bit (LSB).

2.5 *QQDKND* I1FHDR(3) Scan Data Kind

The *QQDKND* variable records the kind of data stored in the scan. The different kinds of data are:

0 : Interferogram data
1 : Transformed spectral data

The units in which the data are expressed are explained under *QQUNTS*. See also *QQDYUN, QQDXUN, QQCYUN, and QQCXUN.*

2.6 *QQDREP* I1FHDR(4) Scan Data Representation

The *QQDREP* variable describes what the numbers in the data represent. The variable *QQDTYP* defines the number type (Integer*2, Integer*4, Real*4). The values of *QQDREP* are:

0 : The real part of a complex number is stored. A real interferogram would have a value of 0, as would a phase-corrected spectrum whose real part only was stored.

1 : The imaginary part of a complex number is stored. This is most likely to occur when the imaginary part of a phase-corrected spectrum is of interest, such as a check of phase correction effectiveness. In practice this representation is almost never used, so that it mainly serves the purpose of completeness.

2 : Complex numbers are stored. The complex parts of the number are stored adjacently in the order (real, imaginary).

2.7 *QQDYUN* I1FHDR(5) Scan Data Y Units

The *QQDYUN* variable defines the units of a 'virtual' interferogram. Because a Fourier transform is an integral, the units of a spectrum are the product of the interferogram units with the units of the integration variable. This idea is carried through to define a unit for spectra which are the product of interferogram units (*QQDYUN*) and integration variable units (*QQDXUN*); thus *QQDYUN* does not necessarily reflect the actual units of the original interferogram unless the data set itself is an interferogram (*QQDKND* = 0). The values of *QQDYUN* are:

0 : ARBitrary units. This value is included for cases in which the actual units are unknown or unimportant. In general, ARBitrary units are used when the number output by an A/D is directly stored with no provision for the actual range of the A/D in actual units.

1 : Volts. The numbers stored represent volts. They may or may not represent 'constant' volts which have been corrected for various gain settings. The variable *QQGCOR* specifies the state of the gain correction.

2 : Photons/cm^2-sec-sr. This unit is chosen instead of Rayleighs or radiance because it is both more general and appropriate to the small field of view of interferometers. Rayleighs are good for optically thin plane layered sources whereas radiance is good for optically thick sources. Either may be derived from the current units if the situation warrants it.

3 : Watts/cm^2-sr. This unit is the energy equivalent of the preceding unit. The same discussion of its merits applies.

See *QQDXUN*, *QQCYUN*, *QQCXUN*, and *QQUNTS*.

2.8 *QQDXUN* I1FHDR(6) Scan Data X Units

The *QQDXUN* variable is the other half of the *QQDYUN* variable described above. The units of a spectrum are the product of the *QQDXUN* units and the *QQDYUN* units. The units of an interferogram are given by the *QQDYUN* alone. The defined values of *QQDXUN* are:

0 : Samples. Samples are the integration variable equivalent of ARB units for the interferogram. The distance between two interferogram points is taken as the unit length. Thus in an FFT ΔX is 1. In general this unit will be combined with QQSYUN = 0 or 1.

1 : Slide centimeters. In this case the distance between interferogram samples is taken as QQISMP x QQISLP x 10^{-10} slide centimeters. The 'slide' prefix emphasizes that this unit refers to the displacement of a slide carrying a mirror or wedge rather than to true optical path difference. In general this unit will be combined with QQSYUN = 0 or 1.

2 : Centimeters. This is the true optical path difference in Centimeters. In general it will only arise when the spectrum is corrected for the instrument response. Consequently it will generally be associated with QQDYUN = 2 or 3.

3:: Seconds. This unit is for the case when the interferogram is sampled with a clock or when it is desired to derive a sampling frequency from the beginning and ending times and the number of counts. It is most likely to be used for temporal spectra or for looking at system noise.

It would have been ideal to include a 'unitless' code in the *QQDYUN* variable to enable writing a single subroutine to generate a units string for labeling plot axis. Also see *QQDYUN*, *QQCYUN*, *QQCXUN*, and *QQUNTS*.

2.9 *QQDEVC* I1FHDR(7) Device Code

The *QQDEVC* variable records which instruments were used to obtain data. In practice it has proved of little value as too much control must be exerted to keep the values unique. Therefore, this variable may be considered obsolete. The user must track which instruments provide data.

This variable may be considered obsolete.

2.10 *QQMOTN* I1FHDR(8) Slide Direction

The *QQMOTN* variable is reserved for interferometers which scan in two directions. It has proved necessary to keep a record of the scan direction, because the instrument response is generally different for each direction. The correspondence of *QQMOTN* with actual slide motion is arbitrary, but it must be consistent. For interferometers which only take data in one direction, it is customary to set *QQMOTN* to 0. Defined values of *QQMOTN* are:

0 : Forward direction. This is somewhat arbitrary.

1 : Reverse direction.

2 : Undefined. This case can arise if scans of different directions are coadded.

2.11 *QQCDWN* I1FHDR(9) Count Down

The *QQCDWN* variable is a record of the number of counts which occur during each sampling interval. Counts are the smallest interval (space or time) at which data can be sampled. *QQCDWN* refers to the original data and not to any resampling which may take place later. It is

essentially a record of the instrument configuration. If data are resampled, QQISLP should be modified, not QQCDWN.

2.12 *QQGAN1* I1FHDR(10) Gain #1

2.13 *QQGAN2* I1FHDR(11) Gain #2

2.14 *QQGAN3* I1FHDR(12) Gain #3

The *QQGAN1*, *QQGAN2*, and *QQGAN3* variables may be considered as a unit. They are reserved to record the values of up to 3 gain settings which may have been used in acquiring the data. Because a gain setting may not correspond with an actual gain, the contents of these variables is purely symbolic. The user is responsible for making the connection between the symbolic values and the true gains. In general this connection will differ from instrument to instrument.

2.15 *QQTTYP* I1FHDR(13) Transform Type

The *QQTTYP* variable is a record of the transform type used to obtain a spectrum. The transform type is either Integer*2, Integer*4, or Real*4. Thus, this variable is a measure of the transform accuracy, but has no direct bearing on the resulting spectrum. The defined values of QQTTYP are:

0 : Integer*2. The transform was carried out in fixed point using 2 bytes per number.
1 : Integer*4. The transform was carried out in fixed point using 4 bytes per number.
2 : Real*4. The transform was carried out in single precision floating point. The FTS I/O subroutines are currently set up to use 4 byte single precision floating point numbers. However, the actual size of the floating point number is unspecified, and is likely to be machine dependent.

2.16 *QQTSFT* I1FHDR(14) Rescaling Shifts in Integer Transforms

The *QQTSFT* variable is necessary only if a fixed point transform is used. It is a count of the number of times the data were divided by 2 to avoid fixed point overflow. Like *QQTTYP*, *QQTSFT* has no direct bearing on the resulting spectrum, but may carry some information about the dynamic range of the data. It can be also be used to determine whether a larger word size should be used in the transform. *QQTSFT* is unrelated to the scaling of the stored data; scaling information is contained in *QQDSCL*.

2.17 QQPHCR I1FHDR(15) Phase Correction Flag

The *QQPHCR* variable is a flag that indicates whether or not the spectrum was phase corrected. It is generally redundant with the value of *QQPHSZ*, i.e., *QQPHSZ* = 0 implies no phase correction. However, the general philosophy of the FTS system is to avoid the deduction of action information, such as phase correction, from numeric parameters. Thus, this variable is included as an additional check. The defined values of *QQPHCR* are:

0 : No phase correction has taken place.
1 : Phase correction has taken place.

2.18 *QQAPOD* I1FHDR(16) Apodization Type

The *QQAPOD* variable is a symbolic value representing the type of apodization used on the spectral data. At the present time there are six possible Kaiser-Bessel apodizations available in the FTS system. If apodizations other than Kaiser-Bessel are desired, other general families of apodizations exist that are optimum in various respects and also easily interpreted.

The Kaiser-Bessel apodization functions are defined by :

$$weight(x) = \frac{I_0(\beta\sqrt{1-x^2})}{I_0(\beta)} \quad (1)$$

Where x ranges from -1 to 1, I_0 is the modified bessel function of the first kind, and β is a parameter. The resulting window in the spectral domain is given by:

$$window(v) = \frac{\sinh(\sqrt{\beta^2 - \Pi^2 v^2})}{I_o(\beta)\sqrt{\beta^2 - \Pi^2 v^2}} \quad (2)$$

Where β and I_0 are as above, sinh is the hyperbolic sine, and v is any real number. Note that for some values of v the square root will be imaginary. For this situation, sinh should be replaced by sin using the relationship $\sinh(jv) = j\sin(v)$ and the j's will cancel. The units of v are such that an increment of 1 represents a change in frequency of 1 cycle over the total range of the weight function. The integrated area is 1 using these frequency units.

The defined values of *QQAPOD* are:

QQAPOD	LOBE	FWHM	ZERO	β	$1/I_0(\beta)$
0	1.00000	1.207	1.000	0.000000000	$1.000000000\ 10^{+0}$
1	0.10000	1.737	1.747	4.499913997	$5.720873418\ 10^{-2}$
2	0.01000	2.127	2.525	7.283997681	$4.558416818\ 10^{-3}$
3	0.00100	2.443	3.304	9.892699526	$3.931900006\ 10^{-4}$
4	0.00010	2.717	4.079	12.42304040	$3.518708342\ 10^{-5}$
5	0.00001	2.961	4.850	14.90796851	$3.217936989\ 10^{-6}$

Where LOBE is the size of the first sidelobe relative to that of the sinc function, FWHM is the window full width at half maximum, ZERO is the location of the first window zero relative to the window center, β is the β in the formula for Weight, and $1/I_{0(\beta)}$ is the window normaliza-

tion factor. Note that for *QQAPOD* = 0, the window *is* the sinc function. Real FORTRAN functions for the weighting and window functions can be found in FTSWTAPD.

The function names are listed in the following table:

QQAPOD	WEIGHT	WINDOW
0	R4APD0	R4WND0
1	R4APD1	R4WND1
2	R4APD2	R4WND2
3	R4APD3	R4WND3
4	R4APD4	R4WND4
5	R4APD5	R4WND5

2.19 *QQNTRP* I1FHDR(17) Interpolation Zero Fill

The *QQNTRP* variable specifies whether or not the data was zero-filled before transform to allow accurate spectral interpolation. The FTS system is designed to allow easy and accurate interpolation of spectral data. Interpolating functions are available in the FTS system for use in interpolating real (phase corrected) and complex data, respectively. These functions are also capable of taking first and second derivatives. The functions are based on interpolating polynomials which minimize the least squares error in interpolating or differentiating band limited noise. For the interpolating functions to achieve their specified accuracy, the data must be oversampled by at least a factor of two. This can be achieved for the spectra by zero-filling the interferogram with zeros to twice its given size before transforming. Interferograms may also be interpolated if they are initially over-sampled, which is often the case. The defined values of *QQNTRP* are:

0 : Not zero-filled.
1 : Zero-filled to twice its given size.

Two additional features associated with interpolation should be noted. First, the interpolating function requires 8 data points to either side of the point being interpolated. This means that in order to interpolate spectra over the full range, additional points must be added outside of the range. The transform program TRNSFORM automatically extends the spectrum using symmetry. Second, if the spectrum has a nonlinear wavenumber scale, such as is the case for field widened interferometers, it is advisable to use zero-fill so that the spectrum can be easily and accurately resampled on a new scale if desired. See the section on interpolation for more information on the interpolating functions.

2.20 *QQPWGT* I1FHDR(18) Weighting Used in Phase Correction

The *QQPWGT* variable specifies the type of weight used in the phase correction process. Phase correction is achieved in the transform programs by shifting the phase of the transformed spectrum to produce a spectrum similar to one from a symmetrical interferogram. The phase cor-

rected spectrum is then recovered as the real (symmetrical) part of the complex spectrum. However, in some cases there are more points on one side of the interferogram center than the other. This type of interferogram is often called single sided. When a single sided interferogram is made symmetric by reflecting it across the center and averaging the two resulting interferograms, points out on the wings will only be represented once, whereas points near the center will be represented twice. To achieve a good estimate of the true symmetrical interferogram, the points on the wings that only occur once must have a weight of 2, whereas the points near the center that occur twice should have weights which add up to 2. The *QQPWGT* variable specifies the type of weight that was used. Because the different weightings can produce differences in the spectra, this variable is included as a reference in comparing spectra produced using different weightings. Its defined values are:

0 : The data is uniformly weighted with 1. This weighting is appropriate for interferograms with data symmetrically sampled about the center (double sided interferograms) or for amplitude spectra for which the phase is of no consequence. Note that amplitude spectra will be most appropriate for symmetrical interferograms or interferograms with no centers at all (noise). For symmetrical interferograms *QQPWGT* = 0 is actually equivalent to *QQPWGT* = 1, given below, but is more efficient in application.

1 : The data is uniformly weighted by 1 about the symmetrical part of the center, 0 on the short side, and 2 on the long side. This is a generally effective weighting and is optimum for noise. It is less efficient when the interferogram is highly dispersed and large at the end of the symmetrical portion. In fact, the error is sinusoidal and proportional to the product of the phase slope and the interferogram value at the weighting function discontinuity. If problems occur when using this function, it should be compared with *QQPWGT* = 2.

2 : The symmetrical portion of the data is weighted with a linear function of 0 at the short side of the symmetrical part of the interferogram, 1 at the center, and 2 at the long side. The rest of the interferogram is weighted with 2. In contrast to *QQPWGT* = 1, this weighting produces an error proportional to the product of the phase slope and a smoothed spectrum. Consequently, it may be better for spectra dominated by strong lines.

2.21 *QQSSUN* I1FHDR(19) Spectral Sampling Units

The *QQSSUN* variable gives the units in which the variable *QQSSMP* is given. Because data can be sampled independently of its units, *QQSSUN* may differ from *QQCXUN*. The defined values for *QQSSUN* are:

0 : Slide centimeters.

1 : Centimeters.

2 : Seconds.

See the discussion under *QQCXUN* for more information on these variables.

2.22 *QQCYUN* I1FHDR(20) Correction Data Y Units

2.23 *QQCXUN* I1FHDR(21) Correction Data X Units

The *QQCYUN* and *QQCXUN* variables have exactly the same meanings as *QQDYUN* and *QQDXUN*, but are only defined for response curves. Response curves are defined as the quotient of the output divided by the input. When the scan data is a response curve, *QQDYUN* and *QQDXUN* will give the output units, and *QQCYUN* and *QQCXUN* will give the input units.

See *QQDXUN, QQCYUN, QQCXUN, and QQUNTS.*

2.24 *QQDYPW* I1FHDR(22) Power of Data Y Units

2.25 *QQCYPW* I1FHDR(23) Power of Correction Y Units

The *QQDYPW* and *QQCYPW* variables give information as to how the units of the data are to be constructed using *QQDYUN* and *QQCYUN*. Their defined values are:

0 : The power of *QQDYUN* or *QQCYUN* is 1.
1 : The power of *QQDYUN* or *QQCYUN* is 2.
2 : The power of *QQDYUN* or *QQCYUN* is 0.

The somewhat arbitrary order of these values is due to their late definition and a desire to maintain compatibility. They are defined to accommodate noise, which in its simplest form is derived from an autocorrelation (*QQDYPW* = 1).

2.26 *QQDINV* I1FHDR(24) Data Inversion

The *QQDINV* variable tells whether or not the data is represented as its inverse. Its primary use is in maximum entropy or linear prediction spectra. The defined values are:

0 : The data is as given. No inversion.
1 : The data is given in inverse form. The true data values are the inverse of the stored data values.

2.27 *QQDRAT* I1FHDR(25) Data Ratio

The *QQDRAT* variable specifies whether the data was derived as a ratio. If it was, its units are given by using both *QQDYUN, QQDXUN* and *QQCYUN, QQCXUN*. The defined values are:

0 : Not a ratio.
1 : Data is a ratio.

2.28 *QQGCOR* I1FHDR(26) Gain Correction Flag

The *QQGCOR* variable is a flag that tells whether data taken by an instrument with several gain settings has been corrected to some standard gain. The significance of the variable is instrument

dependent, and its use, like the gain variables *QQGAN1*, *QQGAN2*, and *QQGAN3*, is optional. The defined values are:

0 : No gain correction or not applicable.
1 : Gain corrected for.

2.29 *QQSCOR* I1FHDR(27) Spectrum Frequency Correction Flag

The *QQSCOR* variable tells if the spectral correction factors *QQCFA0* - *QQCFE1* are defined. See sections 3.37 through 3.46 for the definition of these variables. The defined values are:

0 : The correction factors are not defined.
1 : The correction factors are defined.

2.30 *QQCTYP* I1FHDR(28) Center Definition Type

The *QQCTYP* variable tells how the center was located. Its intended use is to allow automatic recentering of interferograms that are coadded and aligned on the first point. The current values are:

0: The point of maximum deviation from the average was used.
1: The maximum point of the interferogram was used.
2: The minimum point of the interferogram was used.

2.31 *QQISUN* I1FHDR(29) Interferogram Sampling Units

The *QQISUN* variable gives the units of the variable *QQISMP*. Because data can be sampled independently of its units, *QQISUN* may differ from *QQCXUN*. Its defined values are:

0 : Slide centimeters
1 : Centimeters.
2 : Seconds.

Section 2.8, *QQCXUN*, provides additional information on these variables. Centimeter units are unlikely to be used but are included here for compatibility with *QQSSUN*.

2.32 *QQBORD* I1FHDR(30) Byte Order Parameter

The *QQBORD* variable specifies whether the integers have been byte swapped. The defined values are:

0: Integers are not byte swapped.
1: Integers are byte swapped.

2.33 Undefined I1FHDR(31-64)

These variables are reserved for future modifications to the FTS software. User-defined parameters can be written to this space with the understanding that they may conflict with new parameters written by subsequent versions of FTS.

3. 4-BYTE VARIABLES

3.1 *QQFCDY* I4FHDR(1) File Creation day

The *QQFCDY* variable gives the day (counted from 1/1/1) on which the current scan was written to the file. *QQFCDY* is automatically filled in if FTS system subroutines are used to write the file header. The day is found using the subroutines in FTSCALDR.

3.2 *QQFCMS* I4FHDR(2) File Creation Millisecond

The *QQFCMS* variable gives the millisecond in which the current scan was written to the file. *QQFCMS* is automatically filled in if the FTS system subroutines are used to write the file header.

3.3 *QQFLEN* I4FHDR(3) File Length

The *QQFLEN* variable is the number of FTS records (512 bytes) remaining in the scan headed by this header. If *QQFLEN* records are skipped after reading the file header, the file will be positioned so that the next record read will be the next scan header. *QQFLEN* will be automatically filled in if any of the FTS system data storing subroutines are used to store the data.

3.4 *QQDSIZ* I4FHDR(4) Data Size

The *QQDSIZ* variable is the number of data words in the scan. The words may be either Integer*4, Integer*2, or Real. A complex number contains 2 words. *QQDSIZ* will be automatically filled in if any of the FTS system data storing subroutines are used to store the data.

3.5 *QQDSCL* I4FHDR(5) Data Scaling

The *QQDSCL* variable is the power of 2 scaling factor for data stored in one of the integer formats. The true value of a number stored in Integer*2 format is given by stored-value x $2^{(QQDSCL-15)}$. Likewise, the true value of a number stored in Integer*4 format is given by stored-value x $2^{(QQDSCL-31)}$. The values retrieved from a scan using the FTS system subroutines should be assumed scaled according to the retrieved data type, not the data type stored in the scan. If the retrieved data type is real this variable may be ignored because the values retrieved into a REAL array will automatically have the correct scaled values. *QQDSCL* will be automatically filled in if any of the FTS system data storing routines are used to store data . Data stored in a real format cannot be retrieved into an integer format because it is stored unscaled and *QQDSCL* is undefined.

3.6 *QQSRCF* I4FHDR(6) Source Frame

The *QQSRCF* variable is the location of a particular scan within a data taking run. For example, when ten interferograms are collected during ten consecutive scans of an interferometer and put in a single file, the individual scans should be numbered from 1 to 10. these numbers are assigned to the variable *QQSRCF*. When scans are coadded , this variable becomes undefined unless the coadding is part of the original data acquisition.

3.7 *QQIBDY* I4FHDR(7) Begin Day

The *QQIBDY* variable is the day (counted from 1/1/1) on which the first data point in a scan was taken. The day can be obtained from the normal date format through the use of the FTSCALDR subroutines. Ideally, *QQIBDY* should be obtained from some standard date, such as GMT. When scans are coadded, either before or after transforming, *QQIBDY* should be set to the smallest value in the coadded set of values. See Coadding, FTSCALDR.

3.8 *QQIBMS* I4FHDR(8) Begin Millisecond

The *QQIBMS* variable is the millisecond in which the first data point in a scan was taken. Ideally, *QQIBMS* should be obtained from some standard time, such as GMT. The millisecond is counted from 12:00 PM, with 12:00 PM having a value of 0. FTSCALDR subroutines ,can be used to convert normal time formats to elapsed milliseconds. When scans are coadded, either before or after transforming, *QQIBMS* should be set to the value associated with the earliest scan. See Coadding, FTSCALDR.

3.9 *QQIEDY* I4FHDR(9) End Day

The *QQIEDY* variable is the day (counted from 1/1/1) on which the last data point in a scan was taken. The day should be obtained from the normal date format through the FTSCALDR subroutines. Ideally, *QQIBDY* should be obtained from some standard date, such as GMT. When scans are coadded, either before or after transforming, *QQIBDY* should be set to the largest value in the coadded set of values. See Coadding, FTSCALDR.

3.10 *QQIEMS* I4FHDR(10) End Millisecond

The *QQIEMS* variable is the millisecond on which the last data point in a scan was taken. Ideally *QQIBMS* should be obtained from some standard time, such as GMT. The millisecond is counted from 12:00 PM, with 12:00 PM having a value of 0. FTSCALDR subroutines can be used to convert normal time formats to elapsed milliseconds. When scans are coadded, either before or after transforming, *QQIBMS* should be set to the value associated with the latest scan. See Coadding, FTSCALDR.

3.11 *QQIDUR* I4FHDR(11) Duration Milliseconds

The *QQIDUR* variable is the number of milliseconds over which data was taken. For a single scan *QQIDUR* is the difference between the beginning and ending times. When a portion of an interferogram is used in a transform, *QQIDUR* should be updated to reflect the (estimated) duration of the data used. When scans are coadded, either before or after transforming, *QQIDUR* should be replaced with the sum of individual *QQIDUR* values for the coadded scans. See Coadding.

3.12 *QQADDS* I4FHDR(12) Number of Coadded Scans

The *QQADDS* variable is the number of original scans that have been coadded to obtain the current scan. This number is 1 for the original instrument scans. When scans are coadded, this number is replaced by the sum of the *QQADDS* values of the coadded scans. See Coadding.

3.13 *QQISIZ* I4FHDR(13) Original Interferogram Size

The *QQISIZ* variable is the number of counts in the original (not resampled) interferogram. Counts are defined as the smallest sampling interval which could have been used; i.e., for a single scan *QQISIZ* is the product of number of samples and *QQCDWN*. *QQISIZ* serves as a record of the original interferogram and as such can be used to determine the location of a resampled interferogram in the original sequence of points. This is useful in estimating times at the beginning and end of the resampled data. *QQISIZ* may be defined for coadded interferograms, but it seems to lose much of its usefulness for coadded spectra. Note that a frequency scale could be derived for spectra by dividing the time of the scan by *QQISIZ* to get the time equivalent of *QQISMP*.

3.14 *QQADBT* I4FHDR(14) Number of A/D Bits

The *QQADBT* variable is the number of bits in the A/D converter used to acquire the data. It is used to convert data into volts and to measure of data accuracy.

3.15 *QQADRG* I4FHDR(15) A/D Range

The *QQADRG* variable is the A/D minimum to maximum range in micro-volts. For example, if the A/D accepts voltages from -10 to +10 volts, *QQADRG* = 20×10^6.

3.16 *QQADSB* I4FHDR(16) A/D Voltage Offset

The *QQADSB* variable is the voltage in micro-volts subtracted from the signal before it entered the A/D. This number is useful in systems where the voltage was offset before it was sampled. It is most significant in DC coupled systems where the true DC voltage may have physical significance. For example, if 5 volts is subtracted from a DC signal to sample it with an A/D ranging from -5 to 5 volts, then *QQADSB* = 5×10^6.

3.17 *QQISMP* I4FHDR(17) Distance between Counts

The *QQISMP* variable is the distance in pico-meters or pico-seconds between counts. Counts are the smallest interval at which data can be sampled. By present convention, if the displacement of an interferometer wedge or mirror is measured using laser fringes, then a count would represent one wavelength of the laser light. For example, if the mirror displacement of a Michelson interferometer is measured using the fringes produced by shining a red HeNe laser through the system, then *QQISMP* = 632800.

3.18 *QQISLP* I4FHDR(18) Counts per Sample

The *QQISLP* variable is the number of counts between data samples. In the original data *QQISLP* will be equal to *QQCDWN*. However, if the data is resampled, *QQISLP* must be modified to reflect the new sampling. For example, if data taken with a countdown of 5 is digitally filtered so that only data in a small band is retained, and then the data is resampled at every 40th original sample to take advantage of the resulting small bandwidth, $QQISLP = 5 \times 40 = 200$.

QQISLP is updated to reflect the interferogram portion used in a transform.

3.19 *QQIOFF* I4FHDR(19) Counts Offset

The *QQIOFF* variable is the number of counts offset into the original interferogram of the current interferogram. This number is 0 for the original interferogram, i.e. the first original sample has an offset of 0. It is of use in those cases where the interferogram is resampled and it is desirable to keep track of where the new data starts in order to estimate the time at which the new interferogram starts.

QQIOFF is updated to reflect the interferogram portion used in a transform.

3.20 *QQILEN* I4FHDR(20) Interferogram Length

The *QQILEN* variable is the number of samples in the current, possibly resampled, interferogram. This number differs from *QQISIZ* both in units (samples instead of counts) and in the fact that *QQISIZ* refers to the original interferogram.

QQILEN is updated to reflect the interferogram portion used in a transform.

3.21 *QQICEN* I4FHDR(21) Interferogram Center

The *QQICEN* variable is the number of samples into the current interferogram where the center occurs. *QQICEN* is 0 if the first sample is the interferogram center. The interferogram center will usually be defined as the point at which the interferogram has its maximum absolute deviation from its average.

QQICEN is updated to reflect the interferogram center used in a transform.

3.22 *QQIMOD* I4FHDR(22) Interferogram Modulation

The *QQIMOD* variable is the maximum absolute deviation of the interferogram from the interferogram average. The maximum may be searched for over a subrange of the interferogram data points to avoid electrical transients. *QQIMOD* is intended to supply qualitative information about the interferogram, such as a sudden change in value due to changes in the source or to failure of the instrument. To avoid carrying scaling information., this number is given in ARBitrary units; i.e., the number which comes directly from the A/D.

3.23 *QQISUB* I4FHDR(23) Interferogram Offset

The *QQISUB* variable is the number which may have been subtracted from the original A/D data in order to produce 2's complement numbers. This number is in ARBitrary units. For example, is a 16 bit A/D measures voltages from 0 to 10 and produces numbers ranging from 0 to 65535, 32768 is subtracted from each number to produce 2's complement Integer*2 numbers, and *QQISUB* = 32768.

3.24 *QQIAVG* I4FHDR(24) Interferogram Average

The *QQIAVG* variable is the average value of the interferogram in ARBitrary units after *QQISUB* has been subtracted. This number, like *QQIMOD*, is intended to supply qualitative information.

3.25 *QQTBEG* I4FHDR(25) Transform Beginning

The *QQTBEG* variable is the sample at which the data used in a transform begins. This number is independent of any amount of zero fill which may be added. It is recorded as an aid in tracking down the cause of oddities in transformed spectra and in estimating the times covered by the interferogram data.

3.26 *QQTLEN* I4FHDR(26) Transform Data Length

The *QQTLEN* variable is the number of interferogram samples used in a transform. This number is independent of any amount of zero fill which may be added. It may be used in conjunction with *QQTBEG* to determine which interferogram points were used in a transform and to estimate the time at which the last used data point was taken. *QQTLEN* differs from *QQDSIZ* in that it may refer to complex data points, each of which would contains two data words.

3.27 *QQTCEN* I4FHDR(27) Transform Center

The *QQTCEN* variable is the center location in samples used by the transform. If the first sample was used as the center, *QQTCEN* would have the value 0. This number may differ from *QQICEN*. It will have the most significance in phase corrected spectra.

3.28 *QQTSIZ* I4FHDR(28) Transform Size

The *QQTSIZ* variable is the size of the transform in samples. This number includes zero fill and will generally be a power of 2.

3.29 *QQRESL* I4FHDR(29) Transform Resolution

The *QQRESL* variable is the number of sample points that are effective in determining the transform resolution. This number is dependent on the center location relative to the ends of the data used in transform and phase correction processes, but is independent of the amount of zero-fill

and the apodization. The maximum FWHM resolution of a spectral line in the spectrum is given by :

$$FWHM = Factor/(QQRESL \times QQISLP \times QQISMP \times 10^{-10}) \text{ cm}^{-1}$$

or

$$FWHM = Factor/(QQRESL \times QQISLP \times QQISMP \times 10^{-12}) \text{ hz}$$

The first formula is used if *QQISMP* equals 0 or 1, and the second is used if it equals 2. Factor depends on the apodization; its values can be found in the documentation in FTSAPOD. The units of FWHM are the inverse of the *QQISMP* units. *QQISMP* is used for the unit of length instead of *QQSSMP*, because a spectrum may be resampled in a nonlinear way that makes it impossible for a single fixed number to define the resolution. See Section 2.18, *QQAPOD*.

3.30 *QQPHSZ* I4FHDR(30) Phase Points

The *QQPHSZ* variable describes the number of samples used to determine a phase to be used for phase correction. This number is significant only if the phase is determined from the transformed interferogram. It is intended mainly as record of how the interferogram was processed if something goes wrong or the validity of the phase correction is questioned.

3.31 *QQAPAR* I4FHDR(31) Apodization Parameter

The *QQAPAR* variable is currently undefined. It was originally intended to store a parameter that would determine which parameter set of apodizations (such as Kaiser-Bessel) was used to apodize the interferogram data.

3.32 *QQSSMP* I4FHDR(32) Spectral Sample Size

The *QQSSMP* variable is the fundamental sample size, in pico-meters or pico-seconds, in which the spectrum parameters are given. This parameter is provided for use when the spectrum is resampled. This could occur when a spectrum, produced by a field widened interferometer and originally in slide cm^{-1} units that were determined by the transform size, was resampled in terms of wavenumbers at some fixed interval. The units are determined by *QQSSUN*. It should initially be set to the same value as *QQISMP*.

3.33 *QQSTSZ* I4FHDR(33) Spectral Transform Size

The *QQSTSZ* variable is the value of the *QQSSMP* variable that would have been used in a transform to produce a given Bin resolution. The spectrum is sampled in multiples of Bins, with the size of a Bin equal to:

$$Bin = 1/(QQSTSZ \times QQSSMP \times 10^{-10}) \text{ cm}^{-1}$$

or

$$Bin = 1/(QQSTSZ \times QQSSMP \times 10^{-12}) \text{ Hz}$$

The first formula is used if *QQSSUN* equals 0 or 1, and the second is used if it equals 2. The units are the inverse of the units used for *QQSSMP*. A Bin plays the same role for spectra as a count does for interferograms, but is somewhat more flexible in that *QQSTSZ* can be defined for a fictional interferogram; i.e., *QQSTSZ* and *QQSSMP* can be manipulated to produce any convenient value for Bin as long as *QQISLP* and *QQSOFF* are adjusted to give the correct location of the spectral samples. *QQSTSZ* will generally start off equal to *QQTSIZ* * *QQISLP* unless manipulation occurs. The value of Bin is the spectral equivalent of *QQISMP*.

3.34 *QQSSLP* I4FHDR(34) **Spectral Sampling Slope**

The *QQSSLP* variable is the number of Bins per spectral sample in the current scan. *QQSSLP* is included mainly as an aid to track resampled data. It will generally be set to 1 after a transform unless the value of Bin has been manipulated. It is analogous to *QQISLP*.

3.35 *QQSOFF* I4FHDR(35) **Spectral Sample Offset**

The *QQSOFF* variable is the number of Bins offset from 0 at which the current spectral samples begin. A frequency of 0 is taken as an absolute reference point for spectra in the same way as the first point of an interferogram is taken as an absolute point. *QQSOFF* is intended to be used in resampling spectra, particularly if only a part of the spectrum contains significant data. It is analogous to *QQIOFF*.

3.36 *QQSLEN* I4FHDR(36) **Spectrum Length**

The *QQSLEN* variable is the number of samples in the current spectrum. *QQSLEN*, in conjunction with *QQSSLP* and *QQSOFF*, determines the range of spectral data. It is analogous to *QQILEN*. *QQSLEN* differs from *QQDSIZ* in that it may refer to complex data points, each of which contains two data words.

3.37 *QQCFA0*	I4FHDR(37)	**Correction Parameter A Mantissa**
3.38 *QQCFA1*	I4FHDR(38)	**Correction Parameter A Exponent**
3.39 *QQCFB0*	I4FHDR(39)	**Correction Parameter B Mantissa**
3.40 *QQCFB1*	I4FHDR(40)	**Correction Parameter B Exponent**
3.41 *QQCFC0*	I4FHDR(41)	**Correction Parameter C Mantissa**
3.42 *QQCFC1*	I4FHDR(42)	**Correction Parameter C Exponent**
3.43 *QQCFD0*	I4FHDR(43)	**Correction Parameter D Mantissa**
3.44 *QQCFD1*	I4FHDR(44)	**Correction Parameter D Exponent**
3.45 *QQCFE0*	I4FHDR(45)	**Correction Parameter E Mantissa**
3.46 *QQCFE1*	I4FHDR(46)	**Correction Parameter E Exponent**

These ten variables may be considered as a unit. They are intended to reflect the nonlinear relationship that may exist between the units used in the sampled spectrum (defined by *QQSSUN*) and the units used to define the spectral resolution (defined by *QQDXUN*). They are also carried as an aid in synthesizing spectra to match the resampled spectra. This approach is not recom-

mended, however, because it is generally easier to synthesize a spectra that matched the original spectrum. If the spectral calibration is considered sufficiently fixed or known for a given spectra, these ten variables should be filled in, even if the data isn't resampled, so that they may be used for plotting the data on a corrected (true wavenumber) axis, if desired. If these variables are present, *QQSCOR* should be set to 1; otherwise *QQSCOR* is set to 0. They are to be interpreted as follows:

$$V_s = V_c*[QQA + QQB/V_c^2 + QQC/V_c^4 + QQD*V_c^2 + QQE*V_c^4]$$

Where

V_s	= Spectral frequency in inverse slide centimeters
V_c	= Spectral frequency in inverse (true) centimeters
QQA	= QQFCA0 x 2^{QQFCA1}
QQB	= QQFCB0 x 2^{QQFCB1}
QQC	= QQFCC0 x 2^{QQFCC1}
QQD	= QQFCD0 x 2^{QQFCD1}
QQE	= QQFCE0 x 2^{QQFCE1}

Once V_s is known it can be used to determine an index for the uncorrected spectrum. Likewise, the spectral resolution in true wavenumbers can be obtained by the formula:

$$FWHM_c = FWHM_s * dV_c/dV_s$$

The derivative in the preceding formula can be most easily calculated using implicit differentiation of the formula for V_s. Both the plotting of uncorrected data on a corrected scale and the determination of the corrected resolution are most conveniently performed starting with known values of V_c. This explains the 'inverted' form of the formula relating V_s and V_c.

NOTE: if a linear correction to the scale is required, it is more efficient to revise *QQSSMP* and change *QQSSUN* than to use these variables. These variables are obsolete in practice.

3.47 *QQALIA* I4FHDR(47) Alias Number

The *QQALIA* variable is the alias number of the spectrum or interferogram. Normal interferograms have an alias number of 0, the first alias 1, etc. The alias number of a given frequency can be obtained by dividing the given frequency by the Nyquist frequency and truncating. All of the frequencies in a spectrum must have the same alias number except the largest frequency, which may have an alias number which is 1 greater.

3.48 Undefined I4FHDR(48-64)

These variables are reserved for future modifications to the FTS software. User-defined parameters can be written to this space with the understanding that they may conflict with new parameters written by subsequent versions of FTS.

3.49 Undefined USER(1-32)

These variables are reserved for user-defined parameters.

Table 1. FTS HEADER VARIABLE DEFINITIONS

HEADER VARIABLE	ARRAY VARIABLE	BYTE NUMBER	PARAMETER NAME
QQDSRC	FHDR(1..32)	1 - 32	Data Source
QQRMRK	FHDR(33..64)	33 - 64	Remark
QQSYSV	I1FHDR(1)	65	FTS System Version Used
QQDTYP	I1FHDR(2)	66	Scan Data Word Type
QQDKND	I1FHDR(3)	67	Scan Data Kind
QQDREP	I1FHDR(4)	68	Scan Data Representation
QQDYUN	I1FHDR(5)	69	Scan Data Y Units
QQDXUN	I1FHDR(6)	70	Scan Data X Units
QQDEVC	I1FHDR(7)	71	Device Code
QQMOTN	I1FHDR(8)	72	Slide Direction
QQCDWN	I1FHDR(9)	73	Count Down
QQGAN1	I1FHDR(10)	74	Gain 1
QQGAN2	I1FHDR(11)	75	Gain 2
QQGAN3	I1FHDR(12)	76	Gain 3
QQTTYP	I1FHDR(13)	77	Transform Type
QQTSFT	I1FHDR(14)	78	Rescaling Shifts in Integer Transforms
QQPHCR	I1FHDR(15)	79	Phase Correction Flag
QQAPOD	I1FHDR(16)	80	Apodization Type
QQNTRP	I1FHDR(17)	81	Interpolation Zero Fill
QQPWGT	I1FHDR(18)	82	Weighting Used in Phase Correction
QQSSUN	I1FHDR(19)	83	Spectral Sampling Units
QQCYUN	I1FHDR(20)	84	Correction Data Y Units
QQCXUN	I1FHDR(21)	85	Correction Data X Units
QQDYPW	I1FHDR(22)	86	Power of Data Y Units
QQCYPW	I1FHDR(23)	87	Power of Correction Y Units
QQDINV	I1FHDR(24)	88	Data Inversion ?
QQDRAT	I1FHDR(25)	89	Data Ratio ?
QQGCOR	I1FHDR(26)	90	Gain Correction Flag
QQSCOR	I1FHDR(27)	91	Spectrum Frequency Correction Flag
QQCTYP	I1FHDR(28)	92	Center Definition Type
QQISUN	I1FHDR(29)	93	Interferogram Sampling Units

Table 1. FTS HEADER VARIABLE DEFINITIONS (Continued)

HEADER VARIABLE	ARRAY VARIABLE	BYTE NUMBER	PARAMETER NAME
QQBORD	I1FHDR(30)	94	Byte Order Parameter
Undefined	I1FHDR(31)	95	Reserved for Future Use*
Undefined	I1FHDR(32)	96	Reserved for Future Use
Undefined	I1FHDR(33)	97	Reserved for Future Use
Undefined	I1FHDR(34)	98	Reserved for Future Use
Undefined	I1FHDR(35)	99	Reserved for Future Use
Undefined	I1FHDR(36)	100	Reserved for Future Use
Undefined	I1FHDR(37)	101	Reserved for Future Use
Undefined	I1FHDR(38)	102	Reserved for Future Use
Undefined	I1FHDR(39)	103	Reserved for Future Use
Undefined	I1FHDR(40)	104	Reserved for Future Use
Undefined	I1FHDR(41)	105	Reserved for Future Use
Undefined	I1FHDR(42)	106	Reserved for Future Use
Undefined	I1FHDR(43)	107	Reserved for Future Use
Undefined	I1FHDR(44)	108	Reserved for Future Use
Undefined	I1FHDR(45)	109	Reserved for Future Use
Undefined	I1FHDR(46)	110	Reserved for Future Use
Undefined	I1FHDR(47)	111	Reserved for Future Use
Undefined	I1FHDR(48)	112	Reserved for Future Use
Undefined	I1FHDR(49)	113	Reserved for Future Use
Undefined	I1FHDR(50)	114	Reserved for Future Use
Undefined	I1FHDR(51)	115	Reserved for Future Use
Undefined	I1FHDR(52)	116	Reserved for Future Use
Undefined	I1FHDR(53)	117	Reserved for Future Use
Undefined	I1FHDR(54)	118	Reserved for Future Use
Undefined	I1FHDR(55)	119	Reserved for Future Use
Undefined	I1FHDR(56)	120	Reserved for Future Use
Undefined	I1FHDR(57)	121	Reserved for Future Use
Undefined	I1FHDR(58)	122	Reserved for Future Use
Undefined	I1FHDR(59)	123	Reserved for Future Use
Undefined	I1FHDR(60)	124	Reserved for Future Use
Undefined	I1FHDR(61)	125	Reserved for Future Use

Table 1. FTS HEADER VARIABLE DEFINITIONS (Continued)

HEADER VARIABLE	ARRAY VARIABLE	BYTE NUMBER	PARAMETER NAME
Undefined	I4FHDR(62)	373	Reserved for Future Use
Undefined	I4FHDR(63)	377	Reserved for Future Use
Undefined	I4FHDR(64)	381	Reserved for Future Use
Undefined	USER(1)	385	User Defined
Undefined	USER(2)	389	User Defined
Undefined	USER(3)	393	User Defined
Undefined	USER(4)	397	User Defined
Undefined	USER(5)	401	User Defined
Undefined	USER(6)	405	User Defined
Undefined	USER(7)	409	User Defined
Undefined	USER(8)	413	User Defined
Undefined	USER(9)	417	User Defined
Undefined	USER(10)	421	User Defined
Undefined	USER(11)	425	User Defined
Undefined	USER(12)	429	User Defined
Undefined	USER(13)	433	User Defined
Undefined	USER(14)	437	User Defined
Undefined	USER(15)	441	User Defined
Undefined	USER(16)	445	User Defined
Undefined	USER(17)	449	User Defined
Undefined	USER(18)	453	User Defined
Undefined	USER(19)	457	User Defined
Undefined	USER(20)	461	User Defined
Undefined	USER(21)	465	User Defined
Undefined	USER(22)	469	User Defined
Undefined	USER(23)	473	User Defined
Undefined	USER(24)	477	User Defined
Undefined	USER(25)	481	User Defined
Undefined	USER(26)	485	User Defined
Undefined	USER(27)	489	User Defined
Undefined	USER(28)	493	User Defined
Undefined	USER(29)	497	User Defined

Table 1. FTS HEADER VARIABLE DEFINITIONS (Continued)

HEADER VARIABLE	ARRAY VARIABLE	BYTE NUMBER	PARAMETER NAME
Undefined	USER(30)	501	User Defined
Undefined	USER(31)	505	User Defined
Undefined	USER(32)	509	User Defined

* To be used for future modifications or user-specified parameters.

Appendix 2

Control file format

The control file is an ASCII file consisting of several lines. Each line contains either information to be inserted into the FTS header to identify the experiment and location of the data acquisition; to direct the output file to a specific path; to control the number of interferograms to be written to a file before a new file is opened; and to pause or terminate data acquisition.

<Remark> :Up to 32-character field inserted into the FTS header

<Latitude > :Integer = the latitude of measurements in arc-seconds (+=N)

<Longitude> :Integer = longitude of measurements in arc-seconds (+=W)

<Elevation> :Integer = metres above sea level

<Time> :Flag (spare) to select time source, set to 0 = DOS time

<Max IFG> :Maximum number of interferograms (scan-pairs) written to file before opening new file

<IFG Path> :Path for the interferogram data file

<FFT Path> :Path for transformed interferograms file, Not used, set to 0

<IFG Plot> :Flag (spare) to plot interferograms. Not used, set to 0 = No Plot

<FFT Plot> :Flag (spare) to plot transformed interferograms, set to 0 = No Plot

<Start time 1> <End time 1> <Number of added interferograms per write>

<Start time 2> <End time 2> <Number of added interferograms per write>

...

<Start time 3> <End time 3> <Number of added interferograms per write>

Times are given as HH:MM, and the number of interferograms (scan-pairs) to be added coherently during the time interval from <Start Time> to <End Time> is an integer. If this number is set to 0, the programme will pause data acquisition until the next active time interval. If the number is negative, the programme will exit when it reaches that time interval.

Appendix 3

Acquisition software documentation (Thesis)

REAL-TIME INTERFEROMETER DATA ACQUISITION USING 80386 IN PROTECTED MODE

by

Balakumar Nagarajan

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Gardiner S. Stiles
Major Professor

Patrick J. Espy
Committee Member

Scott E. Budge
Committee Member

James P. Shaver
Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

1992

ACKNOWLEDGMENTS

This research was funded by the National Science Foundation (NSF) through the Space Dynamics Lab (SDL).

I would like to express my sincere acknowledgment to Dr. Gardiner Stiles, my major professor, for the support and motivation he gave me throughout the course of my research.

I am indebted to Dr. Patrick J. Espy, principal investigator, for having given me a chance to work on this project and for his guidance, without which the research would not have been a success.

Thanks are due to Dr. Scott E. Budge for his critical review of this report and for his suggestions.

I thank Charles Harris for his suggestions and help in making the program faster and more modular, and for teaching me many tricks of C programming.

It has been a pleasure to work with SDL, and I truly appreciate the support of the SDL staff.

I am grateful to my friends for their constant encouragement and criticism throughout my study here.

Balakumar Nagarajan

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Real-Time Interferometer Data Acquisition Using 80386 in the Protected Mode

by

Balakumar Nagarajan, Master of Science

Utah State University, 1992

Major Professor: Dr. Gardiner S. Stiles
Department: Electrical Engineering

This thesis is a description of a method to perform a fast real-time data acquisition of large Interferometer data using 80386 in the protected mode. The intent of this research was to enable the Space Dynamics Lab (SDL) at Utah State University, which is involved in research related to atmospheric science, 1) to continuously acquire large amounts of data from a Michelson interferometer through an IBM PC-AT, 2) to do a real-time coaddition of interferograms, and 3) to automatically acquire and analyze data that would allow SDL to make accurate and systematic measurements of the integrated band radiances and rotational temperatures of the NIR OH Meinel bands in the twilight and nightglow using integration times less than or equal to 3 minutes. This entailed 1) developing an acquisition software interface between Michelson interferometer and IBM PC-AT that would run under 80386 protected mode, 2) interfacing the acquisition software with the Fourier Spectrometer software used by SDL, 3) developing a real-time coaddition of interferograms, and 4) developing a scheduler to automate the acquisition and analyzing process.

The interferometer provides data to the PC through DMA, which transfers the data to the conventional memory (Lower 640kB). It also interrupts the processor after transferring a set of data called a half-scan. Interrupts can be handled only in real mode. The data from conventional memory should be transferred to the extended memory and the hardware interrupt should be handled directly from the protected mode in order to process the data using the available software that runs in protected mode. A scheduler was developed to start and stop the data acquisition at specific Solar Depression Angles (SDA), to add data from the scans in the same direction, and to store the data in FTS format. The validity of the data was checked by simultaneously acquiring data from an established instrument and comparing the two data sets. The performance was compared with that of the existing real-mode acquisition software. Results indicated that there was a 40% increase in the signal to noise ratio, and the software was rugged and fast enough not to miss any scans at the fastest resolution.

(74 pages)

CHAPTER I

INTRODUCTION

The mesosphere and lower thermosphere (MALT) play a strategic role in the coupling of atmospheric regions. Long-term measurements of the intensities and rotational temperatures of the bright emissions of OH and O_2 from geographically distinct locations have been recommended by the National Science Foundation CEDAR (Coupling, Energetics and Dynamics of Atmospheric Regions) program in order to investigate the dynamics, chemistry, and thermal structure of the MALT [6]. The ideal instrument for these near infrared (NIR) measurements should operate over a broad spectral range so that multiple emission bands and atomic lines can be investigated simultaneously. The Bomem 100 series Michelson interferometer, because of its low cost, ruggedness, stability, alignment tolerance, and throughput advantages, is well suited for this task. In this thesis we develop a real-time software package to acquire an interferogram from the interferometer, perform coaddition and spectral analysis, display the interferogram and its spectra, and store the interferogram and its spectra.

A. Data Acquisition Overview

Computerized data acquisition systems have become an integral part of test and research throughout industry and education programs [8] in the past decade. Data acquisition is a restricted kind of data collection [2] that has been a topic of interest mainly to scientists and engineers. Most of the data we wish to acquire is contained in *physical variables*, such as temperature, light intensity, and velocity. Special measuring devices, called *sensors*, are used to measure the values of these variables; the outputs

of these measuring devices are converted into a form suitable for computer processing. A typical data acquisition system is shown in Fig. 1.

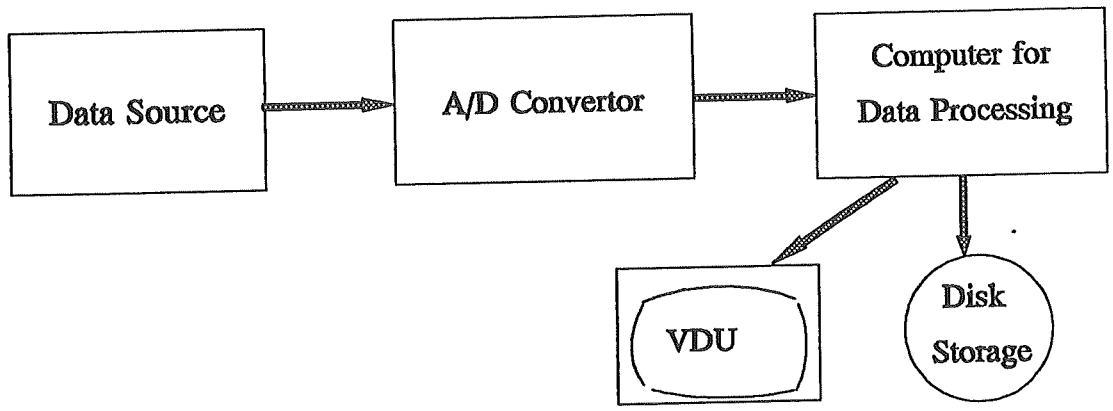


Fig. 1. Basic data acquisition system

Advances in digital and integrated-circuit technology have increased the speed requirements of data acquisition systems. The need to maximize data acquisition is one of the challenges that faces researchers. To do so data storage requirements must be minimized, since storage space is expensive [12]. This requires the data to be processed and analyzed as it is acquired i.e., in real time. Many applications require real-time measurements and operations on the data such as filtering and plotting. It is possible to satisfy this requirement by using multitask computers [13], [5] that can run the data collection and data analysis simultaneously. This solution, however, has several drawbacks: high costs, difficulties in using and programming under the more complicated operating systems, and reduced reliability. Hence low-cost PCs are routinely used for this purpose. Due to the low hardware costs incurred with a PC system, it is possible

for the user to create sophisticated, dedicated, customized systems. The commercially available, low-cost interferometer from Bomem, Inc. uses a standard IBM PC for a data system.

We use an IBM PC/AT with an Intel 80386 processor and 4 MegaBytes (MB) of physical memory, running Microsoft's Disk Operating System (MS-DOS). The upper limit of memory requirements for capturing the interferograms and performing a spectral analysis on them is on the order of 1 MB. The 80386 processor has the capability of addressing 4 GigaByte (GB) of physical memory. The only way an 80386 can address memory locations beyond the 1 MB boundary is by switching the 80386 to operate in the protected mode. So for our memory requirements the 80386 has to operate in the protected mode. This does not entirely solve our problem, since application programs have access only to the first 640 KiloByte (KB) of memory, for code and for their data, when running under MS-DOS. Chapter III covers this problem, and the solution adopted to overcome it, in detail.

B. Background and Purpose

The Michelson interferometer had been a powerful remote sensing tool for studying the atmospheric regions by sensing and analyzing the light emanated from or passing through these regions. The technological advances in solid-state detectors and Fourier transform software have extended remote sensing of the atmosphere into the infrared. The conventional Michelson interferometer is limited to a narrow field of view. However, the recently available thermo-electrically cooled indium-gallium-arsenide (InGaAs) detectors, which are as sensitive as that of intrinsic germanium for the 840 to

However, the recently available thermo-electrically cooled indium-gallium-arsenide (InGaAs) detectors, which are as sensitive as that of intrinsic germanium for the 840 to 1640 nm region, increase the probability of successfully making short time frame measurements with the smaller-aperture, commercial instruments without the aid of field-widening [6]. This new technology, when incorporated into small, low-cost, commercially available interferometer systems, may be useful for atmospheric measurements.

Most commercially available interferometers are designed for laboratory analysis. Hence the software systems are designed for single measurements such as transmittance and absorbance. In addition, most commercial interferometers come with their own custom, computational hardware. These non-standard computer systems are not easy to adapt or program for long-term atmospheric measurements. The new interferometer from Bomem is among the first to use a simple interface to a general purpose computer which may be programmed for atmospheric measurements.

The Space Dynamics Laboratory (SDL) at Utah State University (USU) has been involved in atmospheric research for the past two decades and has developed hardware and a great deal of software for atmospheric measurements and for analyzing atmospheric data acquired with an interferometer. Although Bomem marketed acquisition software with their interferometer to interface it with the PC, it had several drawbacks:

1) It is designed to do a single experiment and stop,

2) It does not write data in DOS format, therefore the only access to the data is through the Bomem software,

3) It is not set up to fit synthetic spectra and derive radiance and temperatures; i.e.,

could not be used on the data written out by Bomem software. Also, the software used by SDL expected the data to be in a format called the Fourier transform system (FTS). This software was developed to run under the 80386 protected mode in order to use the vast memory accessing capabilty of 80386, such that large volumes of data could be used for better analysis.

The purpose of this thesis was

1) To interface the Michelson interferometer with the IBM PC-AT, by developing acquisition software that would run under 80386 protected mode.

2) To interface this acquisition software with the Fourier spectrometer software used by SDL.

3) To implement real-time coaddition of interferograms according to a scheduler.

4) To automate the acquisition and analysis of data that would allow us to make accurate, systematic measurements of the integrated band radiances and rotational temperatures of the NIR OH Meinel bands in the twilight and nightglow using integration times less than or equal to 3 minutes.

C. Layout of this Thesis

The rest of this thesis is organized as follows. In Chapter II we present a brief description of the data acquisition hardware used along with a summary of acquisition and real-time processing problems encountered when using a PC running under DOS. A detailed description of the operation of the 80386 processor in protected mode is

and real-time processing problems encountered when using a PC running under DOS. A detailed description of the operation of the 80386 processor in protected mode is presented in Chapter III. Interrupt handling is one of the more difficult topics to understand in any data processing text; Chapter IV deals with the problems of interrupt handling under protected mode and solutions adopted to solve them along with the design and implementations of the various interfaces developed. Tests and results are presented in Chapter V and concluding remarks and recommendations are given in Chapter VI.

CHAPTER II

REAL-TIME DATA ACQUISITION

Data acquisition systems are used to electronically monitor or gather data from the external physical environment. These systems differ from normal data processing systems in that the data is typically derived from transducers, sensors, and other devices automatically, or at least semiautomatically [3]. Data acquisition systems are designed to interface a computer to the correct equipment in order to collect data from external sources. In this chapter, the principle of operation of the Michelson interferometer along with the hardware detail of the interferometer used in this project is presented. This chapter also deals with the specifications required to be met by data acquisition software, a description of real time processing of the acquired data, and the limitations of DOS.

A. Data Acquisition

The first step in computerized data acquisition involves the measurement of some physical parameter whose value has been converted into an analog electric voltage using a transducer. This voltage is then digitized by the process of sampling and quantization using an analog-to-digital (A/D) convertor. However, either before or after digitization, there may be the need to perform some pre-processing [2], [3] in order to achieve data reduction and signal improvement. This may include operations such as analog filtering or digital filtering. The data may be stored in analog form or in digitized form, in which case the storage medium will be one of the common computer storage devices. The data may then be processed in the same computer or transferred to another computer for

central processing. The advantage of using such a data acquisition system is that the measuring device can be controlled directly by a suitable program, which can choose when, what, and how much data are to be collected, depending on various operational conditions. A great deal of human intervention can thus be eliminated, and full automation of data acquisition and processing becomes feasible. Allied to this is the possibility of *real-time* data processing and control.

In many data acquisition problems, the signal source is located at some distance from the observer. In such cases the measuring device must be located away from the signal source. This type of data acquisition is called *remote sensing*. Remote sensing may be either passive or active [2]. In passive remote sensing the measuring devices merely receive signals produced by the data sources; in the active case, the system sends out its own signal and measures signals returned from the objects under observation.

All objects emit *infrared* radiation to varying degrees depending on their temperature. The rays provide useful information about both the composition of an object and its thermal state. Spectrometric techniques, a form of passive remote sensing, are usefully employed when observations are carried out in the infra-red range. The emissions from an individual data source are received by an interferometer or grating and the spectrum of the emissions is thus obtained. Because the spectrum is determined by the chemical composition and the temperature of an object, these parameters may be remotely sensed by measuring its spectrum. In this project we use a Michelson interferometer (MI) since the signal-to-noise ratio (S/N), which is quite high for the MI under photon noise limited conditions [6], is even greater in the IR range where the

higher receiver noise is the limiting factor in weak airglow auroral signal detection (Fig. 2).

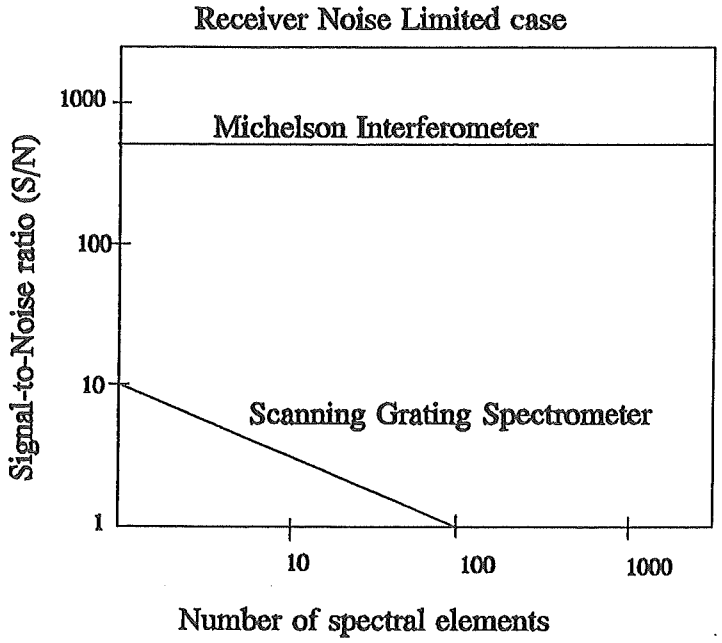


Fig. 2. S/N ratio comparison

B. Interferometer: Operation and Hardware Detail

This section deals with the principles of operation of the interferometer used in this project to measure the intensity and rotational temperature of the bright emission of OH from MALT. The optical layout of a double beam interferometer is shown in Fig. 3. The basic building block of the unit is a Michelson-type modulator which provides a method of coding spectral elements of the incoming optical energy from a source. The components of a Michelson modulator, in general, include [9] a beamsplitter-compensator, a translating mirror, and an adjustable fixed mirror. The Bomem interferometer, on the other hand, uses two translating roof-type retroreflectors, as shown in Fig. 3. This design allows for a more compact and stable interferometer design since the beam is

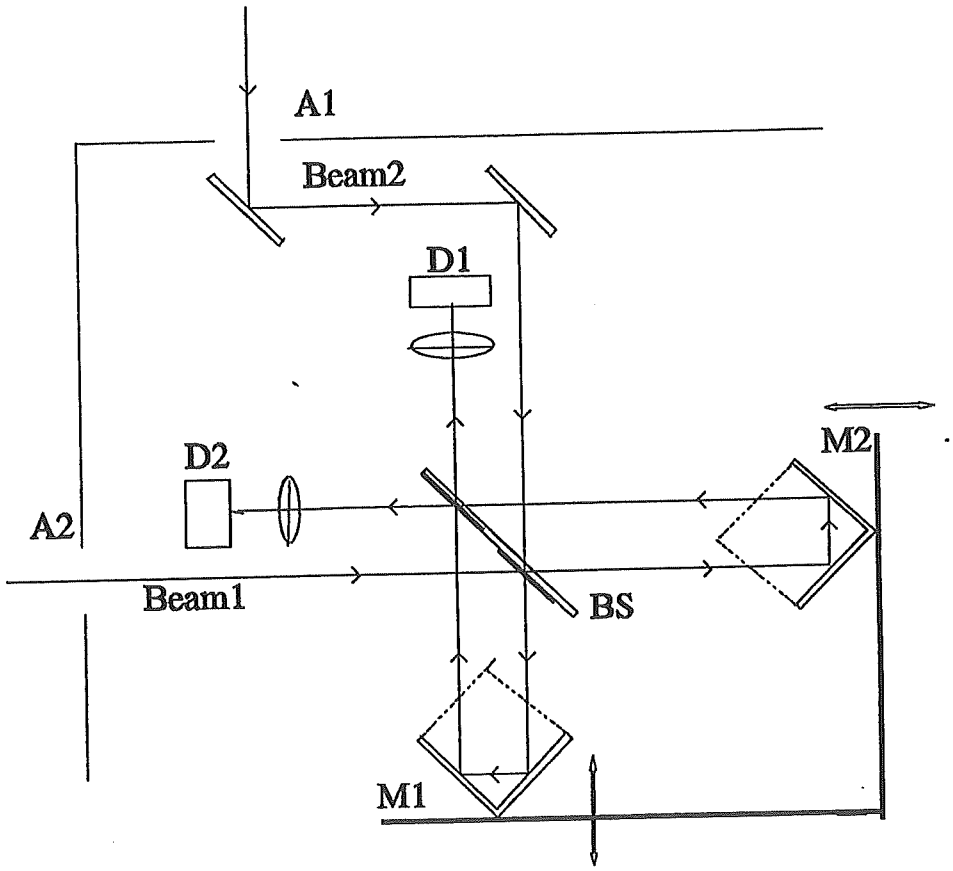


Fig. 3. Double-beam interferometer schematic

reflected 180° independent of the tilt of the retroreflectors. The incoming source beam is divided into two almost equal beams by the beamsplitter; the beams then respectively travel to the two mirrors and back to the beamsplitter. After traveling these different paths, the beams are recombined. If the beams are from a coherent source, longitudinal interference occurs, producing Haidinger fringes (fringes of equal inclination). The centers of these fringes are focused on the center of the detector array. The signal produced at the detector consists of an interferogram and a DC component. The interferogram is made up of many superimposed cosinusoidal signal frequencies, each of which is proportional to a wavenumber in the spectrum. As the mirrors are moved,

the optical path of one beam relative to that of the other is changed. The output of the detector during this scan can be expressed as:

$$D(\sigma) = 2\varepsilon \int_0^{\sigma_m} B(\sigma)(1 + \cos(2\pi\sigma\delta))d\delta, \quad (1)$$

where $B(\sigma)$ is the source spectral energy density, σ is the spectral wavenumber, σ_m is the upper wavenumber cutoff of the instrument, ε is the absolute value of the reflectance and transmittance product of the beamsplitter, and δ is the optical path difference between the two interfering beams. If the DC term is removed, Equation (1) reduces to an expression of the interferogram which is

$$I(\sigma) = 2\varepsilon \int_0^{\sigma_m} B(\sigma)(\cos(2\pi\sigma\delta))d\delta \quad (2)$$

This expression is the cosine Fourier integral of the spectrum. Recovery of the spectrum is then achieved by taking the inverse Fourier transform of the signal as a function of optical path difference.

The interferogram detected by Detector D1 as a result of M1 retardation with only Beam1 entering is complementary to the interferogram detected by D1 as a result of only Beam2 entering. When both Beam1 and Beam2 enter the interferometer simultaneously, the interferogram resulting from background is suppressed resulting in the detection of the target interferogram alone. This method is used only for measurements in the far infrared region. In case of the NIR emission source, where there is low background, one

beam is blocked.

The mirrors M1 and M2 are mounted on a rocker arm. When M1 moves a distance of one-fourth-wavelength away from the beamsplitter M2 moves by the same distance but towards the beamsplitter thus introducing an optical path difference (OPD) of one-half-wavelength and causing the interference. A He-Ne (Helium-Neon) laser, of wavelength $\lambda = 632.8$ nm, is mounted on the interferometer along with its own detector to provide an absolute retardation reference. This is the smallest interval at which data can be sampled. The mirror is moved continuously and the data from the main detector is sampled at every zero crossing of the laser signal, which is sensed by the laser reference detector electronics. Thus, the interferogram is sampled at equal intervals of path difference 632.8/2 nm. By moving the mirrors through different distances and keeping the sampling frequency constant, different resolutions can be obtained.

The Bomem interferometer has a switch to select a particular resolution to sample the data. It is given in terms of wavenumber, the inverse of wavelength or path difference, given in cm^{-1}. At higher resolutions the mirror drive distance should be longer. For example, at a resolution of 2 cm^{-1} the mirrors should move by 0.5 cm in one direction and at a resolution of, say, 32 cm^{-1} the mirrors should move by $^1/_{32}$ cm in each direction. The number of samples and hence the number of bytes transfered by the interferometer can be computed from this as shown below.

Let the resolution be x cm^{-1}. Therefore, the total path difference = 2/x cm. Distance between two samples = 632.8/2 nm = 316.4 nm = 316.4 * 10^{-7} cm. Hence,

$$Number\ of\ samples = \frac{2}{Resolution * 316.4 * 10^{-7}} \quad (3)$$

Equation (3) gives the relationship between number of samples and resolution. In interferometer terms, a half-scan is the sampling of the data when the mirrors move in one direction, and a scan is the data obtained from both the directions of the mirror motion. The Bomem marks one as direction '0' and the other '1'. The Bomem interferometer has eight resolutions from which to select, 1, 2, 4, .., and 128cm^{-1}.

An indium-gallium-arsenide (InGaAs) alloy-based detector is used to measure the infrared radiation. The InGaAs photodetectors in NIR instruments offer improvements in terms of sensitivity, dynamic range, and speed of response. Due to its high absorption coefficient [1] and a large energy band gap, it excels in the 1000 to 1700 nm spectral region over all other available materials, especially germanium, which operates in the 1300 to 1550 nm region.

The energy-band diagram for both InGaAs and Ge is shown in Fig. 4. InGaAs is said to have a "direct band gap" because the smallest energy gap between the valence band and conduction band occurs at the same value of momentum (k). Thus, when light of sufficient energy is absorbed, an electron can transfer directly from valence to conduction band without changing momentum. On the other hand, in Ge the gap occurs at different values of k, and when light is absorbed, an electron can only jump from valence to conduction band if a photon is created. The photon is required to conserve momentum. Complications arising from this indirect band structure cause Ge to have a lower absorption coefficient near 1300 nm.

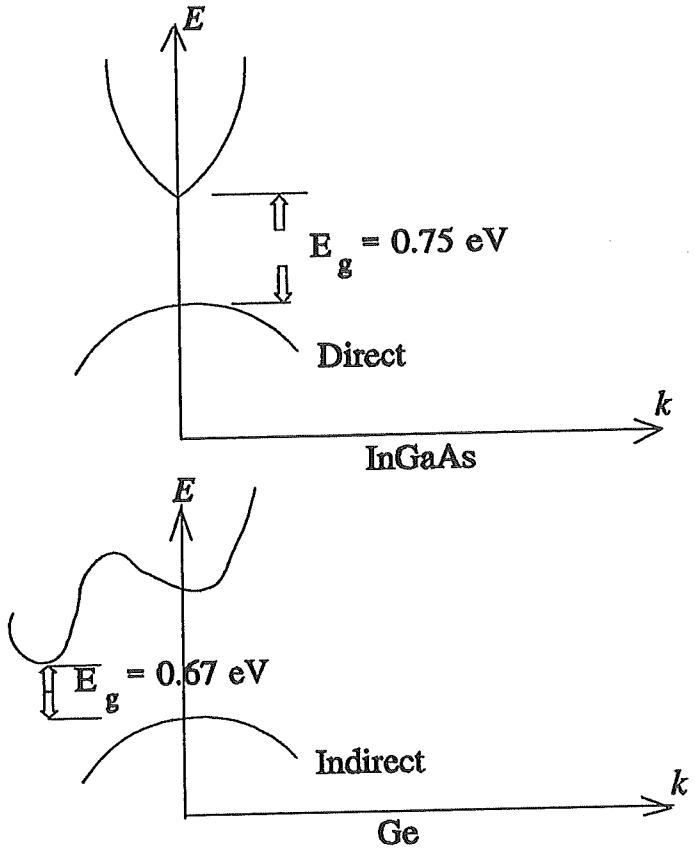


Fig. 4. Energy-band diagram of InGaAs and Ge

The Bomem interferometer transfers data to the PC using the direct memory access (DMA) technique, because it provides a simple solution [11] from the hardware design standpoint. With DMA, the data from the interferometer is fed directly into the PC memory a byte at a time. While the transfer of data is taking place, the processor is free to perform any other tasks. This, however, degrades the processor throughput since the DMA operates by stealing cycles, holding off the processor for just long enough to transfer the next byte. This can be thought of as a primitive kind of multitasking: The DMA transfer takes place in the foreground, in real time, while the

analysis and display of data take place in the background. Appendix A gives the details of the signal from the interferometer and its hardware specifications.

C. Software Specifications

Despite the great variety of remote data acquisition problems, one design consideration common to all is the signal-to-noise ratio. Because the information must travel over a distance, it is subject to a great deal of interference and the signal must be extracted from the interfering noise. Increasing the sensitivity of the receiver does not solve the problem since a more sensitive receiver picks up more noise as well. Increasing the power at the transmitter is not a solution since in remote sensing we do not have control over the transmitter since these are, in most cases, natural sources. The answer to this lies in appropriate processing of the data.

In many cases the cost of losing data could be prohibitive to the analysis. For example, conditions under which the data source needs to be observed could be of limited existence in time. Hence the software must be capable of processing the data set and releasing the buffer for the next data set before the data is overwritten by the new set. A typical solution is to operate the software on a *Ping Pong* buffer principle. Two buffers are allocated for data to be collected. While one buffer is being filled by DMA, the other buffer's contents are available to the program for whatever data processing is desired. When the transfer into the first buffer is complete, the buffers are swapped and a new transfer is started. The Bomem interferometer can send data at the speed of 6 KHz or 16 KHz. Assuming the speed to be 16 KHz (16 kilobytes per second) there are just four seconds of data in a 64K byte buffer. Although a lot of processing can be done

in four seconds, it is not an infinite amount of time. If a program needs to do numerically intensive calculations, its first action when buffers are swapped should be to copy some or all of the new buffer into a working space in memory.

Although the Michelson interferometer has a better signal-to-noise ratio than the grating spectrometer (Fig. 2.), its signal-to-noise ratio, for observing the NIR OH Meinal bands, can further be improved by integrating the data, that is, by coherently adding the respective half-scans, over a time interval less than or equal to 3 minutes, as recommended by CEDAR. Coadding reduces the variance of the additive noise hence increasing the S/N ratio. The spectral analysis may then be performed on this high S/N ratio, coadded data.

The interferometer interrupts the processor after it sends the right number of bytes depending on the interferometer resolution. This is to allow the processor to swap the buffer and start a new transfer. Hence the data acquisition is entirely interrupt-driven, and once started, it runs by itself, alternately filling and swapping buffers like clockwork. It can be thought of as the higher priority *foreground* task, with the data processing filling up the time as the *background* task [11].

D. Limitations of DOS

The challenge of maximizing data acquisition for better analysis of the data was met with the advent of high speed A/D convertors. A large main memory was required to process these data, which caused developers of data acquisition software to constrain their software development to main frames and other computers with large memory. For some applications the costs of these computers were prohibitive. Also, the development

of real-time data acquisition software for these computers, each of which may use a different operating system, was a costly affair as compared to the cost of software development for a low-cost IBM PC using MS-DOS. However, MS-DOS had limitations in accessing the memory.

The IBM PC was announced with no less than three different operating systems of which MS-DOS, for various reasons, rapidly became the operating system of choice. The first IBM PC had Intel's 8088 as its processor, a slightly slower variant of the 8086. The 8086 is capable of accessing 1024K or 1 megabyte of physical address space, but IBM's design restricted the operating system and application programs to the first 640K of the address space, reserving the remaining 384K for use by routines in read-only memory (ROM) and by hardware subsystems. Fig. 5 shows how the address space was divided.

The interferometer, for a wavenumber resolution of 2 cm^{-1}, transfers 32768 (32K) samples of data, with two bytes per sample, amounting to 65536 bytes (64K bytes) of data. For this project, with the software specifications as mentioned in the previous section and with the interferometer at 2 cm^{-1} resolution, the minimum memory requirement would be 768 KB as shown in Table I. Also, memory is needed for executing the program itself. So we proposed to use a PC-AT with Intel's 80386 processor with 4 MB of physical memory. This still would not solve our problem since MS-DOS, designed for a 8086 based PC, would not allow us to use the memory above 1MB of physical address. The solution was to use a DOS Extender which would allow our program to use all of the 4MB by operating the 80386 in the protected mode. The

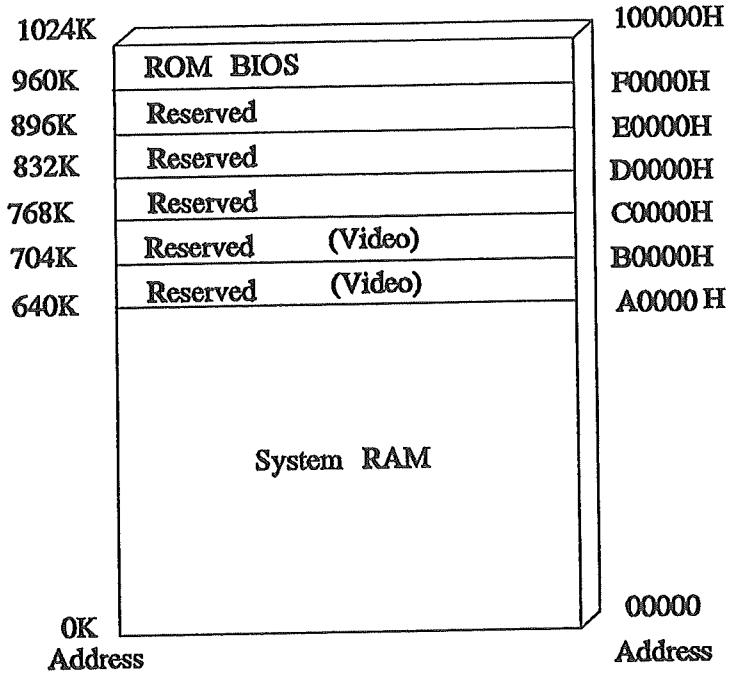


Fig. 5. The IBM PC address space.

TABLE I
Memory Requirements

Type of Data	Number of buffers	Data type	Buffer size in KBytes	Total size in KBytes
Raw	2	Short	64	128
Coadd	2	Short	64	128
FFT : Real	2	Float	128	256
Complex	2	Float	128	256
			Total Size	768

Short is 2 Bytes long and Float is 4 Bytes long.

operation of 80386 in the protected mode is discussed in detail in Chapter III.

CHAPTER III

PROTECTED-MODE OPERATION

Intel first began shipping its second generation 16-bit microprocessor, the 80286, in 1982. To those who were paying attention, the 80286 represented a significant advance in the capabilities of the microprocessor as compared to the 8086; it extended the physical address space from 1 megabyte to 16 megabytes. It included a mechanism with which one program could be prevented from corrupting the code or data of another; this provided for the development of secure multitasking systems. Intel introduced its first 32-bit microprocessor, the 80386, in 1985. It was followed by the 80486 in 1989. The 80386 includes the features of 80286 but has larger memory-addressing capabilities. The 80386 can address 4 gigabytes of physical memory and 64 terabytes of virtual memory. All these processors allow applications to "overcommit" memory, running in a logical address space that was much larger than the physically available memory. This was accomplished through a mechanism called *protected virtual address mode*. This chapter deals with the difference in real and protected mode operation of the 80386 and the capabilities of 80386 in protected mode. This chapter also deals with the requirements and problems of running in protected mode along with DOS.

A. Protected Mode Versus Real Mode

The 80386's protected mode derives its unique capabilities from a change in the way memory addresses are interpreted. The 80386 CPU starts up in the so-called *real mode*, which is basically an 8086 emulation mode; in this mode it forms addresses in

exactly the same manner as an 8086. In 8086 the segment register, a 16-bit register, is simply multiplied by 16 to specify the starting physical address of a block or segment of memory. The offset, which is likewise a 16-bit value, determines which byte in the segment is referenced. The segmented architecture led to various styles of programming. If an application requires no more that 64K of code and 64K of data, then addressing is non-paged and it is called a *small model* program. A *large mode* program uses both the segment and offset to reference any memory location within 0M to 1M of physical memory. When an 80386 is switched into protected mode, however, it interprets the contents of a segment register in a radically different way. The 16-bit value in the segment register is called a selector, and it is used by the CPU hardware as an index into a look-up table--called a *descriptor table*--which contains 32-bit physical base addresses for all the memory segments in the system. The 32-bit offset allows the CPU to address 4,096 megabytes, or 4 gigabytes, of physical memory. The 32-bit base address is added to the 32-bit offset to give a 32-bit *linear address*. Because the same selector and offset may reference any one of many different physical addresses, depending on the base address in the look-up table, the protected mode 48-bit selector:offset pair is called a *logical address*. Fig. 6. shows the addressing mode contrast between 8086 and 80386. An element in the descriptor table is called the *segment-descriptor*. Fig. 7 illustrates the segment-descriptor format. Segement-descriptor fields relevant to us are:

BASE: Defines the location of the segment within the 4 gigabyte linear address space. The processor concatenates the three fragments of the base address to form a single 32-bit value.

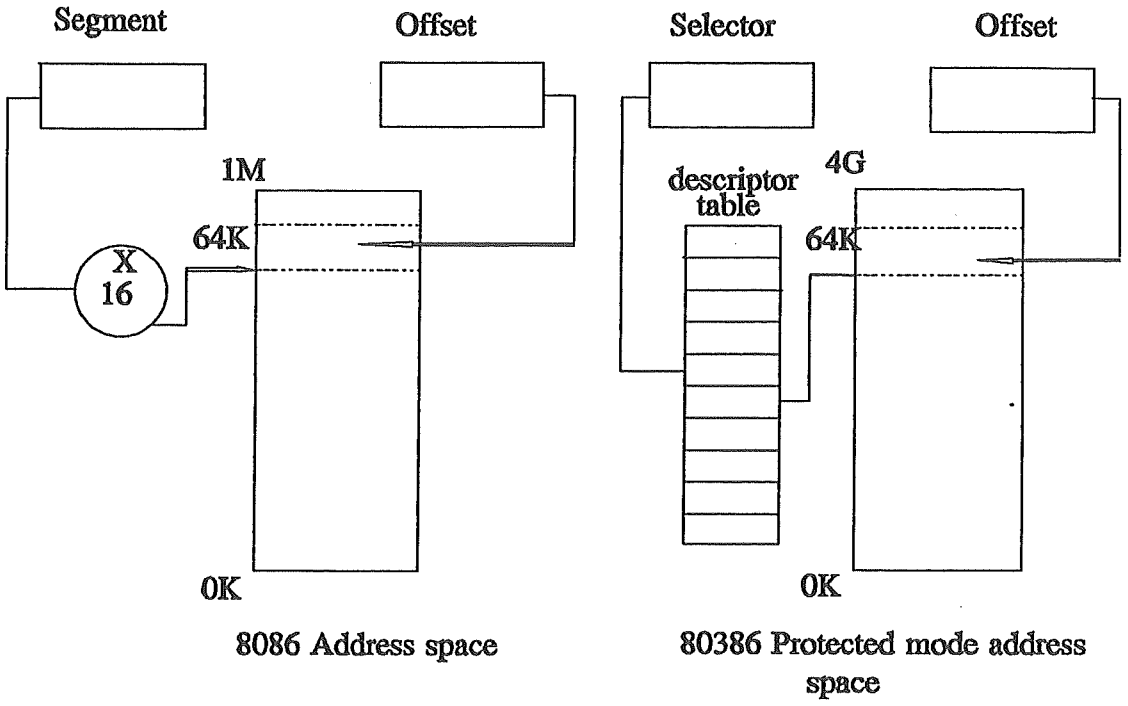


Fig. 6. Addressing modes of 8086 and 80386 contrasted

LIMIT: Defines the size of the segment. The processor concatenates the two parts of the limit field to form a 20-bit value. The processor interprets the limit field in one of two ways, depending on the granularity bit G.

1) In units of one byte, to define a limit of up to 1 megabyte.

2) In units of 4 Kilobytes to define a limit of 4 gigabytes.

Granularity bit (G): Specifies the units with which the LIMIT field is interpreted. When the bit is clear, the limit is interpreted in units of one byte; when set, the limit is interpreted in units of 4 Kilobytes.

TYPE: Distinguishes between various kinds of descriptors.

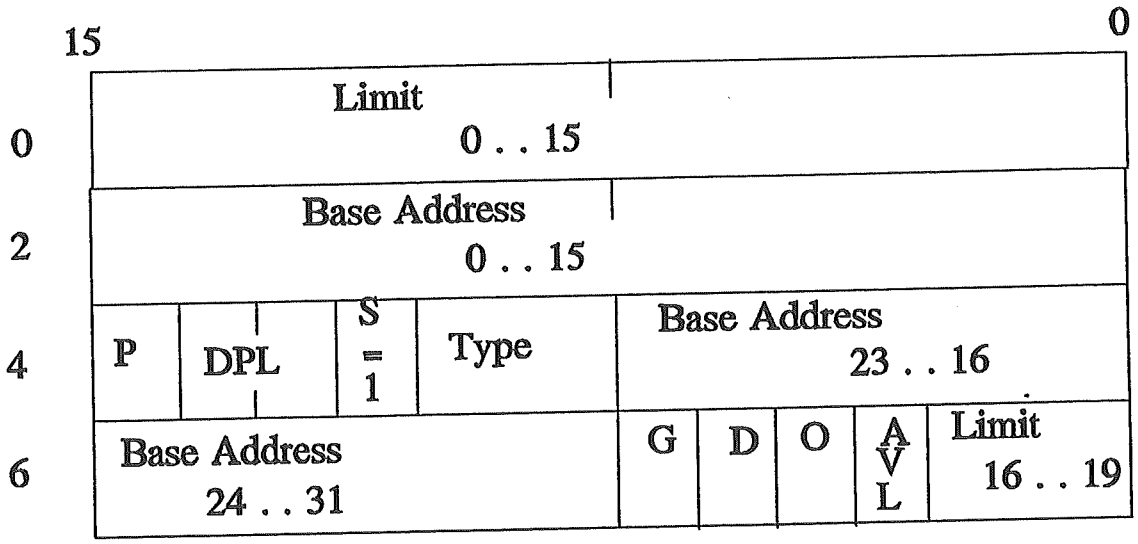


Fig. 7. 80386 segment descriptor

DPL: Descriptor Privilege Level.

Segment-Present bit (P): If zero, the descriptor is not valid for use in address transformation.

Bit S: When set, used for applications code and data segments. When clear, used for special system segments.

AVL bit: Available for use by systems programmers.

The IBM PC/AT incorporating the Intel 80386 CPU has a true 32-bit bus and the capacity to support the full 4 gigabytes of RAM addressable by the 80386. The memory above the 1-megabyte boundary (called *extended memory* by IBM) could only be accessed by a program running in the 80386 protected mode. The "protection" in protected mode is derived from an operating system's point of view. Protected mode

isolates one program from another by not allowing direct access to the system resources. A level of indirection is imposed on all memory access, which can be validated by the operating system. The segment registers contain special values called selectors (Fig. 6), which point to a system resource called a *descriptor table*. This table is interpreted by the CPU but maintained by the operating system.

Under a true protected operating system, application programming is actually simplified. The programmer need not worry about the selectors. There is no need to compute addresses or to do segment arithmetic; the operating system takes care of selectors at load time, or in response to a memory allocation request. This leads to an increase in the execution speed of programs which use this memory since they do not need to perform these calculations for each memory access.

B. 80386 Capabilities in Protected Mode

The 80386 supports real-mode operation, for the sake of compatibility with DOS and its applications. It also supports all the features of 16-bit protected mode on the 80286. But when the 80386 is running in its preferred, native protected mode, it is a true 32-bit CPU with many new capabilities. All the registers and instructions on the 80386 (with the exception of segment registers and the instructions that manipulate them) can perform their operations 32 bits at a time. Sixteen-bit operations are still supported. Table II lists the 16-bit general register names and their 32-bit counterparts.

The 80386 also introduced a significant change to the familiar 8086 instruction set. The addressing modes of 8086/80286 and 80386 are contrasted in Table III [10]. It can be seen that the registers AX, CX, DX and SP cannot be used in 8086 or 80286

TABLE II
16- and 32-bit General Registers

80286 General Registers	*80386 General Registers*
AX, BX, CX, DX	EAX, EBX, ECX, EDX
SP, BP, DI, SI, IP	ESP, EBP, EDI, ESI, EIP

address computations. In contrast, on the 80386, register addressing is fully generalized, and any of the eight general registers, EAX, EBX, ECX, EDX, EBP, ESP, ESI, and EDI, may be used.

The 80386 has other capabilities too; among them is a new form of address indirection called *paging*. The base address and offset are combined to form a 32-bit *linear address* which can be passed through the CPU's paging mechanism to yield the final, physical address. In effect, paging allows each 4K block of RAM to have its own virtual address. Fig. 8 [10] illustrates a simplified version of the paging mechanism.

C. Protected-Mode Operations on a PC

Though the 80386 has many advanced capabilities when operating in protected mode, it is not used often on 386 PCs. There is a host of difficulties in trying to get a DOS-based PC to do anything in protected mode. The primary difficulty is DOS itself; DOS expects to run in real mode. The ROM BIOS also expects to run in real mode. To remain compatible with the 8088-based PCs, all 386 PCs have one of their address lines switched off to prevent accessing memory above 1 Mbyte. Also, some of the hardware interrupts used by the PC conflict with the interrupts the 80386 uses for error handling in protected mode [14]. However, the PC's address lines and interrupt

TABLE III
Addressing Modes Contrasted

8086/80286 addressing mode		80386 addressing mode	
Operand	Description	Operand	Description
DISP	16-bit displacement (offset)	DISP	displacement (32-bits)
[BX]+DISP	Base Register + displacement	[REG]+DISP	Base register + displacement
[BP]+DISP	Stack frame + displacement	[REG]+ [REG*SCALE] +DISP	Base register + scaled index register + displacement
[SI]+DISP	Source index + displacement		
[DI]+DISP	Destination index + displacement		
[BX]+[SI]+DISP	Base + index + displacement		
[BX]+[DI]+DISP	Base + index + displacement		
[BP]+[SI]+DISP	Stack frame + index + displacement		
[BP]+[DI]+DISP	Stack frame + index + displacement		

controllers are reprogrammable. Also, the 80386 has a special mode (virtual-86 or VM86 mode) that allows old style 8086 programs to operate in protected mode. According to the Intel documentation, running 8086 code in VM86 mode is straightforward. This, however, fails when it comes to the PC's BIOS and DOS.

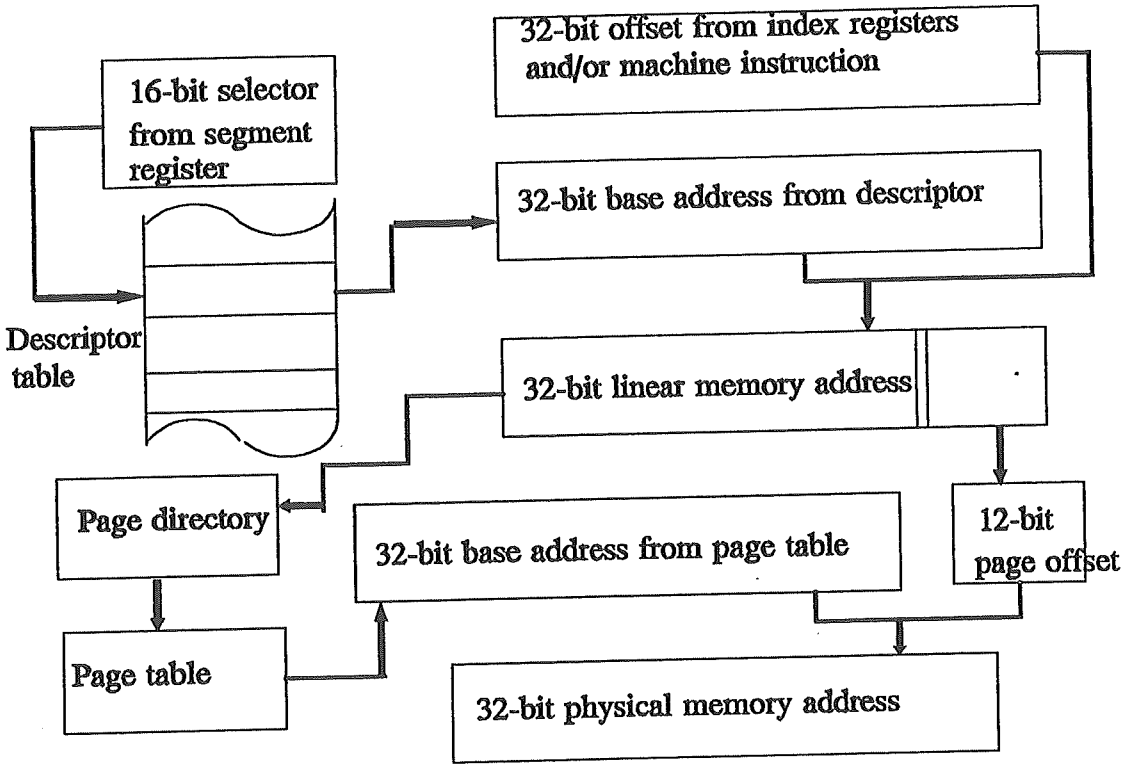


Fig. 8. Virtual address translation through paging

The lack of a DOS-compatible operating system to manage the descriptor tables and other system resources tended to put a damper on the development of protected-mode PC software. Hence, 386-base protected-mode DOS extenders grew out of the need to take advantage of the more powerful features of 80386 computers, without foregoing MS-DOS. The 386 DOS extenders allow large applications to run under DOS. These 386 DOS extenders set up a bridge to the DOS environment and put the 80386 chip into protected mode so that both larger memory space and the full 386 instruction set are available. DOS extenders are something less than an operating system, but more

than a subroutine library. Essentially, they act like an operating system when it comes to memory management features, hiding descriptor table management the way an operating system would, but they have no device handlers or file system. The DOS extender passes an application's requests for these features on to DOS. The mechanism used to perform this is called *mode switching*. To an end user, an application built with a DOS extender can look like any other DOS program. When the user executes the program, control passes invisibly to the 386 DOS extender, which loads and runs the application in protected mode. Such a protected mode application can transparently invoke real-mode DOS or BIOS services. A 386 DOS extender has an option of running DOS and BIOS in Virtual 8086 mode rather than in real mode, but some of the protected-mode features that are also available in real mode are unavailable in VM86 mode. For example, a VM86 task cannot switch the processor into protected mode in the same way a real mode program can. Also, to run the BIOS and DOS in VM86 mode, certain instructions, most notably interrupts, have to be emulated, the hardware interrupt controllers have to be reprogrammed, and their interrupts redirected to the proper routines. Hence most DOS extenders prefer switching the 386 to the real mode to service DOS and BIOS system calls.

Before switching the 386 to the protected mode the descriptor tables have to be constructed. A protected-mode loader starts off running in real mode under MS-DOS. The loader constructs a global descriptor table (GDT), local descriptor table (LDT), and interrupt descriptor table (IDT) for the application, switches the machine into protected mode, and then spawns the application. The IDT is set up to handle interrupts for MS-

DOS and BIOS services. When an application makes a DOS or BIOS request, say an INT 21H, the extender's handler catches it and acts either as a front end to, or replacement for, the corresponding interrupt service routine in real mode. The DOS extender can service the interrupt itself, or it can modify it, switch the machine back into real mode, resignal the interrupt, modify the return value, and switch the machine back to protected mode. All this happens inside the INT instruction with the knowledge of the application. Hence the IDT is crucial for the operation of a DOS extender. When the program exits, the DOS extender puts the computer back into real mode and exits back to DOS.

D. Advantages of Using the 80386 in Protected Mode

To illustrate the advantage of using a 386 machine as it was meant to be used, in protected mode and not as a fast 8086, consider the C function shown in Fig. 9.

```
int foo(char *p)
{
char *q;
q = p;   /* assign char pointer to another char pointer */
return *q; /*return char*/
}
```

Fig. 9. foo(), C function.

When this is compiled using a small model with a typical 16-bit MS-DOS compiler such as Borland C++, the resulting assembly output is shown in Fig. 10. This assembly language implementation closely matches the higher-level C representation.

Unfortunately, most software requires more data space than the 64K maximum

```
;{
;char *q;
;q = p;
;
mov si,word ptr [bp+4]  ; mov a 2byte address into si
;
;return *q;
;
mov al,byte ptr [si]   ; mov the byte value at address si to al
cbw   ; convery byte in al to a word in ax
```

Fig. 10. Assembly output when compiled using small model.

allowed by the small memory model. Thus, PC software is frequently compiled with the compact or large model, using 32-bit pointers in a 16-bit environment. Fig. 11 shows the assembly output of the C function using Borland C++ compiler with a large model.

```
;
;{
;char *q;
;q = p;
;
mov ax,word ptr [bp+8] ; mov 2 byte segment (of p) into ax
mov dx,word ptr [bp+6] ; mov 2 byte offset into dx
mov word ptr [bp-2],ax ; move ax (segment) into higher order word of q
mov word ptr [bp-4],dx ; move dx (offset) into lower order word of q
;
;return *q;
;
les bx,dword ptr [bp-4];load the 4byte(dword) value into es and bx
mov al,byte ptr es:[bx];load the byte vale at es:[bx] into al
cbw    ; convert byte to word
```

Fig. 11. Assembly output when compiled using large model.

Fig. 11. shows the inherent inefficiency of 16-bit code: 32-bit quantities such as longs and far pointers are moved 16 bits at a time.

The implementation of *foo()* in the flat memory model (four-byte pointers) by one 32-bit C compiler, NDP C-386, that produces code suitable for a 386 DOS extender, Pharlap's 386|DOS-Extender, is shown in Fig. 12.

```
mov    eax,[esp]+4        ;mov 32-bit value at address [esp]+4 (p) into eax
movsx  eax,byte ptr [eax];mov with sign extend one byte at address eax into eax
```

Fig. 12. Assembly output of a 32-bit C compiler.

The advantages exhibited in this example shows that 32-bit code can easily execute much faster than comparable 16-bit code on the same hardware. The 32-bit protected mode removes not only the 640K barrier, but also the 64K limit on the segment size. Since 32-bit registers can be used as base and index registers, the near pointer loaded into these registers can use up to 32 bits for addressing memory, which means the maximum index within a memory segment is no longer 64K (FFFFH), but 4Gbytes (FFFFFFFFH).

E. Data Acquisition Using the 80386 in Protected Mode

For real-time data acquisition and analysis it is necessary that the PC execute the program at a rate comparable to the rate at which data flows into it from the external device. Also, in our case the memory requirements as shown in TABLE I exceed the 640Kbyte barrier of the PC. To accommodate these requirements it is necessary to operate the 80386 in protected mode but without foregoing the advantages, such as simplicity in programming, of MS-DOS. The 386 protected-mode DOS extender suits well this application due to the features summarized below:

1) large memory spaces for code and data

2) powerful 32-bit instructions

3) highly-optimizing compilers

4) faster numerics using Weitek math coprocessors.

Though there are many advantages in using a 386 protected mode DOS extender, there are some problems when it comes to real time data acquisition. One such problem is handling of interrupts. Interrupts on the 80386/486 fall into three categories:

1) hardware interrupts generated by an external hardware event

2) software interrupts (commonly used for DOS and BIOS system services)

3) processor exceptions generated by the 386/486 chip when memory protection or other programming errors (divide by zero, for example) are detected.

All three types of interrupts are handled in a similar manner by the DOS extenders. When an interrupt occurs in protected mode, the DOS extender always gains control unless the application has taken over an interrupt vector. Taking over interrupts is more complex for protected mode programs than it is for real mode programs, because a protected mode program must be prepared to deal with interrupts that occur in real mode, as well as those that occur in protected mode. A protected mode program taking over interrupts must be aware of the mode in which the processor is operating when the interrupt occurs. This is because of the fact that for taking over an interrupt occurring in protected mode, the IDT has to be set for that interrupt vector and, for taking over an interrupt occurring in real mode, the real mode interrupt vector table needs to be set up.

The interferometer interrupts the processor after transferring the required amount of data. This interrupt needs to be handled so that the buffer can be switched for storing data from the next half-scan. The interferometer also sends two bytes of status information, immediately after the interrupt, on the half-scan data it has just finished transferring. The status word provides information about the data such as the quality of data, the resolution at which the data was acquired, and the direction of the scan to which the data belong. Chapter IV deals in detail with the approach taken to acquire the data from the interferometer and the way the hardware interrupt is handled from the protected mode.

CHAPTER IV

DESIGN AND IMPLEMENTATION

Although there exists a huge amount of commercial data acquisition software, the need for customized software is still growing. In our case, though Bomem's software does the acquisition from the interferometer, it has several drawbacks as described in Chapter II; hence there was a need to overcome these pitfalls and also to make other enhancements such as improving the speed, and S/N ratio, and automating the measurements for longtime observation of the atmosphere. The design of the software involved three stages; the first was the design of the acquisition routines, followed by the interrupt handler design and finally the design of various interfaces.

A. Acquisition Approach

The Bomem interferometer interrupts the processor at the end of every half-scan. This interrupt can be handled to switch buffers to collect the next half-scan. Typically a *Ping Pong* buffer approach is used to collect the data as described in Chapter II. Our implementation involved the design of a buffer management mechanism which allows the user to configure the number of buffers to be used for data transfer. The advantage of such a mechanism is that no data is lost even if data is being collected at a high rate. This is possible due to the availability of vast memory when the 80386 is operated in the protected mode.

1) Buffer Management: The interferometer transfers data using DMA. The 8237 DMA chip can access only a 64K segment memory block at a time and this

segment must be in the first 640K boundary. Each 64K memory segment is a buffer. Hence the number of buffers for data transfer is limited by the availability of memory in the lower 640K memory.

In our buffer management mechanism we allocate the required number of buffers in the lower memory. This constitutes the buffer pool. Separate lists of empty buffers and full buffers are maintained. Initially all the buffers in the buffer pool will be on the empty list. When the interrupt occurs the interrupt handler will ask for a new empty buffer which will release the first empty buffer from the empty buffer list. This buffer is then removed from the empty buffer list and marked as "in use." When the next interrupt occurs, the buffer marked "in use" is added to the tail of full buffer list and a new empty buffer from the empty buffer list is allocated for the DMA to transfer data.

While the DMA is filling the buffer, the data from one of the buffers from the full buffer list is being processed. The data processing routine, when finished with the data buffer it has been processing, will release the buffer and add it to the tail of the empty buffer list. This routine will next ask for a full buffer from the full buffer list for processing.

The priority at which the buffers are processed follows a first-in-first-out (FIFO) algorithm, meaning that the buffer added first to the full list is processed first and the buffer added first to the empty list is allocated first. The advantage of this buffer management mechanism is that the main data processing routine is fully separated from the interrupt handling routine, the only connection being the global *objects* -- the buffer lists. Also the two "processes" can fill and process data respectively regardless of the

time the other "process" takes to do its job.

Although this type of buffer management is effective, it poses a simple problem. The data structures used to build this buffer pool and the buffer lists were linked lists. The initial condition of the buffer pool and the buffer lists are shown in Fig. 13 where the buffer-full-list is empty and the all the buffers in the buffer pool are in buffer-empty-list. Since the data structure is a linked list, when a buffer is being added to the empty or full list if an interrupt occurs, there is a possibility of losing a buffer.

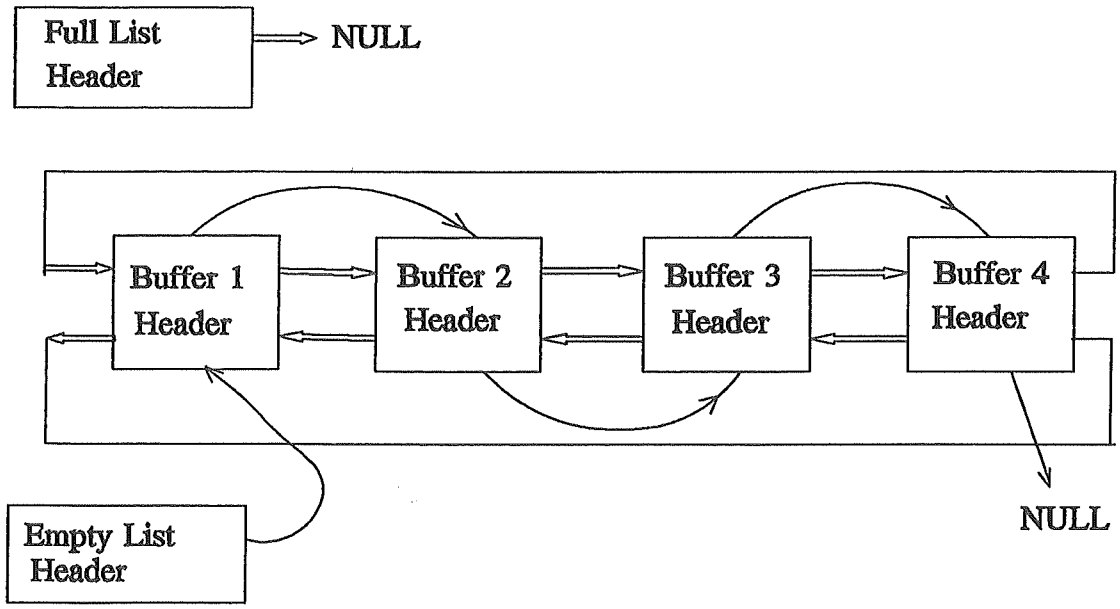


Fig. 13. Buffer pool and lists, initial condition

Consider for example the case of removing a buffer from the empty list and marking it to be "in use." Suppose an interrupt occurs as soon as the buffer has been removed

from the list and before marking it "in use." If this interrupt handler is going to manipulate the buffers, then the buffer just removed from the empty list will never be returned to the full list and the buffer could be lost. A similar situation arises if there are no empty buffers and the data processing routine has finished processing the data from a buffer and it is about to return this buffer to the empty list; if, before it can return the buffer to the empty list, the interferometer interrupts, the interrupt handler will ask for an empty buffer and will not get one though a buffer is ready to be used for data transfer. To overcome these difficulties we switch off all the interrupts whenever any buffer manipulation is in process.

2) *Real Mode Memory Mapping to Protected Mode:* The interferometer sends a DMA request signal for every byte it transfers. To move this data into the PC's memory the 8237 DMA chip is used. The 8237 has registers which can be programmed. It has a register for the target address which can be of 24 bits. Hence data transfer using the 8237 DMA chip can be done only into the lower 640Kbyte of the PC's memory.

First memory has to be allocated from the lower 640Kbyte by a program running in protected mode. There can be no direct manipulation of the segment registers, because in protected mode the value in the segment register is an index into a descriptor table and not the actual segment. Secondly the data in this memory must be accessed from protected mode like any other buffer allocated in extended memory. Pharlap's 386|DOS-Extender helps to overcome these problems.

DOS-Extender is used to allocate memory for the buffers. The extender always allocates and deallocates memory in units of four-kilobyte pages. Both conventional

memory (memory below one megabyte) and extended memory are available for use by the application program. When allocating memory to the application program, conventional memory is allocated first, and extended memory second. Conventional memory is allocated out of a block of memory that the DOS-Extender obtains by calling DOS during initialization. The amount of available conventional memory is fixed at initialization time. By default, DOS-Extender will take all the available conventional memory. The -MINREAL and -MAXREAL command line switches can be used either at link time or at run time to force DOS-Extender to leave some conventional memory free. To guarantee that the buffers are in the conventional memory, the 386|DOS-Extender provides a set of system calls (25C0h, 25C1h, and 25C2h) to call the MS-DOS memory allocator. We use these system calls to allocate the buffers for DMA transfer. This also takes care of returning the exact address which has to be used to load the DMA target address register.

To access the data from the protected mode program, the initial design involved a block transfer of the data from conventional memory to a known virtual address. In other words a conventional memory to extended memory copy was performed. However, this affected the performance of the program, since at higher resolutions huge blocks of 64KByte data needed to be transferred. Hence the 386|DOS-Extender's system call 250Ah was used to map the physical memory (conventional memory) at the end of the data segment. This returns the offset within the segment of mapped memory, which can be assigned to a pointer and hence can be accessed as an array. In other words, only one copy of the data existed and that was in the conventional memory while the virtual

address generated by the system call maps to this conventional memory.

In Fig. 13 the blocks are marked buffer headers and not buffers because they are headers that contain the real mode address (conventional memory address) as well as the protected mode pointer to the actual data buffer. Fig. 14 shows the mapping mechanism.

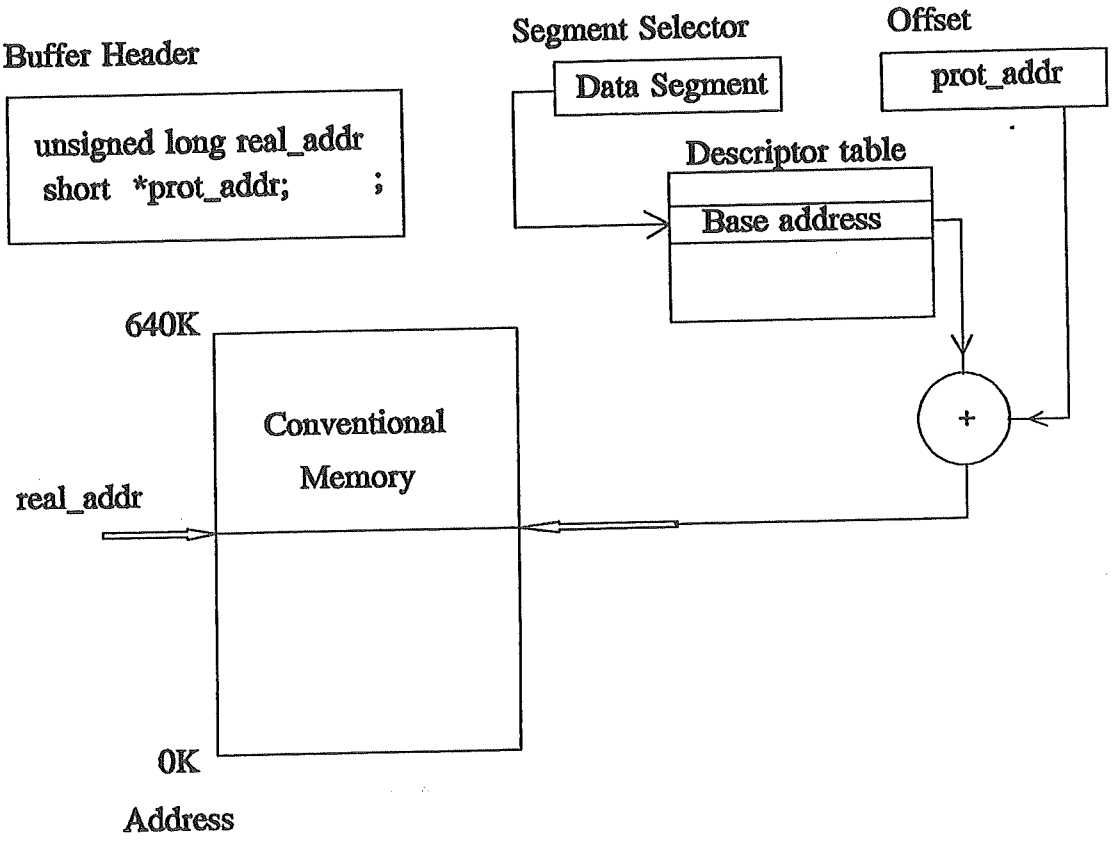


Fig. 14. Physical memory mapping.

The system call 250Ah returns the offset value into *prot_addr* which when referenced later gets mapped as shown in Fig. 14.

3) Transfer of Data to the PC Memory: Once the buffer for the data has been allocated, the next step is to take care of the data transfer mechanism itself. Since the

interferometer sends a DMA request for every byte it wants to transfer, the DMA chip was programmed to operate in the *single transfer mode*. Then the target address and the number of bytes needed to be transferred were loaded into the respective registers. The chip was then enabled to start the data transfer. The data was accepted on channel 1 of the DMA chip.

The number of bytes transferred by the interferometer varies depending on the resolution at which it is operating. The theoretical value of the number of bytes to be transferred can be calculated from the resolution. This however need not be the true count of bytes that has been transferred. Hence the DMA count register is initially loaded with a value of 65535 or FFFFh. When the interrupt occurs at the end of the half-scan the DMA count register is read and the value is subtracted from 65535 to give the actual number of bytes transferred. If, due to some reason, such as the shaking of the instrument or switching the resolution too often, the number of bytes transferred is less than or greater than 10% of the theoretical value, the data is rejected.

The resolution at which the interferometer is operating can be found from the status word, which the interferometer transfers, for the half-scan just transferred, after it interrupts the processor. Hence the theoretical value of the number of bytes to be transferred can be calculated. When the interferometer operates at 2 cm^{-1} resolution, it transfers 65536 bytes. When the interrupt occurs, the value in the count register would be FFFFh which when subtracted from FFFFh would give a value of zero, although we know that 65536 bytes were transferred. This problem was overcome by checking if the resulting value was zero. If so, the number of bytes transferred was set to 65536. This

is a valid assumption because if no byte is transferred, then no interrupt would have occurred.

B. Interrupt Handler

Designing the interrupt handler and developing it was one of the most challenging stages of this thesis. It was especially difficult to write an interrupt handler that would operate from the protected mode and still take over the hardware interrupt. As mentioned earlier (Chapter III) a protected-mode program handling an interrupt must be aware of the mode in which the processor is operating when the interrupt occurs. Hardware interrupts can occur at any time, and a program taking over the hardware interrupt must account for the possibility that the interrupt could occur in real mode as well as protected mode.

When an interrupt occurs, the processor pushes the processor flags and the current CS:EIP instruction address on the stack before transferring control to the interrupt handler. The interrupt handler saves all the registers it is going to modify, performs its processing, and restores all the saved registers. It then executes a return from interrupt instruction, which restores the saved flags and instruction address from the stack.

There are two forms of the interrupt return instruction: a 16-bit form, created by the IRET instruction mnemonic, and a 32-bit form, created by IRETD instruction mnemonic. It is extremely important to select the correct instruction. When an interrupt occurs in protected mode, the processor always saves information on the stack in 32-bit form, regardless of the segment use type attribute. Therefore a protected-mode interrupt handler must always return with the IRETD instruction.

Hardware interrupt handlers must make no assumptions about their environment, since a hardware interrupt can occur at any time. If a handler uses more than 100 bytes of the stack space, it should switch stacks when it gains control. No assumptions of any kind should be made about any segment registers except CS. If the handler makes any calls to routines written in a high-level language, it must set up the environment the high-level language expects. Hardware interrupts must also guard against being reentered. They must either be written to be reentrant, or they must prevent it by (1) leaving interrupts disabled, (2) not issuing the end-of-interrupt (EOI) command to the 8259 interrupt controller until done with its processing or (3) using a semaphore to gate entrance to the handler. Also, hardware interrupt handlers must not make DOS or BIOS system calls, since neither the DOS nor the BIOS is reentrant.

A typical way to implement a hardware interrupt handler is to set a flag when the interrupt occurs and exit; later the main program may check the flag and take appropriate action if the flag is set. Initially our interrupt handler was designed to take over the hardware interrupt in real mode. However, for the real mode interrupt handler to switch the DMA buffers, the buffer address needs to be provided by the protected mode program. A variable declared in the real mode program can be accessed by the protected mode program directly. But a real mode program cannot access the protected mode variables because the descriptor table is not active when the real mode program is executing. Hence the interrupt handler was designed so that a flag variable, declared in the real mode interrupt handler code, is set when an interrupt occurs; the handler then simply exits. This flag is checked periodically in the protected mode main program

which, if the flag is set, resets it and switches the buffer.

This switching of buffers may not occur immediately after the interrupt has occurred. Hence there exists a possibility of losing or overwriting the data. So the design was changed to take over the hardware interrupt always in protected mode. The DOS extender provides a system call 2506h which can take over a specified interrupt such that the protected mode interrupt handler always gains control, regardless of whether the interrupt occurred in real mode or in protected mode. This function modifies both the protected mode interrupt descriptor table and the real mode interrupt vector table. The original interrupt vectors for both real mode and protected mode interrupt should be saved before calling this function and restored before the program exits. This design helped in implementing the interrupt handler in protected mode; the handler can access all the variables the other protected mode functions can access within its scope. Therefore, the buffers could now be changed from within the interrupt handler ensuring that the buffer switch occurs immediately after the interrupt occurs. Fig. 15 shows the implementation of the interrupt handler. Since there are many functions called from within the interrupt handler, the stack is switched as soon the interrupt handler gets control.

C. Data Processing and Analysis

Once the data has been collected properly, the next step involves processing the data correctly to enhance the analysis. To a large extent in the near infrared the background noise is negligible and one must deal only with the random noise of the detector and associated electronics. One of the main objectives of signal processing is

```
newstackseg equ 14h

_pdma_inth   proc   near
             cld            ;Clear direction flag
             pushad         ;save all registers
             push   ds      ;save data segment
             push   es      ;save es on stack
;Switch stack
             mov    ax,ss   ;mov stack segment to ax
             mov    ebp,esp;mov extended stack pointer to ebp
             mov    bx,newstackseg      ;load new stack seg to bx
             mov    ds,bx   ; move bx to ds
             mov    es,bx   ; move bx to es
             mov    ss,bx   ; move bx to ss so that ds=es=ss=newstackseg
             mov    esp,dword ptr offset newstackptr    ;load stack pointer
             push   ebp     ; save ebp and eax on stack
             push   eax     ;these contain the old stack segment and pointer

             call   _SwitchBuf   ;call SwitchBuf function

             call   _eoi    ;call eoi to indicate end of interrupt
;Restore original stack
             pop    eax     ; get old stack seg into eax
             pop    ebp     ; get old stack pointer to ebp
             mov    ss ,ax  ;move ax to ss
             mov    esp,ebp; move ebp to esp

             pop    es      ;restore es and ds
             pop    ds
             popad          ; restore all registers
             iretd          ; return from protected mode interrupt handler
_pdma_inth   endp
dataseg segment dword rw use32 public 'data'
             dd     200 dup(?)    ; 200 * 4 bytes of local stack
newstackptr:
dataseg ends
```

Fig. 15. Interrupt handler.

to improve the signal-to-noise ratio before analyzing. Since the noise that corrupts our signal is random, it can be minimized by adding a set of signals together, so that the

random components of the noise offset each other. The signal, which is coherent, adds linearly whereas the random noise components decrease as the square root of the number of scans added. In our case a certain specified number of half-scans from one direction can be added together to achieve reduction in noise. This type of adding half-scans from the same direction is called *coadding*.

The number of half-scans to be coadded is read from the scheduler file. The data processing routine calls the coadd function which adds the data from direction 0 into coadd buffer 0 and data from direction 1 into coadd buffer 1. The signal-to-noise ratio improvement due to coadds from both directions is compared with the S/N ratio due to coadds just from one direction, which is what the Bomem software does, as described in Chapter V. If a half-scan is bad or the number of bytes transfered is out of bounds or if the resolution is different, then the data is rejected and the buffer is simply added back to the empty list without coadding. Coadding one bad half-scan will corrupt the final data of an entire experiment.

After adding the specified number of half-scans together a format conversion routine is called. This sets up a header for each coadded half-scan. The header and the data are written to the disk, if asked for, and/or a display routine is called to display the interferogram. If a fast Fourier transform (FFT) is asked to be performed by the user (which has to be specified in the scheduler file), the FFT routine is called, the header variables are calculated, and the header is set for the spectra and written to disk (if asked for) and/or the display routine is called to display the spectra of the coadded scan. The data processing routine then returns to check the time and compares it with the time from

the schedule file and reschedules the same experiment or a new one. The pseudo code of the data processing routine is shown in Fig. 16.

```
bufheader = GetFullBuf();
CheckData(bufheader);
if (data is good)
        coaddfn(bufheader);
AddtoEmptyList(bw,pcbuf);
if(data write)
        convert format to FTS;
        write data;
if(data plot)
        Plot data;
if(fft)
        do fft;
        if(fft write)
                convert format to FTS;
                write fft;
        if(fft plot)
                Plot fft;
return;
```

Fig. 16. Pseudo code of data processing routine.

D. User Interface

Finally all the developed stages need to be interfaced together appropriately and a user interface developed to provide to the software necessary information. One of the objectives of this thesis was to automate the atmospheric data collection and analysis totally. Therefore, to conduct different experiments at different times of the day a scheduler is used to start the experiment.

The scheduler schedules a particular experiment at a particular time. It reads the scheduling information from the schedule file. It reads a range of time in 24-hour format

and the number of coadds to be done for this experiment. It gets the current universal time (UT) and checks if it is within the range of any experiment. If it does, the scheduler then starts that experiment with the specified number of coadds. Instead of time, the start and end solar depression angles, can also be given. The current time is converted to solar depression angle depending on the day of the year. This is then compared to the solar depression angles from the schedule file and an appropriate experiment scheduled. The format of the schedule file is shown in Fig. 17.

If the scheduler is not able to schedule any experiment, it waits until an experiment can be scheduled, which could be the next day. An experiment with a coadd of zero specifies that there will be no experiment to schedule for some time, most

```
<Remark field>       : This can be upto 32 characters
<Latitude field>     : This has to be in decimal degrees. egs. 234.56 + = N
<Longitude field>    : This has to be in decimal degrees. + = W
<Elevation field>    : Should be in meters above sea level
<Time flag>          : 0 = Dos     1 = IRIG
<Max Frames>         : Max frames in a file (# ifgs = 2 * Max Frames)
<ifg data Path>      : Path for ifg data file.
<fft data path>      : Path for fft data file
<ifg plot flag>      : 0/1
<fft plot flag>      : 0/1
<start time1>  <end time1>  <# coadd1/direction>
                                .
                                .
                                .
                                .
                                .
<start time n>  <end time n>  <# coadd n/direction>
end_of_file
```

Fig. 17. Schedule file format.

probably till the next day, hence it calls a dummy routine (which will be implemented later) to close the shutters of the interferometer.

The tests done on the software and the results obtained from it are discussed in Chapter V.

CHAPTER V

TESTS AND RESULTS

The software was tested extensively by running it continuously for many days. One of the main features of the software was its capability to run continuously for days without any operator intervention. This was also a requirement demanded by SDL to observe the dynamics of the MALT. The test involves running the software and validating that the data is correct by computing the FFT on the interferogram and by investigating the spectra for OH Meinal lines. We also test the speed of the software and compare it with that of the other data acquisition software available, verifying the maximum data transfer rates that it can handle. We have run the software on different machines to check its portability.

The software was run to collect night sky airglow data. The spectra of the interferogram was analyzed in the range corresponding to a wavenumber of 5000 cm^{-1} to 7000 cm^{-1}. The OH Meinal (3,1) band and the OH Meinal (4,2) band could be observed at 6625 cm^{-1} and 6325 cm^{-1}, respectively. This correspond to the spectral estimation obtained from the past observations and also agrees with the theoretical estimation. Hence the data collected by the software from the interferometer is correct.

Another test made on the software is the signal to noise ratio comparison. The software from Bomem does coadds but only of half scans from one direction. Appendices B and C give the error bars in rotational temperature and intensity derivation, from the interferogram data, for various SNR. The goal is to have minimum error. The software we have developed coadds half scans from both directions. The reason for

coadding the half-scans is to minimize the random noise. The signal to noise ratio is computed by calculating the root mean square value of the data and subtracting the maximum value from it and then dividing the maximum by this number. Equation (4) and equation (5) completely describe the SNR computation. N is the total number of data points, I_j is the jth data point value and I_{max} is the maximum of all N data points.

$$SNR = \frac{I_{max}}{I_{rms} - I_{max}} \tag{4}$$

where,

$$I_{rms} = \sqrt{\frac{\sum_{j=1}^{N} I_j^2}{N}} \tag{5}$$

Though it is a crude estimate of signal to noise ratio it gives a decent measure of the improvement. Fig. 18 and Fig. 20 show the spectra of a single direction coadded interferogram. Fig. 19 and Fig. 21 present the spectra of interferograms from both directions combined. The spectra of the coadded interferogram from either direction is computed and the average of these spectra is computed to obtain the combined spectra. Fig. 19 and Fig. 21 clearly show a noticeable reduction in the noise level. The S/N ratio obtained from the single direction coadded scans was 28.352 and the S/N ratio obtained from both directions combined was 40.593, which means there is nearly a 40% increase in the S/N ratio.

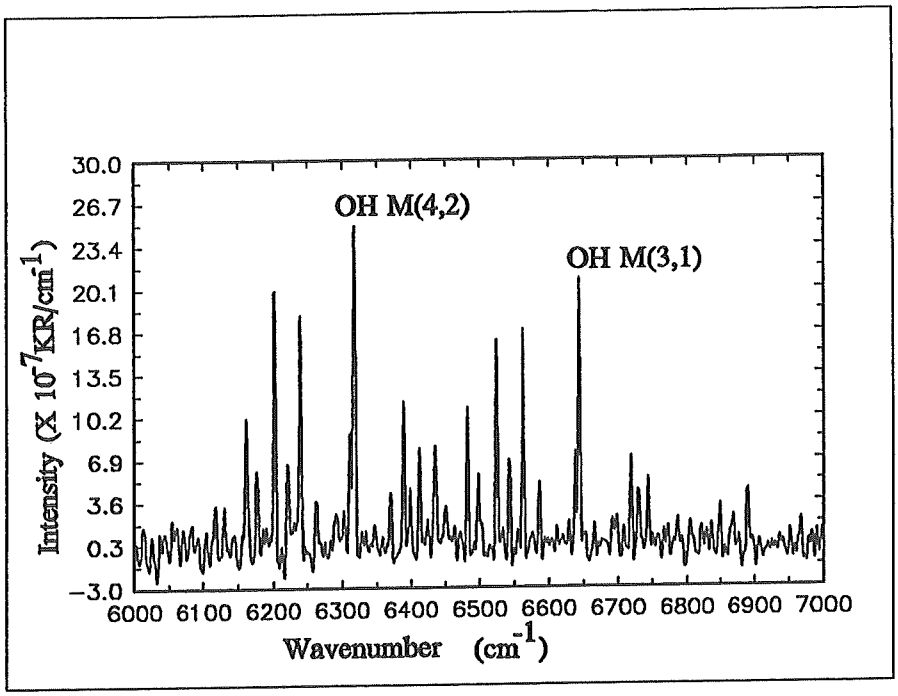


Fig. 18. Spectra of the single direction interferogram with 200 coadds in one direction.

The interferogram at a resolution of 4 cm^{-1} transfers one half-scan data, i.e., 32 Kbytes in 2.7 sec. Coadding the 16 K samples will not take as much of the CPU's time as the FFT on the coadded data will. However, a FFT is performed only after a specified number of coadds are performed. A 16 K sample FFT with phase correction and interpolation takes on the order of 5 sec with the Intel's 80387 math coprocessor. With a Weitek coprocessor this can be reduced to nearly 2 sec. If a single coadd is performed, i.e., the scans are directly stored without any coadd, then the processor is busy full time and might push the software performance to the limits. With lower resolutions the number of samples is less; for example, at a resolution of 64 cm^{-1}, there are only 512 samples, so at the same data transfer rate one half-scan will be transferred

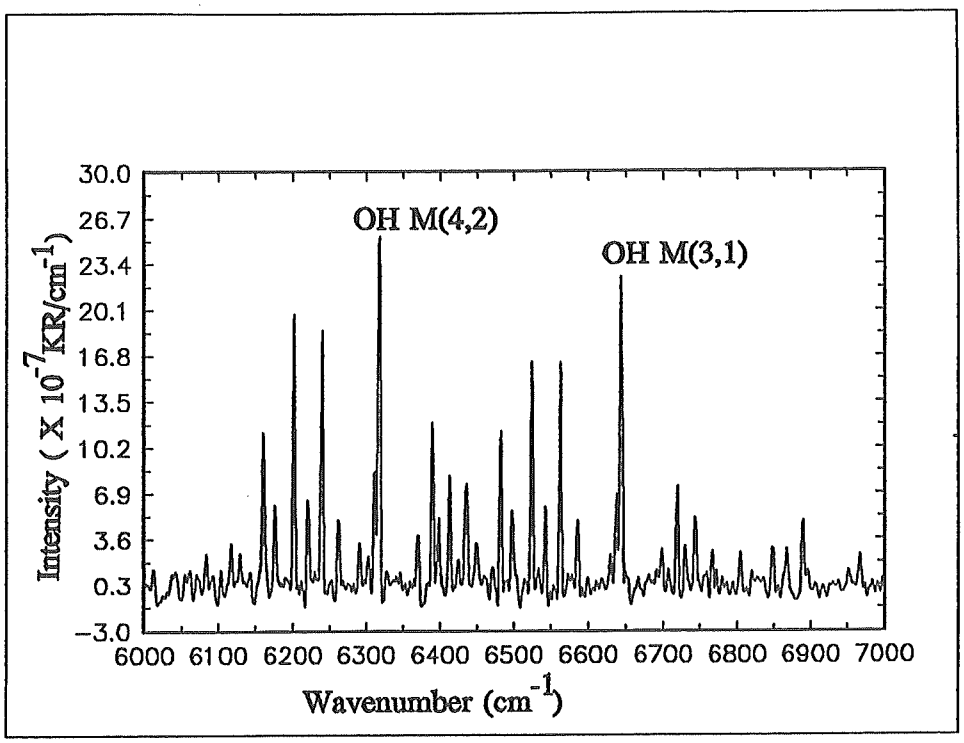


Fig. 19. Spectra of the combined interferogram from both directions.

in 0.0854 sec and the FFT on this data will also require roughly the same time. Hence the data processing part of the software itself can cope with the current data transfer rates.

The software was tested for its ruggedness. No bad scan should be considered for coadd. The interferometer was subjected to a vibration which causes a misalignment in the mirrors which the interferometer detects and indicates by setting a bit in the status word to signal a corrupted scan. The software reads this bit from the status word, and if it is set, it simply throws the scan out.

Since the data transfer is done using DMA and since DMA steals cycles from the processor, the actual time for transferring data is difficult to determine. However, if the

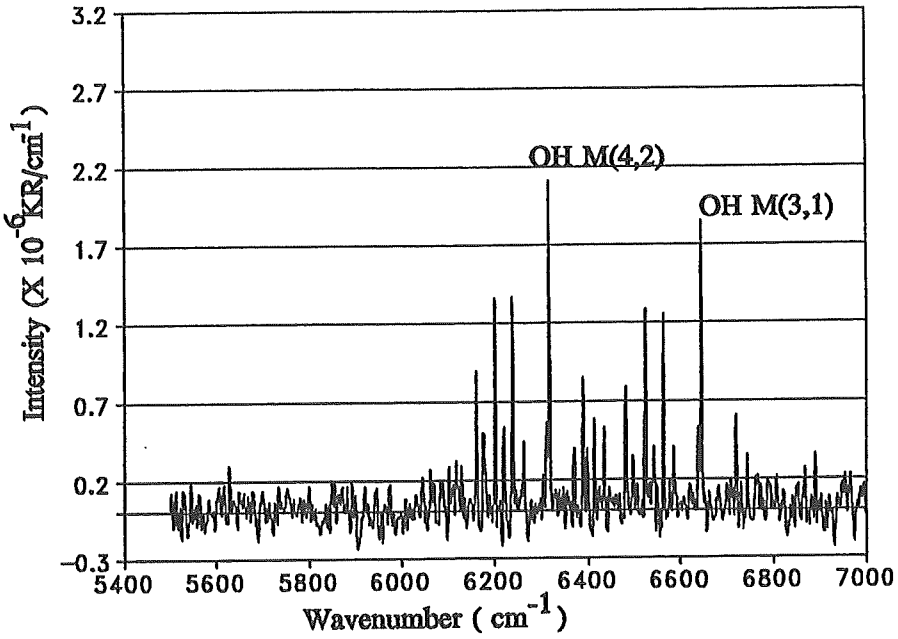


Fig. 20. Spectra of single direction interferogram.

rate at which the interferometer transfers the data is known, then the approximate rate at which the software handles data transfers can be equated to the rate at which the interferometer transfers it. If the interferometer can be made to transfer at a faster rate, then probably an upper limit of the software's data transfer rate can be determined. However this depends on the number of buffers allocated for data transfer, the DMA bandwidth, and the mode in which the data is transfered.

Finally the portability of the software was tested. The software was run on a Dell System 310 under DOS 3.31. The software was then run on a portable Toshiba 3200 under DOS 4.0. The clock speeds of the processors were different; on the Dell it was 33 MHz and on the Toshiba 25 MHz. The software ran as expected on both these

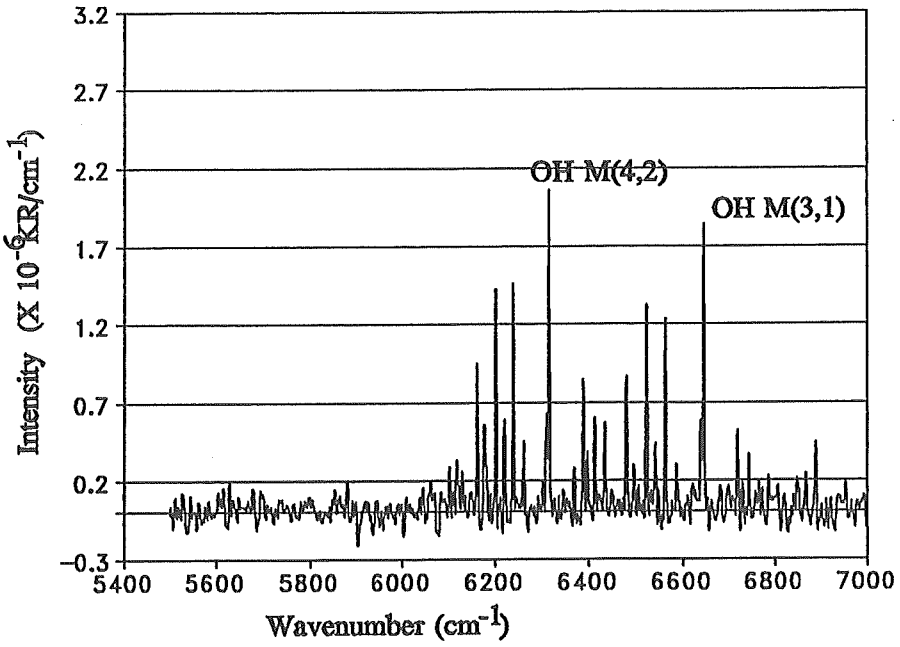


Fig. 21. Spectra of the combined interferogram.

machines.

The software's performance was as expected and satisfied the requirements of SDL to do continuous observations to study the dynamics of the MALT.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

The need for a fast data acquisition system and hence fast acquisition software is ever growing. Also, the need to automate measurements on randomly occurring phenomenona of the atmosphere doesn't have to be exaggerated. Real time acquisition and analysis play a very important role in predictions of atmospheric behaviour based on past measurements. Accuracy in predicting this behaviour is a must and depends on the rate at which data can be acquired and analyzed. This thesis was undertaken in pursuit of a method for more accurate data analysis at a faster rate with low system cost to meet the CEDAR recommendations.

The data that was recorded and the analysis that was performed on the interferograms are good indicators that the software did what was expected from it. The software could be used for long-term scientific observation of the OH Meinal bands. These measurements are needed to separate chemically and dynamically induced perturbations in the MALT. The Michelson interferometer can now run nearly continuously with minimal operator intervention. As proposed [6] the instrument can now be kept running at Millstone Hill for one year in an automated mode and the measurements can then be supplied to the CEDAR data base.

Though the software developed in this thesis satisfies the requirements and meets the recommendations of CEDAR, there is always room for further improvements to enhance the real-time data acquisition and analysis. The rest of this chapter gives some suggestions to enhance the software.

1) A shutter to close the interferometer's window could be added and software can be written to control it from the data acquisition program.

2) The software could be made more *object oriented* [4], [7]. The data buffers can be treated as an object and actions that can be performed on them can be *encapsulated* along with it. For example, the buffers can be made as a base object. The buffer pool, the coadd buffers and the FFT buffers can be derived from this base object. The action needed to be performed on the buffer itself, such as *transfer data*, is *data good*, and is *buffer full*, can be encapsulated with the data buffer. The actions needed to be performed on the buffer pool, such as *get full buffer*, *get empty buffer*, and *do coadd* could be encapsulated with it. On the coadd buffers the *convert format*, *write data to disk*, *display data*, and *FFT* actions can be performed. The actions needed to be performed on the FFT buffers such as *write fft data to disk*, and *display fft data*, can be encapsulated with it. Designing the software in such a way will make future enhancements easier.

3) The software can be rewritten to take advantage of parallel processing facilities [13]. Transputers can be used to perform actions in parallel. One transputer can be delegated the job of getting the data and filling the buffer, while other transputers do the processing. The processing of the data itself can be made parallel. For example, coadding can be delegated to different transputers, each of which coadds a part of the data. Once the

number of coadds specified has been performed, an FFT process can be started again in parallel to perform a FFT on the coadded data. Also, the use of fast vector coprocessors can be used to further speed up the processing. This method will be of greater speed if it is a shared memory environment. If it is on a distributed memory system, then the overhead of transfering data between transputers must be taken into account.

4) A graphical user interface can be developed to create the scheduler file. For example a form can be presented to the user who fills in the necessary data. The user can also be allowed to change an experiment in real time. This means the program need not be exited and the scheduler file changed to start up a new set of experiments. A hot key could be provided which will write the current coadd buffers out and would present the user a form which the user can fill and continue to start the new set of experiments.

Though an endless list of what can be done in the future could be presented, the above mentioned areas are of interest due to their recent advancements.

REFERENCES

[1] V. S. Ban, G. H. Olsen and A. M. Joshi, "High performance InGaAs detectors and arrays for near infrared spectroscopy," *Spectroscopy*, vol. 6, no. 3, pp. 49-53, Aug. 1990.

[2] K. G. Beauchamp, and C. K. Yuen, *Data Acquisition For Signal Analysis*. London: George Allen and Unwin, 1980.

[3] J. J. Carr, *Data Acquisition and Control - Microcomputer Applications for Scientists and Engineers*. Blue Ridge Summit, PA: TAB Books Inc., 1988.

[4] P. Coad, and E. Yourdon, *Object-Oriented Analysis*. Englewood Cliffs, NJ: Yourdon Press, 1990.

[5] E. Duppin, and P. Vavassori, "Simultaneous data acquisition and analysis on a single-task PC," *Meas. Sci. Technical*, vol. 1, no. 12, pp. 1371-1372, Dec. 1990.

[6] P.J. Espy, A. J. Steed and R. J. Huppi, *Proposal to the National Science Foundation: NIR airglow investigations of latitudinal and seasonal variations of the chemistry and dynamics of the mesosphere and lower thermosphere*, Center For Space Engineering, Utah State University, Oct. 1987.

[7] K.E. Gorlen, S. M. Orlow and P. S. Plexico, *Data Abstraction and Object-Oriented Programming in C++*. New York: John Wiley and Sons, 1990.

[8] Z. Guy, "Data acquisition systems: An approach to better programs for data acquisition systems," *Eval. Eng.*, vol. 26, no. 9, pp. 55-61, Sep. 1987.

[9] R.J. Huppi, R. B. Shipley and E. R. Huppi, "Balloon-borne Fourier spectrometer using a focal plane detector array," *Proc. SPIE*, pp. 26-32, Aug. 1979.

[10] R. Nelson, "The IBM PC Programming Architecture," *Extending DOS*, Editor Ray

Duncan. Reading, MA: Addison-Wesley Pub. Company Inc., pp. 1-30, 1990.

[11] T. Nolan, "Real-time data acquisition using DMA," *Dr. Dobb's Journal*, #160, pp. 28-37, Jan. 1990.

[12] F.A. Putnam, "Use your computer to improve data acquisition," *Res. Dev.*, pp. 68-73, Jun. 1988.

[13] T. Scott, "Parallel processing and real-time data acquisition," *IEEE Trans. Nucl. Sci.*, vol. 37, no. 1, Apr. 1990.

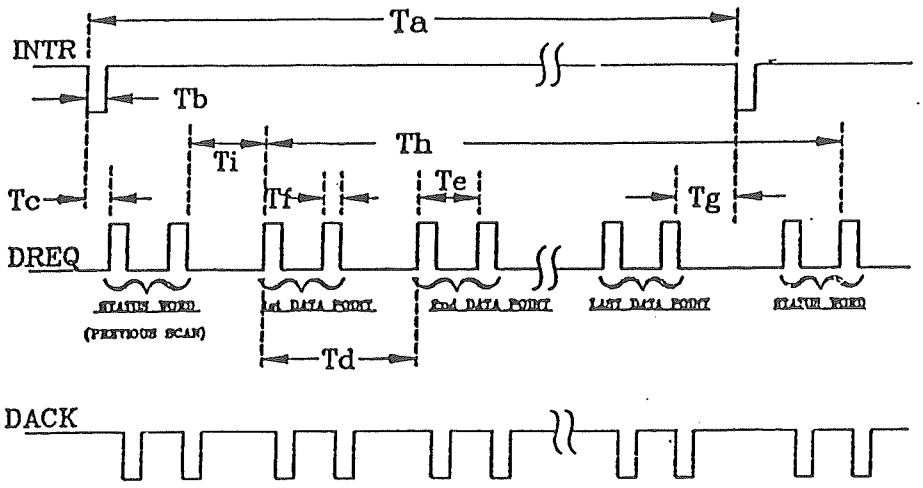
[14] A. Williams, "Roll your own DOS extender: Part I," *Dr. Dobb's Journal*, #169, pp. 16-24, Oct. 1990.

APPENDICES

APPENDIX A:

Timing Diagram Of Interface Board Output Of Interferometer

INTERFACE BOARD OUTPUT



Ta = Half scan duration. Depends on the speed and resolution.

Tb = Interrupt pulse duration = 1 usec

Tc = Delay between interrupt and status word = 12 ms

Td = Delay between two data points. This timing is related to the fringe frequency (16 KHz or 6 KHz) and will fluctuate if the instrument is shaken.

Te = Delay between the transfer of first and second byte of a data point = 23 usec.

Tf = DREQ pulse duration = 1 usec

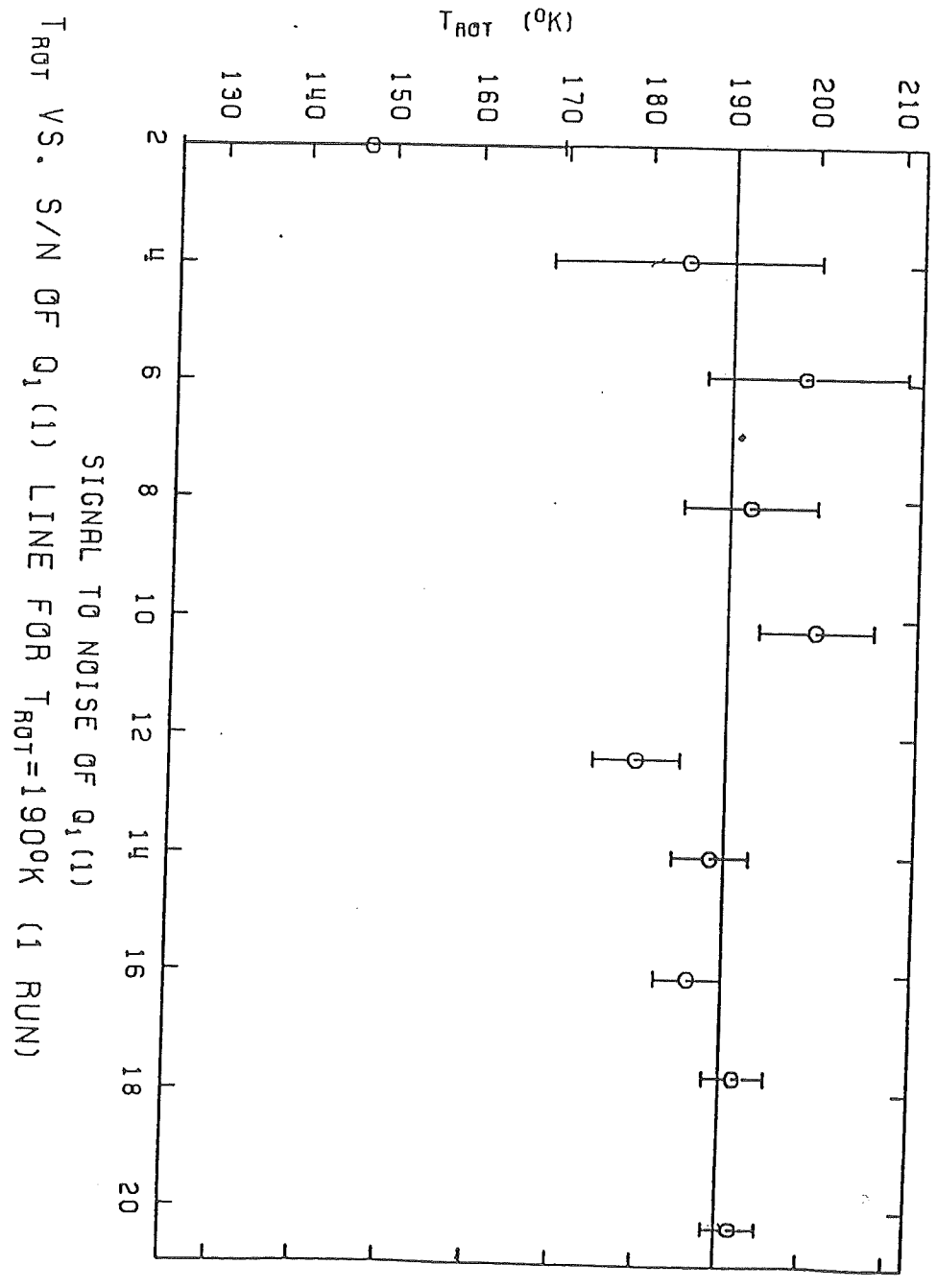
Tg = Delay between the last data point and interrupt. Depends on the speed and number of sample/fringe (1 or 2).

Th = Half scan requisition time.
Depends on the speed and resolution.

Ti = Turn around 150 ms

APPENDIX B:

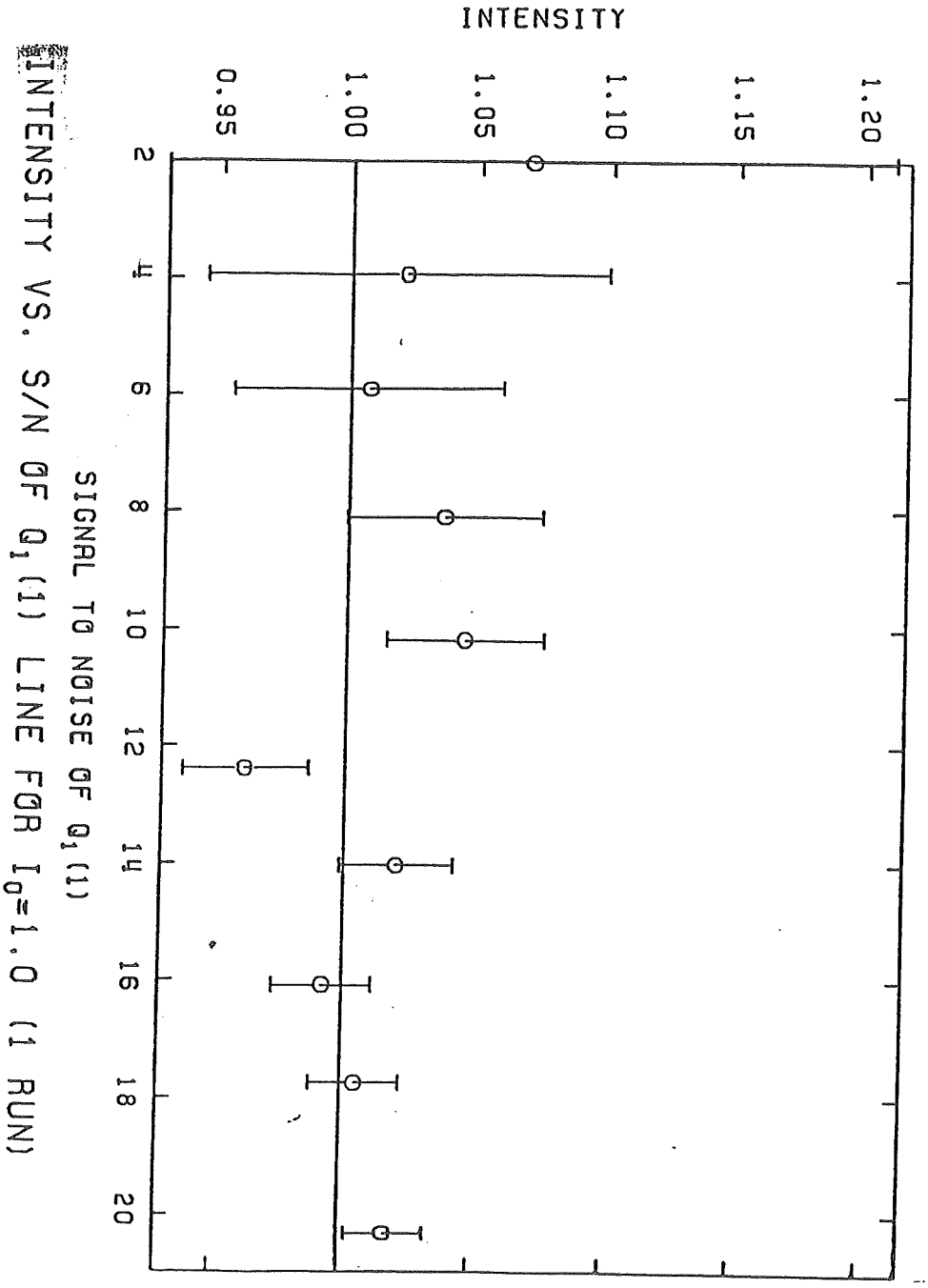
Error Bars Of Rotational Temperature For Various Signal To Noise Ratios



T_{ROT} VS. S/N OF Q_1(1) LINE FOR T_{ROT}=190^{0}K (1 RUN)

APPENDIX C:

Error Bars Of Intensity For Various Signal To Noise Ratios



INTENSITY VS. S/N OF $Q_1(1)$ LINE FOR I_o=1.0 (1 RUN)