/COMPUTER CONTROLLED DEEP LEVEL TRANSIENT SPECTROSCOPY SYSTEM/

205

by

HEMANT MEHTA

B.E. (Hons) EEE

Birla Institute of Technology and Science
Pilani, India

1983

A MASTER'S REPORT

submitted in partial fulfillment of

the requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

Durland Hall

Kansas State University

Manhattan, Kansas 66506

1986

Approved by:

Andrzej Rys

Major Professor

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I, hereby express my sincere gratitude to my major advisor, Dr. Andrzej Rys, for his excellent guidance, constant encouragement, and firm support.

Sincere thanks are due to Dr. M. S. P. Lucas and Dr. Alvin Compaan for serving on my advisory committee.

I also would like to express my sincere thanks to my friends at the Department of Electrical and Computer Engineering for their support throughout the process of working on this project.

Most of the material in chapters one and two is taken from the first two references (papers by D. V. Lang and Wagner, Miller and Mars). My thanks are due to the authorts of the papers.

List of Figures

Figures (1) to (6) curtesy of D. V. Lang, Deep Level Transient Spectroscopy: A new method to characterize traps in semiconductors. J. Appl. Phy. 45, 3023 (1974).
Figure (7) curtesy of Wagner, Hiller, Mars, Fast digital apparatus for capacitance transient analysis. Rev. Sci. Instrum. 51(9), 1205 (Sept. 1980).

CHAPTER ONE

1.1 INTRODUCTION

A large variety of experiments in semiconductor physics rely on the analysis of transient phenomena resulting from the return of a system to equilibrium. They have in common the existence of (ideally) exponential transients. In some fields the analysis of the transients is done by direct evaluation of the transients with regard to the time of decay; other fields make use of correlation-spectroscopic methods, if the dependence of the time constant on another experimental parameter is of interest. One of the latter areas is the analysis of deep levels (DL) in semiconductors, experiments where the correlation processing is usually performed on line by analog instruments.

The study of DL in semiconductors has become a widely used method and an important analysis. One approach to the investigation of DL is the analysis of transient phenomena caused by the slower response of DL to abrupt changes of potential conditions. Thus the thermal emission of charge carriers from DL into the corresponding band can be monitored, e.g., by recording the depletion layer capacitance of a diode vs. the time after electrical DL filling pulse.

Some early work on junction capacitance measurements was based on the direct recording and analysis of capacitance transients and was thereby restricted to very slow transients. In 1974, D. V. Lang of Bell Labs suggested a relatively simple technique called "Deep Level Transient Spectroscopy" (DLTS) which has become a commonly used method to investigate deep impurities. Lang's method is similar to correlation technique and is based on scanning the sample temperature (and thereby changing the DL transient time constant) under a constant decay time constant of the measuring apparatus. The strength of DLTS is mainly the ease of the spectral analysis, since a maximum or minimum always occurs when sample transient matches the time constant of the apparatus. In the spectra which shows the DLTS signal as a function of the sample temperature therefore only the temperature of the extremum has to be determined. (Chapter two discusses the DLTS in more detail.)

On the other hand, the measurement of the spectra is not optimized. The DLTS signal is the difference between two readings of the capacitance transient taken at two different delay times after filling pulse, and usually the signal is transferred directly to a strip chart recorder. The relevant DLTS data are taken from an Arrhenius plot, and one temperature scan provides one data point per DL in that plot. Thus several scans have to be made to characterize the DL. The presence of many DL considerably complicates this method. This means that due to the loss of the original capacitance values, readings even with the same delay time may have to be repeated in the measurement runs to obtain the data for adjacent time constants. A temperature scan in DLTS measurements takes typically between one quarter of an hour and two hours. The heating or cooling of the sample cannot be done much faster for two reasons. Firstly, the sample temperature has to be measured accurately by means of a necessarily remotely mounted temperature detector. Thus, the measurement becomes less accurate during fast temperature changes. Secondly, a possible dependence of the sample equilibrium capacitance on temperature causes the transients to be measured superimposed on a base line not constant in time, which can distort the signal and which is more pronounced at a higher heating rate.

Thus it is desirable to design a system which is able to acquire more data during a temperature scan than conventional DLTS does while keeping the advantage of simple analysis of the correlation-type spectra. Two requirements have to be fulfilled for a system to provide these features:

(i) the data acquired on line have to be stored, and therefore

(ii) the measurement has to be done digitally.

The system described here is based on direct recording of the transients rather than on scanning the temperature under fixed apparatus time constants. Thus acquisition of the capacitance data is separated from signal processing which is done by the computer afterwards. The entire measurement is under computer control and the experimental data are stored for the analysis. In addition to the benefit of automation, this measurement concept reduces the total measurement time considerably.

CHAPTER TWO

2.1 THEORY OF DLTS

In an effort to characterize the traps present in a semiconductor, a preferred technique would be the one which is sensitive, rapid, and straightforward to analyze. It should be able to distinguish between majority- and minority-carrier traps. In addition the technique should be spectroscopic in the sense that the signals due to different traps be resolved from one another and to be reproducible in position when plotted against a single variable.

The most promising of the new techniques use the capacitance of p-n junction. But most of these techniques have lacked either the sensitivity, speed or spectroscopic nature. One such capacitance transient thermal scanning technique which has all of the above features is called "Deep Level Transient Spectroscopy" (DLTS) and is a high frequency junction capacitance technique.

Briefly, the DLTS measurement system consist of a sensitive capacitance measurement apparatus with good transient response, a pulse generator to make rapid changes in the diode bias, a signal integrator, a X-Y recorder, and a variable temperature cryostat. The presence of each trap is indicated by a peak on a flat baseline plotted as a function of temperature. The heights of these peaks are proportional to their respective trap concentrations, the sign of each peak indicates whether it is due to a majority- or minority-trap, and the positions of the peaks are simply and uniquely determined by the integrator settings and by the thermal emission properties of the respective traps. By the proper choice of experimental parameters it is possible to measure the thermal emission rate, concentration profile, etc. of each trap.

2.2 PULSED BIAS CAPACITANCE TRANSIENTS

This technique is used to obtain information about an impurity level in the depletion region of a p-n junction by observing the capacitance transient associated with the return of thermal equilibrium of the occupation of the level following an initial nonequilibrium condition. One can measure the time constant of this transient as a function of temperature and obtain the energy level. The initial magnitude of the transient is related to the concentration of the trap. The form of technique used in DLTS makes use of one or more voltage pulses applied to the sample in order to define the initial conditions. Here for discussion, we will consider an example of n^+-p junction.

Figure (1) is a schematic summary of the emission and capture processes which characterize a particular trap. As shown in figure (1), a capacitance change is caused by using a bias pulse to introduce carriers, and thus changes the electron occupation of a trap from the steady-state value. As this population returns to equilibrium, the capacitance returns to quiescent value. The transient is an exponential function of time with a rate constant equal to emission rate of trap fillers. The sign of the capacitance change depends on whether the electron occupation of the trap had been increased or decreased by the pulse. An increase in trapped minority carriers causes an increase in the junction capacitance. As indicated in the figure, the capacitance transient due to a minority-carrier trap is

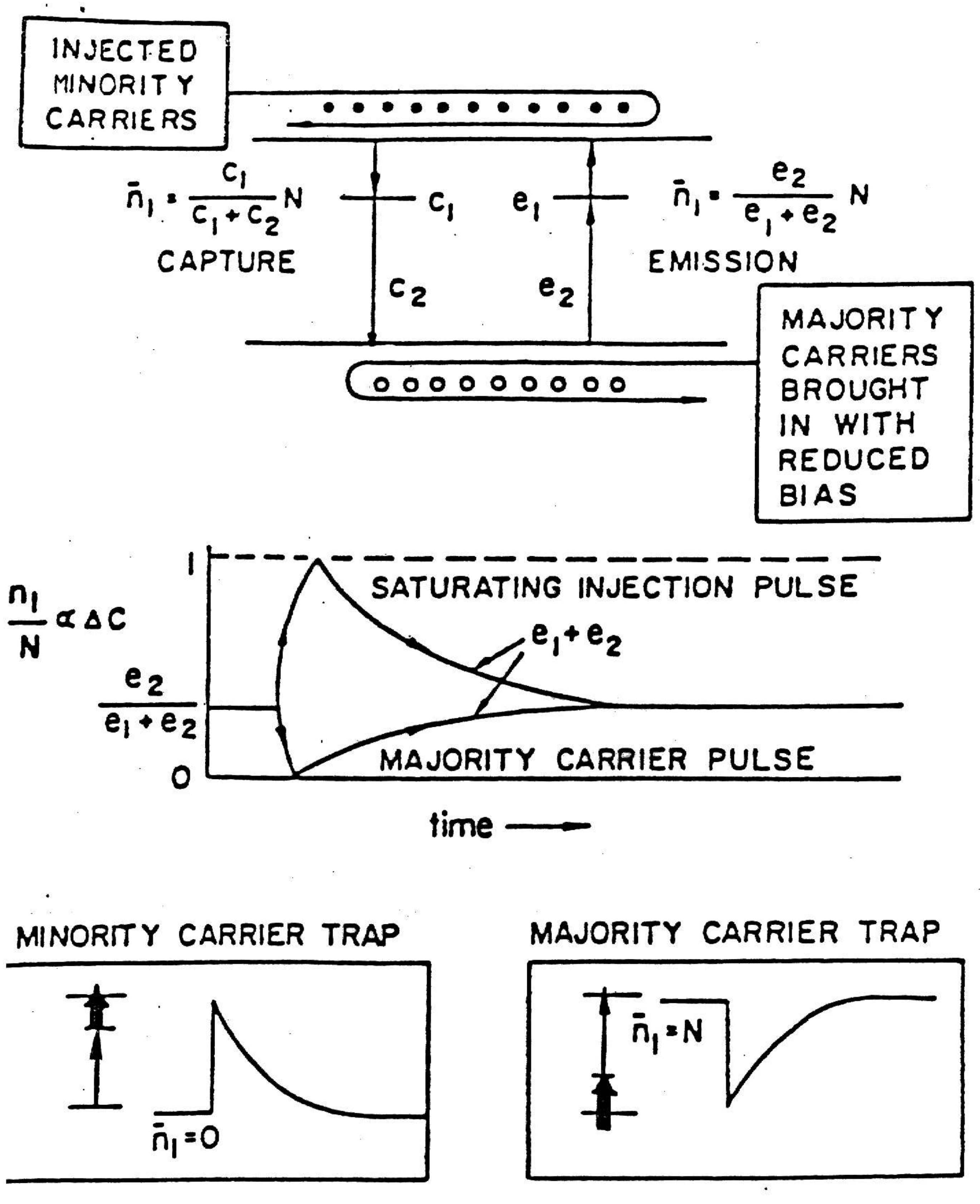

INJECTED MINORITY CARRIERS
$\bar{n}_1 = \frac{c_1}{c_1+c_2}N$
CAPTURE
c_1
c_2
e_1
e_2
$\bar{n}_1 = \frac{e_2}{e_1+e_2}N$
EMISSION
MAJORITY CARRIERS BROUGHT IN WITH REDUCED BIAS
$\frac{n_1}{N} \propto \Delta C$
1
$\frac{e_2}{e_1+e_2}$
0
SATURATING INJECTION PULSE
e_1+e_2
MAJORITY CARRIER PULSE
time
MINORITY CARRIER TRAP
$\bar{n}_1 = 0$
MAJORITY CARRIER TRAP
$\bar{n}_1 = N$

Fig 1. Schematic summary of Emission and Capture Processes

always positive and is induced by injected minority carriers, whereas the transient due to a majority-carrier trap is always negative and is induced only by majority carriers.

There are two types of bias pulses, namely, an injection pulse which momentarily drives the diode into forward bias and injects minority carriers into the region of observation, and a majority-carrier pulse which momentarily reduces the diode bias and introduces only majority carriers into the region.

In all the situations, the capture rates of the carriers are much larger than their corresponding emission rates, which can be neglected during the bias pulse. An injection pulse which introduces a large enough number of electrons so as to make capture rate of minority-carriers much greater than capture rate of majority carriers, will completely fill the trap with electrons. Such a pulse is called a saturating injection pulse. A majority-carrier pulse, on the other hand, introduces only holes, and thus tends to empty all traps of electrons, i.e., fill them with holes.

Figure (2) is an injection pulse sequence which is used to produce a capacitance transient for the case of a minority-carrier (electron) trap, while Figure (3) shows the analogous majority-carrier pulse sequence for a majority-carrier (hole) trap. Both Figures (2) and (3) show the depletion region, capture and emission processes, and trap electron occupation before, during, and after the appropriate bias pulse. Figure (4) is a schematic illustration of the time dependence of the capacitance transients.

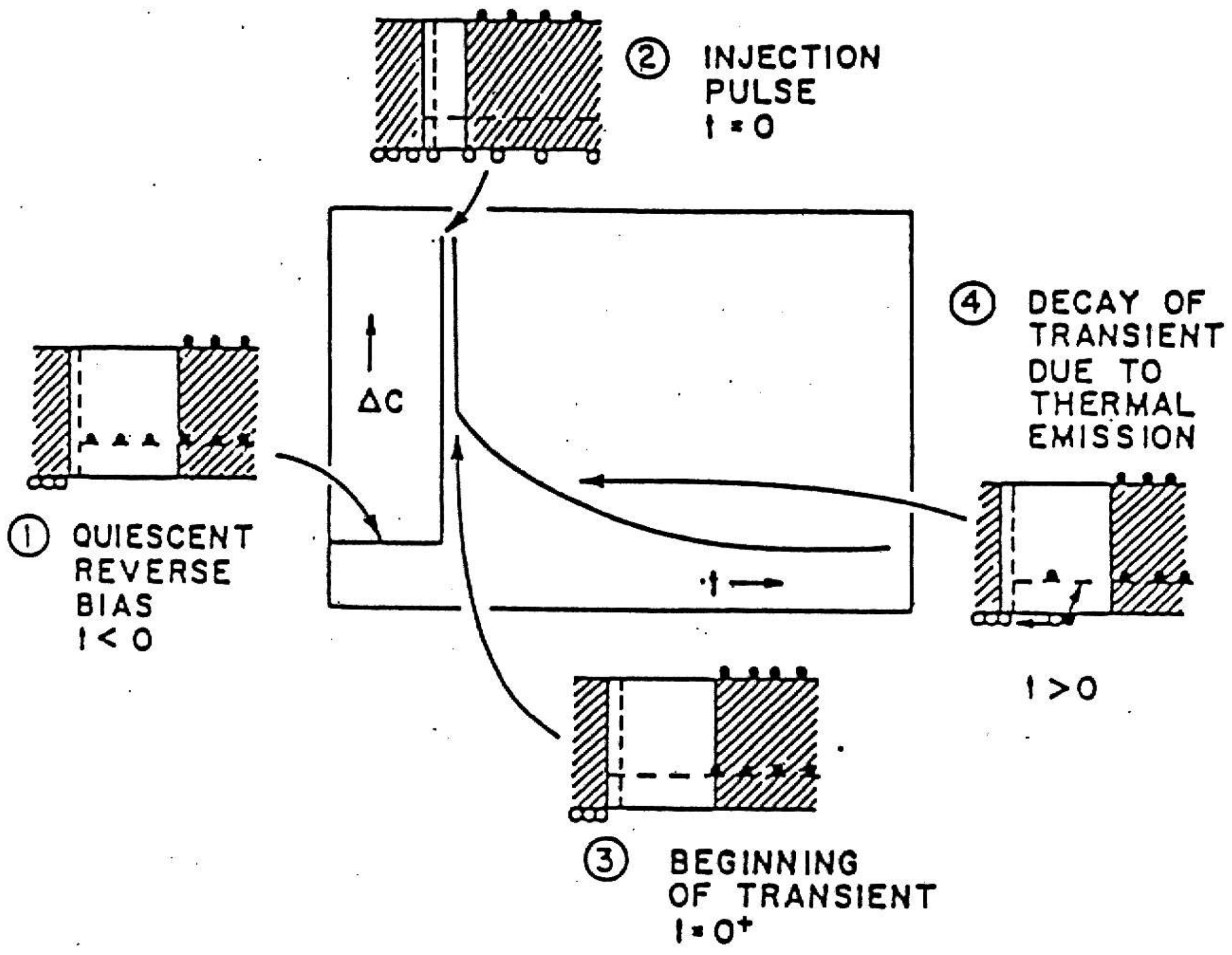

② INJECTION PULSE t = 0
④ DECAY OF TRANSIENT DUE TO THERMAL EMISSION
ΔC
① QUIESCENT REVERSE BIAS t < 0
t
t > 0
③ BEGINNING OF TRANSIENT t = 0⁺

Fig 2. Injection Pulse Sequence

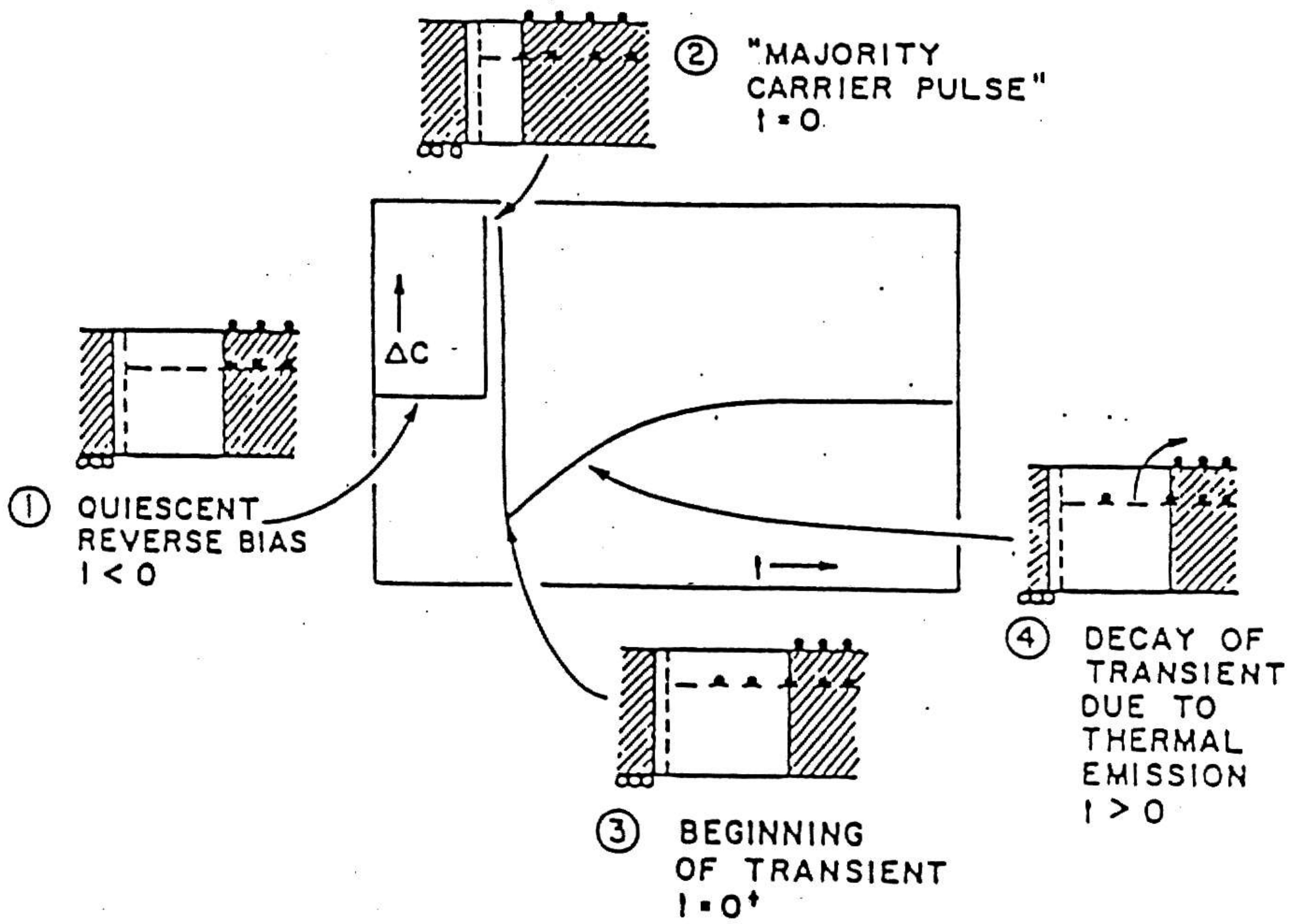

② "MAJORITY CARRIER PULSE" t = 0
ΔC
① QUIESCENT REVERSE BIAS t < 0
t ⟶
④ DECAY OF TRANSIENT DUE TO THERMAL EMISSION t > 0
③ BEGINNING OF TRANSIENT t = 0⁺

Fig 3. Majority Carrier Pulse Sequence

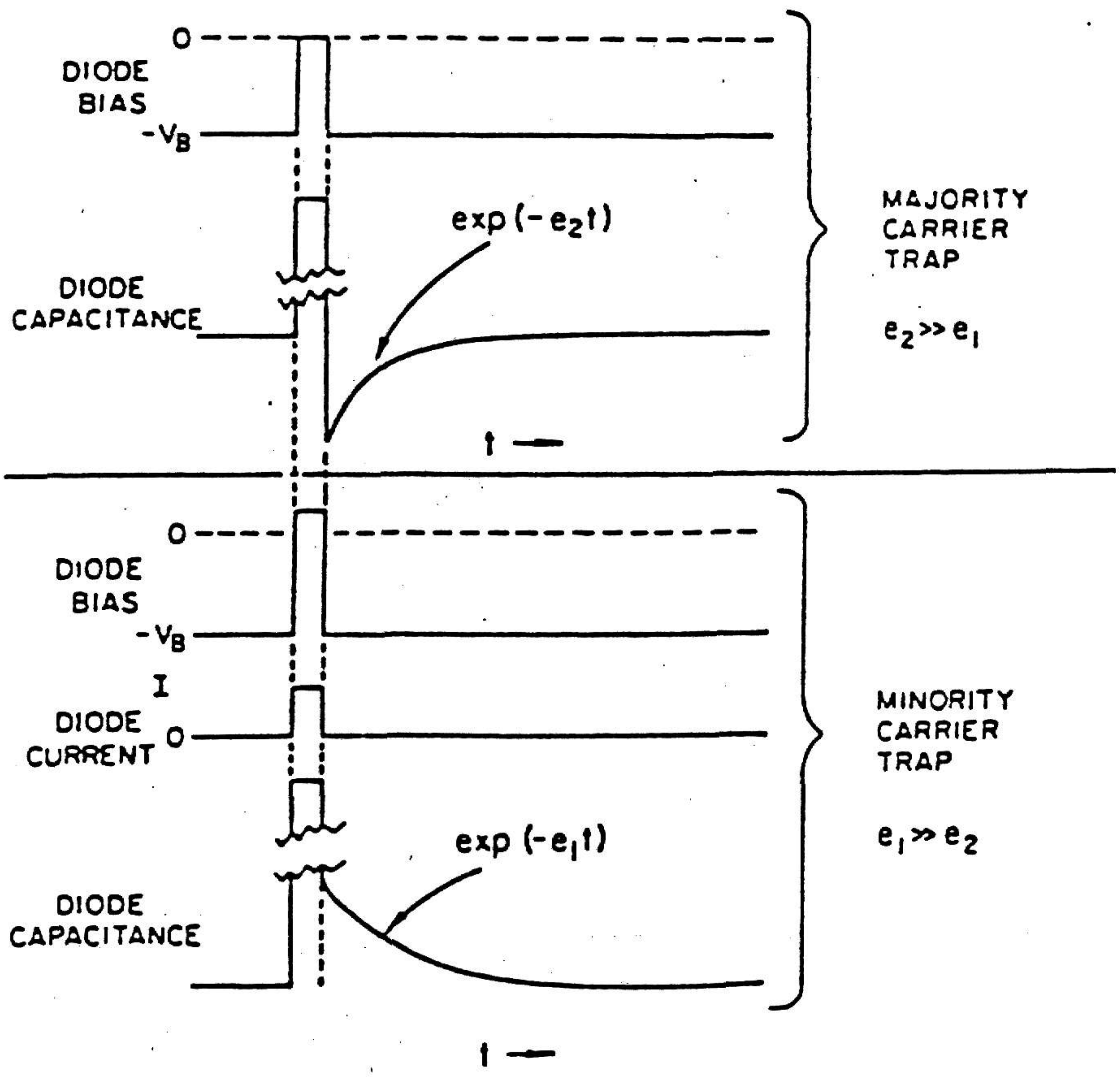


Fig 4. Time dependence of Pulse Biased Capacitance Transient

2.3 DEEP LEVEL TRANSIENT SPECTROSCOPY

The physics of DLTS is in the capacitance transients as described above. The primary contribution of DLTS scheme as presented here lies in the convenience and speed with which the information inherent in a series of capacitance transient experiments may be obtained. That is to say, the information which may be obtained by several 10-minute DLTS thermal scans could also be obtained from a more tedious and time consuming point-by-point observation of the capacitance transients at many different fixed temperatures. This latter alternative may be helpful in the detailed study of a unknown trap since it can be more precise, but in the initial characterization of many traps in an unknown sample or in high-volume monitoring works the DLTS scheme has distinct advantage.

The essential feature of DLTS is the ability to set an emission rate window such that the measurement apparatus only responds when it sees a transient with a rate within this window. Thus, if the emission rate of a trap is varied by varying the sample temperature, the instrument will show a response peak at the temperature where the trap emission rate is within the window (Figs. 5 and 6).

2.3A TRANSIENT ANALYSIS AND DEEP LEVEL PARAMETERS

The information DLTS provides about DL is the depth of the level with respect to the corresponding band edge, the capture cross-section of the DL and its concentration in the crystal. The first two values are derived from the time constant of the transient signal and from its temperature dependence.

BASIC IDEA OF DLTS METHOD

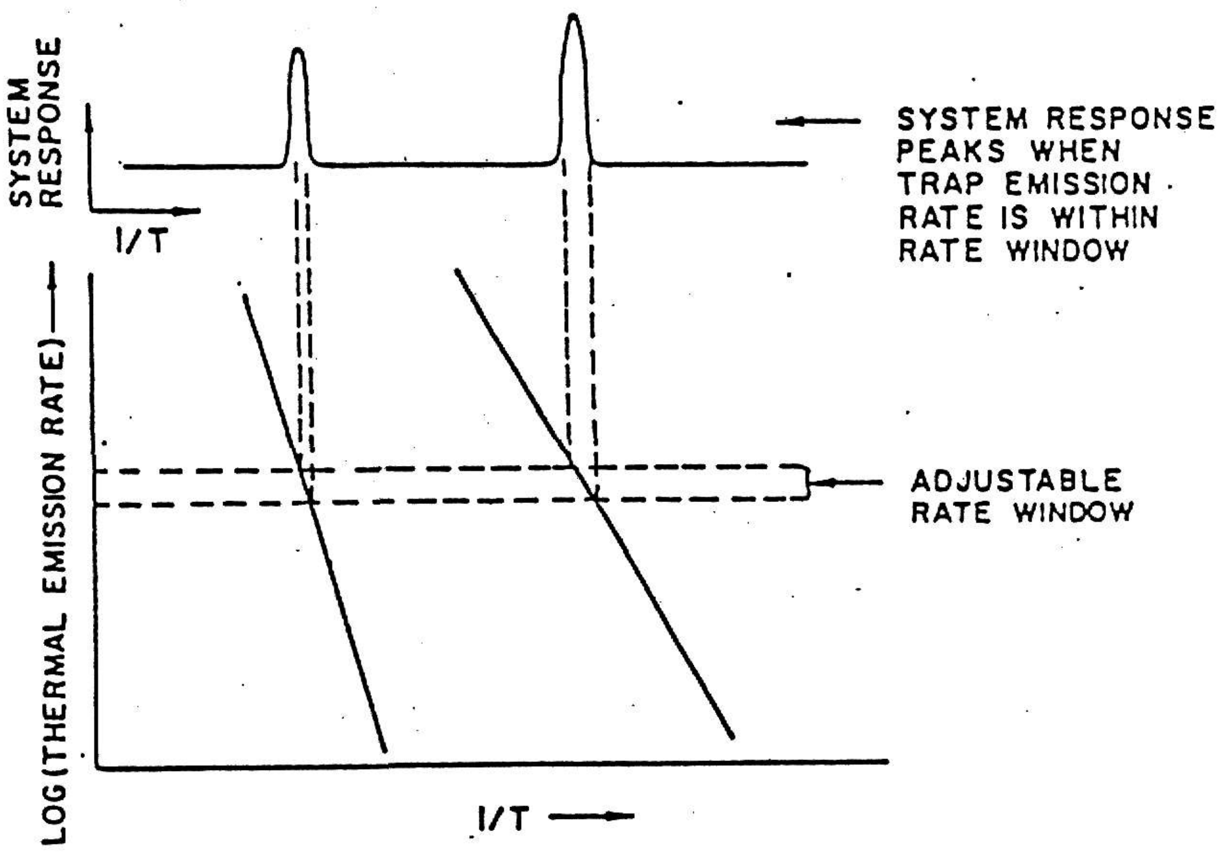

SYSTEM RESPONSE
1/T
SYSTEM RESPONSE PEAKS WHEN TRAP EMISSION RATE IS WITHIN RATE WINDOW
LOG(THERMAL EMISSION RATE)
ADJUSTABLE RATE WINDOW
1/T

Fig 5.

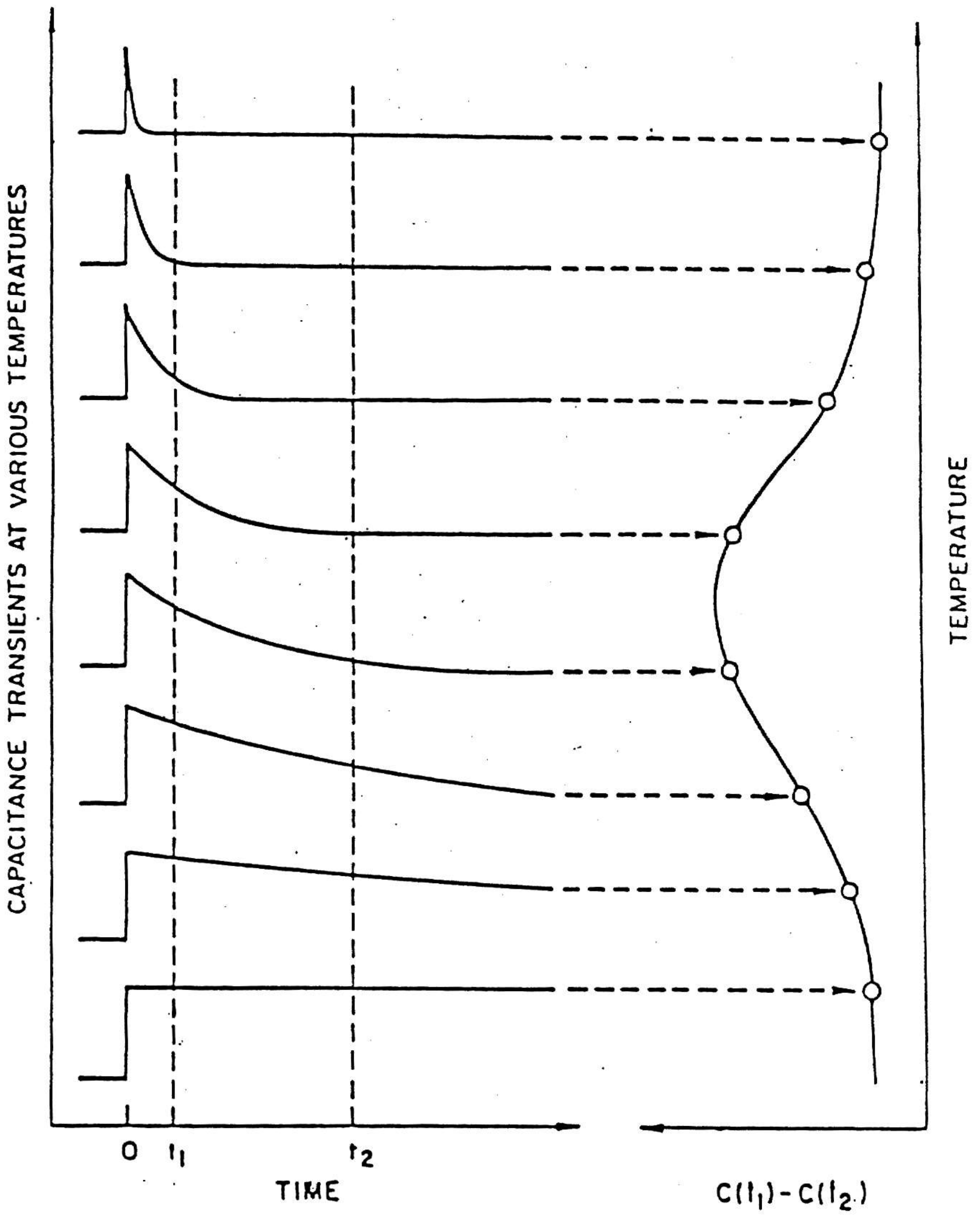


Fig 6. Concept of DLTS Spectrum

Time constant vs. temperature dependence is evaluated by the spectra analysis. The spectra shows extrema whenever the transient or a contribution to it has the time constant τ_t, where

$$\tau_t = (t_1 - t_2)/\ln(t_1/t_2), \tag{1}$$

and where t_1 and t_2 are the two delay times at which the capacitance readings were taken.

The transient time constant can be expressed as

$$\tau_t(T) = \tau_0 \exp(E_t/kT), \tag{2}$$

where

$$\tau_0 = 1/(\sigma\langle v\rangle N_c)$$

where E_t being the trap depth, k the Boltzmann constant, T the sample temperature, σ the capture cross-section of the DL, $\langle v\rangle$ the mean thermal carrier velocity, and N_c the effective density of states of the corresponding band.

The emission rate

$$e = \gamma\ \sigma\ T^2 e^{-\Delta E/kT}, \tag{3}$$

where

$$e = 1/\tau_t$$

and

$$\gamma = \langle v\rangle N_c/T^2$$

simplifying equations 1, 2 and 3 together gives

$$\ln(eT^{-2}) = \ln(\gamma\sigma) - 1000\Delta E/1000kT$$

If we put (represent) this equation in the form Y = A+BX, we have the following results

$$A = \ln(\gamma\sigma)$$

and

$$B = -\Delta E/1000k$$

if we choose

$$X = 1000/T \quad \text{and} \quad Y = \ln(eT^{-2}).$$

Taking into account the temperature dependence of the $\langle v \rangle$ and N_c, one can use an Arrhenius plot for data analysis where $\ln[eT^{-2}]$ is plotted vs. inverse sample temperature 1000/T. The slope of the curve gives then the thermal activation energy as

$$\Delta E = -1000kB$$

and the intercept is determined by the capture cross-section as

$$\sigma = e^A/\gamma,$$

assuming that E_t is temperature independent. γ values are precalculated and are shown in table (1) for silicon and galium arsenide for the cases of n-type and p-type.

The height of the transient signal on the other hand provides information on the concentration of the DL. In the approximation of a DL concentration N_t much smaller than the concentration of shallow impurities N_s,

$$N_t = 2N_s\Delta C/C_0$$

with ΔC amplitude of the transient just after the trap filling pulse ($t = 0$), C_0 equilibrium capacitance value.

2.3B TRANSIENT RECORDING AND SIGNAL PROCESSING

A variety of techniques have been developed to measure and analyze the capacitance transients and their temperature dependence. These techniques have in common that they analyze the transient with respect to a particular time constant. Therefore the output is maximum if the sample time constant matches the apparatus time constant. As all these techniques make use of analog instruments, the primary

	γ_n	γ_p
Si	7.20 E+21	2.94 E+21
GaAs	2.28 E+20	1.70 E+21

Table 1. Gamma Values

information, i.e., the capacitance signal, is lost during the signal processing, and the output is either the difference between readings or convoluted signal. These techniques may have advantage in simplicity or low cost, but they do not provide superior raw data or have fundamental physical advantage over a technique which uses capacitance readings as raw data.

The concept of the DLTS apparatus described here is to record the entire transient at each temperature and store the sequentially acquired transient data for analysis. Thus, only one temperature scan is necessary to acquire the entire set of data for an Arrhenius plot. As the data are acquired at least partially on line with the transient, the average data acquisition rate is much higher than in conventional DLTS. The principle is also better adapted to automation since one temperature scan can be more easily automated and adjustment have to be made only on electronic instruments during the measurement run.

Retrieval of the stored data allows the generation of conventional DLTS spectra. Also, it is possible to apply a software correlation with a theoretical exponential decay function.

2.4 CAPACITANCE TRANSIENT MEASUREMENT

The automated DLTS system consists basically of three parts: the instruments which handle the sample bias condition and measure the capacitance, the part which takes care of the measurement and control of the sample temperature, and the instruments for system control, data storage, display and documentation.

In more detail, the capacitance measurement part includes:

1. A pulse generator, which outputs the sample reverse bias and the filling pulse,
2. The capacitance measurement module having an analog voltage signal output.
3. A fast triggerable digitizing unit.

Figure (7) shows various blocks of the DLTS measurement system.

The various tasks mentioned above are performed by various instruments like a pulse generator from HP, DRC-81C temperature controller from LAKE SHORE CRYOTRONICS, BOONTON capacitance meter and 7912AD Digitizer from TEKTRONIX. The next chapter covers the explanations of some of the above mentioned instruments.

2.5 MEASUREMENT PROCESS AND CONTROL

The sample is mounted on a header and is attached to the sample holder of the liquid nitrogen/helium cryostat. Initial set up consists of adjusting the bias pulse, setting the pulse rate, pulse wdith, trigger pulse for digitizer and feeding the information to the computer regarding initial and final temperatures, heater controller set up, digitizer set up, etc. After the sample is cooled down to the desired temperature, the entire measurement process is controlled by the computer. The experimental parameters are defined by the following values:

i) reverse bias voltage;
ii) voltage of DL filling pulse;
iii) width of filing pulse;

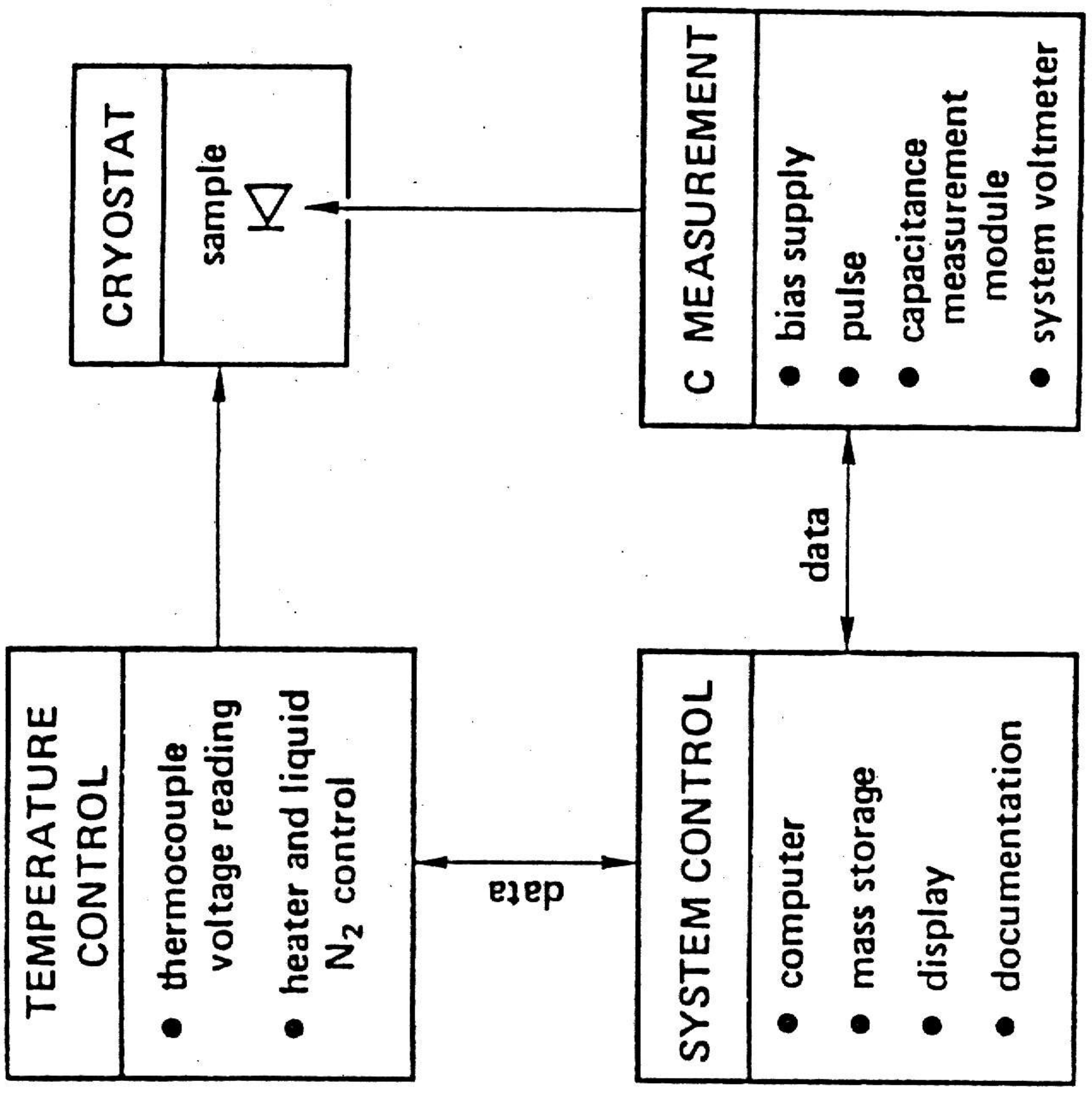


Fig 7. System Block Diagram

iv) delay time;

v) minimum and maximum temperatures;

vi) number of readings as a function of time;

vii) number of readings as a function of temperature;

viii) number of readings for signal averaging.

The number of readings as a function of time means number of points digitized for a transient. Now knowing that analog to digital conversion takes place at a fix rate, we are talking about the time period over which a transient is digitized. The number of readings as a function of temperature means number of data points on an Arrhenius plot per degree Kelvin. The number of readings for signal averaging means the number of transients acquired consecutively and averaged to present a single point for a given temperature on conventional plot.

We usually take 32 (or 64) readings for signal averaging and every reading consists of a digitized transient over a period of 10 msec and in this time period 512 digital points are obtained for the signal transient. All the data together with the corresponding temperature readings, are stored on the disk. The process takes around two and half to three hours to go over a temperature range of 40K to 330K at the temperature rise rate of two degrees per minute, to acquire all the data needed for Arrhenius plot. This compares with one data point per two hour obtained in the conventional DLTS apparatus using analog equipment. Thus this system is considerably faster than conventional analog DLTS, assuming the acquisition of the same amount of information.

2.6 DATA ANALYSIS

The data array in the computer storage describes completely (within the given windows in time and in temperature) the response of the sample to the abrupt change in bias vs. the time after the change vs. the sample temperature. There are various ways to analyze these dependences, implemented in the software. The following subsections discuss some of these ways.

2.6A DIRECT ANALYSIS OF THE TRANSIENTS

As transients are recorded directly, there is a possibility to analyze them with regard to exponential contributions. It has shortcomings of only being accurate in the presence of one level. Thus we do not make use of this technique. However, the display of direct transient data vs. time is used to see the system performance against present noise levels.

2.6B CONVENTIONAL DLTS SPECTRA

An ordinary DLTS spectrum is generated by subtracting readings at two different delay times and plotting as a function of temperature. The resulting action is equivalent to what we obtain by means of conventional analog DLTS measurements. From this set of information, the extrema and their corresponding temperatures are found to generate the Arrhenius plot of $\ln[eT^{-2}]$ vs. 1000/T, which then gives information about capture cross-section and activation energy.

CHAPTER THREE

3.1. HARDWARE SETUP

As mentioned earlier in chapter two, the hardware of the system consists of the following instruments:

1. A pulse generator;
2. Capacitance meter with analog output voltage;
3. Fast digitizing instrument -- a digitizer;
4. Temperature controller;
5. Cryostat with sample holder;
6. Personal computer system;
7. X-Y display unit;
8. TV screen.

In the system described here, a Z-158 computer with a hard disk is used. The pulse generator is HP8015A. It has two pulse outputs with adjustable lower and upper levels, pulse width, rise and fall times, delay period and a trigger output of the same frequency as pulse outputs. At the beginning of the process, the pulse generator is set up to provide required reverse bias voltage and filling pulses. The pulse rate is adjusted to one pulse every 30 milliseconds. The low level of the pulse is set at the required reverse voltage so that we need not apply separate bias voltage.

The capacitance is measured using a capacitance meter BOONTON model 72B, which has an analog output voltage and has a fast response time to any capacitance change at the input.

The analog output of the capacitance meter is fed to a fast digitizing unit which is, in our case, 7912AD digitizer from Tektronix. Digitizer is a wide bandwidth, high speed waveform acquisition instrument which can be remotely controlled using an IEEE 488 interface. When in digital mode, it can acquire a repetitive waveform within a time window of 5 nsec to 10 msec and output it on an IEEE bus. Simultaneously, it can display the waveform acquired on X-Y-Z display so we can see what we are digitizing. It has external trigger input to synchronize the data acquisition operation with some external event, like in the present case in which it is started at the time when the filling pulse is removed.

The temperature controller is model DRC-81C from Lake Shore Cryotronics. It can control the temperature within one tenth of a degree. For controlling action, it uses what is called as "Three-Mode" control.

The next two sections deal, in brief, with the functioning of the two of important instruments namely digitizer and temperature controller.

3.2 DIGITIZER

The digitizer has two operating modes. In the DIGITAL model, it digitizes either a single shot or a repetitive waveform and stores it for internal processing or for output on the IEEE bus. The analog outputs are used to drive a XYZ display unit to show the digitized waveform. In the TV mode, it converts the input signal to a composite video output. This allows the input waveform to be displayed on a TV monitor.

In either mode, digitizer can acquire the waveforms with high bandwidths. The time window can be selected between 10 msec and 5 nsec. This is equivalent to sampling rate (in digital mode) from 50 KHz to 100 GHz.

In digital mode, it begins to store the data on the next time base trigger after either the digital button is pressed from front panel (in local mode) or a digitize command is received (in remote mode). The signal is scanned top to bottom and left to right. Two data values are stored for every vertical scan, one for the top of the trace and one for the bottom of the trace. These data are stored in pairs of top and bottom. From left to right it scans 512 vertical arrays and thus generates 512 points in the time window.

WAVEFORM STORAGE: Data values are stored in various arrays like raw data array and processed data array. The raw data array, as the name suggests, contains raw data whereas the processed data array contains the data after some internal processing. The kind of processing performed on the data is discussed later in the section.

XYZ DISPLAY: The digitizer automatically switches the XYZ display to show the results of the last digitize operation or waveform processing operation. For instance in the local mode when the digitizer is switched to digital mode, the XYZ display shows whatever is digitized as soon as the operation is complete. Under remote control, waveform processing operations can be called. These include digitize and signal average (SA) or average-to-center (ATC). The display shows the results of processing as soon as it is complete. If the digitize and signal average operation is called, for example, the signal averaged waveform is displayed automatically.

The XYZ display mode can be changed under remote control. For instance, the display can be changed from that of the last waveform digitized to show a signal averaged waveform previously acquired if it is still in the memory. Or the display can be turned off.

Whatever XYZ display was called under remote control, the XYZ display automatically changes to show the results of the last digitize or waveform processing operation that follows.

PROGRAMMING THE 7912AD

The digitizer can be operated by remote control over the bus specified in IEEE 488 standards. After it is set to operate in remote mode by the system controller, front panel operating controls are disabled and their functions can be set with mnemonics sent to it in ASCII over the bus.

Data can be output by digitizer at the maximum rate of 710 kB per second. Waveform data is output in binary rather than in ASCII. This enables greater throughput, in that data is moved faster in fewer bytes and requires less bus time.

Some of the operations it performs for data acquisition and/or on data acquired are of importance from the user point of view and are mentioned here in brief (for details, please refer to the instrument manuals).

DIGITIZE AND SIGNAL AVERAGE (SA) a repetitive waveform. The digitize data operation is performed the specified number of times. Each time ATC (average-to-center, discussed later) algorithm is performed on the raw data. The resulting ATC data are summed in the SA array. Since the ATC operation sums each pair of vertical data values (top and bottom edges of trace) from the raw waveform data, the signal average algorithm performs a divide-by-two operation on the data after all waveforms have been summed in SA array. This also prevents the summed data values in SA array from setting bit 16 of the dataword, making later processing easier.

When the operation is complete, the last waveform acquired resides in the raw data area and its simple ATC in the ATC array. No division is performed to scale the SA data by the number of averages performed. Since all averages are to be performed in integer power of two, an implied binary point is used to scale the data to values from 0 to 511.

ATC: The average-to-center command causes the instrument to process the raw data to obtain a simple average of the top and bottom edges of waveform. The ATC routine sums the highest and the lowest values, point-by-point from the raw vertical data. The routine fills the missing points by linear interpolation, i.e., missing points are set equal to the nearest data points. The effect of ATC command is shown in fig (8). Because the values are summed top to bottom, they range from 0 to 1023. However, a block binary point is assumed between bit 0 and 1 for scaling purposes.

WAVEFORM DATA I/O

In general, the waveform data are transferred in order from left to right of the trace. For transmission of data, block binary format is used. The block binary format is:

% <BYTE COUNT>[<DATA VALUE><...>]<CHECK SUM>;

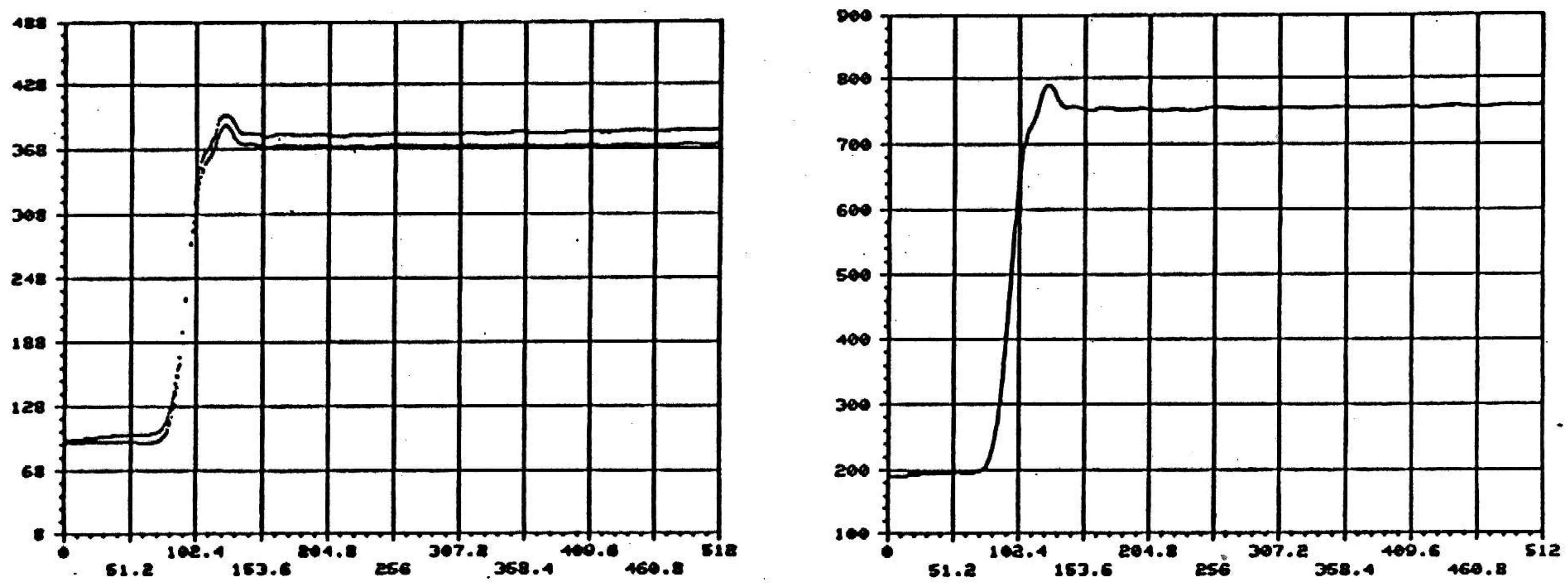


Fig 8. Concept of Average-to-Center

where

% is the ASCII percent character indicating beginning of a binary block

BYTE COUNT is a 16-bit binary number indicating how many bytes remain to be transmitted in the block including the check sum, but not including the message delimiter. It is sent in two bytes, more significant byte first.

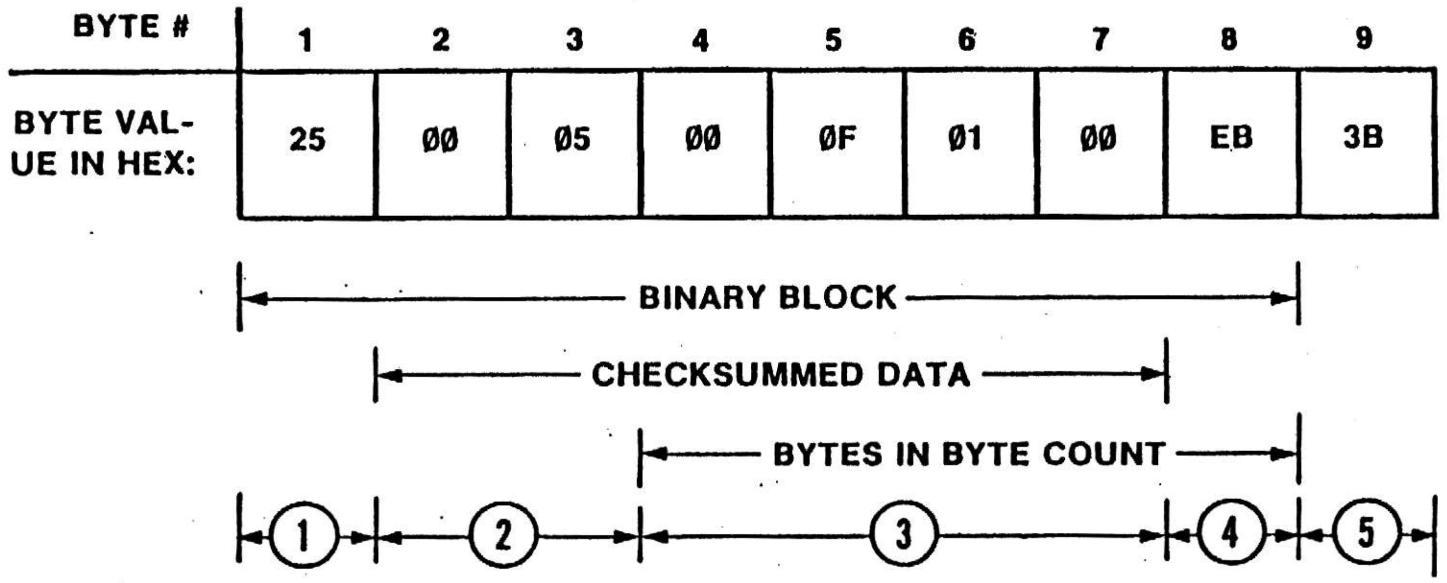
DATA VALUE is a 16-bit binary number sent in two bytes, more significant byte first. If the data value is less than 16 bits in length (which is always the case here), it is sent right-justified with unused bits set to zero.

CHECK SUM is an eight-bit binary number sent in a single byte. It is computed by taking the modulo-256 sum of all the preceding bytes in the block except %, that is, by summing in eight bits the value of preceding bytes and throwing away the carry. The sum is converted to 2's complement before transmission. This allows the receiver to compute the sum in same manner and thereby check that it is zero after the check sum byte is received.

; is the ASCII semicolon character delimiting the message unit. The binary block for a simple two-value array is shown in fig (9).

SCALING THE OUTPUT WAVEFORM

The ATC array or normalized arrays can be scaled using ground reference and vertical scale factor. In case of ATC arrays, values should be divided by two before scaling.



(1) Binary parser = ASCII code for %.

(2) Byte count = 5 (5 bytes follow, including checksum).

(3) Data values = 15 and 256, decimal (each requires two bytes).

(4) Checksum = 2's complement of the modulo-256 sum of all preceding bytes except the parser (%). Thus the modulo-256 sum of all bytes except the parser and delimiter should equal zero (ØØ + Ø5 + ØØ + ØF + Ø1 + ØØ + EB = ØØ).

(5) Message unit delimiter = ;.

Fig 9. Data Output Format for Digitizer

First, the ground reference is acquired by digitizing the waveform with signal input grounded. Then it is normalized using ATC command and divided by two. Then ground level can be computed by taking the average value of a portion of the center of the waveform. Then the vertical scale factor can be read from digitizer.

Now data can be scaled point-by-point by subtracting the ground reference, multiplying by the scale factor (volts/division), and dividing by the number of data points per division.

3.3 TEMPERATURE CONTROLLER

Here, we will talk a little about the principle on which this controller works. The so called "Three-mode" or "PID" controller, utilizing proportional, integral and derivative functions is discussed in this section.

For proportional control, the desired temperature setting is compared to the sensor signal and the difference, or error signal is amplified. When the sensor temperature corresponds to set point, the sensor signal will be equal to but opposite in polarity to the set point signal and error signal will be zero. Modern controllers like the one used here have stored in them the appropriate voltage-temperature sensor characteristics so that the set point is calibrated directly in temperature. However, this convenient feature compromises the resolution and accuracy of the controller.

The output of the controller is dc power to a resistive heater, the output magnitude of which depends on the size of the error signal and its sign. Hence it is evident that power output is "proportional" to magnitude of error signal.

In order to maintain the desired temperature, some kind of power output is necessary which requires a non-zero error signal or what we call as temperature offset. It is proportional to power output level. This offset can be reduced or sometimes eliminated by providing the required power output without the need of an error signal to drive output stage. A circuitry is added to the controller that senses that there is a steady state offset signal and which makes a bit-by-bit addition to the power output, proportional to the magnitude of offset and continues the corrective action until offset is reset or zero. The practical realization of this is what is known as integrator. At this stage, another problem is faced and it is the problem of overshoot. Since this system has finite thermal resistance, there is a thermal lag which causes the overshoot of temperature. This is more prominent during transient changes of the set point. It is significantly reduced by the addition of a third control, called derivative control. Normally overshoot is attributed to application of much more power than is required or to the thermal response as mentioned earlier. The second problem causes time lag between a change in output power and the control sensor sensing this change.

By means of a differentiator circuit which provides a signal proportional to the rate of temperature change, and which is subtracted from the proportional output signal, the rate action is achieved to reduce the overall effective gain driving the output power stage. This slows down the rate of temperature rise and allows more time for thermal stability to be achieved. Consequently overshoot is substantially reduced.

The temperature controller used here can be effective to control the temperature within one tenth of a degree. It has a wide range of temperature that it can control and has remote control facility so that it can be connected to the computer and can be set using IEEE 488 interface.

It has the facility to allow the user to program the temperature voltage characteristics of the sensor in order to achieve a required temperature rise/fall function of time. The set point and control temperatures can be read remotely and also all the controls (proportional, integral and derivative) can be set to desired percentages remotely. This helps in dynamic control of temperature ramping.

CHAPTER FOUR

4.1 SOFTWARE SETUP

The software for the system is divided into two parts. One part deals with data acquisition and other part is signal processing software. In general the software runs in an interactive and user friendly environment. User interaction is straight forward and easy to understand. Programs are modularized and integrated to perform the overall task. The overall operation is divided into several smaller tasks, for example, data acquisition is subdivided into various tasks like instrument set up, actual data acquisition, set up display. Similarly processing task is subdivided into data I/O, generating data arrays for conventional DLTS plots or generating data for Arrhenius plot, calculations of various DLTS parameters, plot routines, curve fitting. Each individual task is performed by a separate routine which is integrated to main program by means of a global data area. Discussion of the programs follow in next couple of paragraphs.

4.2 DATA ACQUISITION SOFTWARE

This program is menu driven for ease of user interaction. Upon start of the program, main menu is displayed after initialization. Initialization part takes care of connecting digitizer and temperature controller on IEEE bus to the system and initializing the devices to start up status. The main menu has the following broad options:

1. Status display of instruments
2. Instrument set ups
3. Data acquisition.

In status display the user can opt to see the present status of either instrument. In the case of the digitizer, it displays the present settings of mode, XYZ display, main and graticule intensities, focus, status of graticule mode, operation complete mode, remote request (for details on each of these functions, please refer to the manual). Similarly in the case of temperature controller, it displays the control temperature, display temperature, set point, gain, rate and reset values and heater power range presently enabled. From each of these display routines, the user can return to main menu by hitting return. Another point to note is that switching between the various options available through the main menu is possible only by returning to the main menu and then choosing another option.

Among the set up options, the user can decide to choose between the digitizer or the temperature controller. Some of the parameters on these instruments are independently controllable and the user is free to change their values. But certain parameters are strictly controlled by software and any request to make changes in their values is simply ignored. For the changeable parameters, user is guided to the possible values or range of values that can be assigned. Any value which is out of range is simply ignored and present value remains in effect. Certain parameters, which have only two possible values, upon selection their value is simply toggled. Everytime a parameter is changed, the display is immediately updated and user can right away see the new value in effect. The program remains in set up subroutine until user gives a specific command to return to main menu.

In the data acquisition subroutine, the user is asked to specify the data, temperature and status file names. Data and temperature file names are stored in a status file along with some run time parameters (as mentioned later) for the use of processing routines. One important point to notice is that data file has to be opened in binary mode for any I/O operation. Before the data acquisition is started, some user interaction is required in order to get some parameters like ground reference (as discussed in waveform scaling), initial and final temperatures, number of signal averages, slope of temperature ramp. Upon getting all these informations, the program starts a clock support routine, which every one second, sets a flag. This flag is used to measure time lapse (in seconds) to check the temperature ramp at the end of the period. If the temperature is not within 0.1 degree of set point, the feedback parameters on temperature controller are changed accordingly. And then digitization process is started. Digitizer is sent a command to perform the digitize and signal average the data number of times specified by user. The digitizer is externally triggered with the same frequency pulse as applied to the sample as filling pulses. After the specified number of samples have been signal averaged, the digitizer asserts the Service Request Signal on IEEE bus. Once the request is received, the digitizer is serially polled to check the exact cause of service request. If it is opeation complete request, then digitizer is sent the command to read out the signal averaged data, which is written out to the data file on the disk and corresponding temperature reading is recorded in temperature file. Then set point is changed by step size as specified by user (incremented/decremented) and process is repeated until the set point reaches the final temperature. At that point all the output files are saved and program returns to main menu. Here user can decide to exit the program by choosing appropriate option or can repeat the steps mentioned above.

4.3 DATA PROCESSING SOFTWARE

Besides signal processing, data processing program includes software for plotting the data at various stages of processing, i.e., plotting raw data, plotting conventional DLTS spectra or generating Arrhenius plots.

User needs to specify the status file name at the beginning of the data processing program. The data file and temperature file names are read from status file. The data file is opened for read operation in binary mode, whereas temperature file is opened in ASCII mode. The temperature values are read from temperature file one at a time and the corresponding digitized data are read from data file in the format as explained in waveform I/O section in Chapter 3. Then data is checked for validity (should be +ve, i.e., bit 16 should be zero). If data set is valid, it is scanned for various capacitance values at different delay times as specified by user. Before this data is normalized to undo the effect of summing for signal averaging. The difference of capacitance values for timings t1 and t2 are stored in arrays for all combinations of t1 and t2 given by no of trials and type of variation in values as specified by user in the beginning of program. Once all the data sets have been scanned and c(t1) - c(t2) values have been calculated, user is asked if he wants to perform a moving window averaging on data. Two options are available to select from. A 3-point averaging or a 5-point averaging. For 3-point averaging, the matrix used in (.25, .5, .25) and for 5-point averaging, matrix used is (.1, .2., .4, .2, .1). Data smoothing is performed as many times as user wants and every time any window size can be selected. After this, if user wants conventional DLTS plots are generated. From the data arrays of conventional DLTS spectra, the peaks are identified and their corresponding temperatures are found. Then Arrhenious plot of log $(1/\tau_{max} T^2)$ vs. 1000/T is plotted. These are nothing but individual points corresponding to each peak in conventional DLTS spectra. Then a curve fitting routine is used to fit a line through these points. The intercept on y-axis and slope of this line are used to calculate activation energy and capture cross-section of traps as discussed in the theory of DLTS.

Before plotting the data, they are sorted by x-variable and the user can choose to scale them to any range. Plot routines are divided into 3 parts to perform the tasks of initialization of plotter, plotting of actual data and labelling of the plots. At the time of labelling, user is asked to provide plot title, x and y axes units.

Chapter 5

5.1 RESULTS AND ANALYSIS

Figures (10) to (15) show the results obtained from this system for a gold doped Silicon (p^+-n type) diode. The time consumption to carry out the data acquisition was around two and half hours and signal processing required another twenty mintues. The signal processing time includes time used in plotting the data, which took most of the twenty minutes time. This compares to the time required by analog methods, using boxcar integrator (as tested in the lab here) which runs into two hours per point on an Arrhenius plot. Also all the calculations need to be done by hand and then the plotting as compared to all this being done by the computer and the plotter in the system described here.

Figure (10) to (13) show the transients as captured by the digitizer at various temperatures. Here they are represented in terms of voltage vs. time. The equivalent capacitance values can be calculated easily if the capacitance meter characteristics are known. Figure (14) is the conventional DLTS plot as generated by signal processing software. The time windows used were starting from t_1 = 0.05 msec and t_2 = 0.5 msec with the increment step of 0.05 msec keeping the ratio t_1/t_2 constant. The plot shows only four peaks although 20 peaks were generated (software allows to plot any number of peaks). Values in the plot have been offsetted by a count of 256 (511 being the highest count possible as output of digitizer and difference of two readings can not be greater than that. Offset value is user selectable). Actual values are zero and below and for plotting purposes, they are offsetted.

Fig 10. Transient at 290.5 K

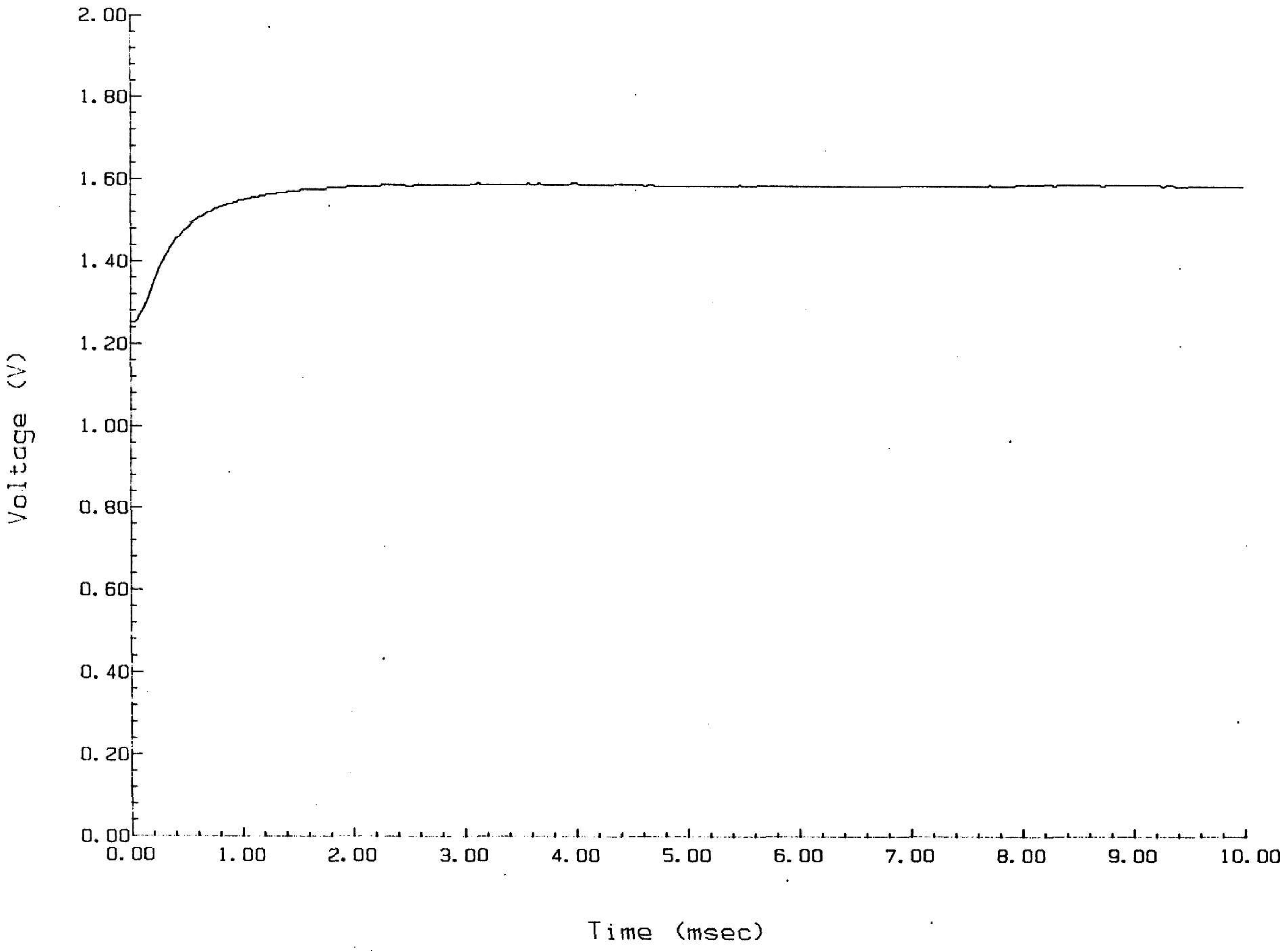


Fig 11. Transient at 295.5 K

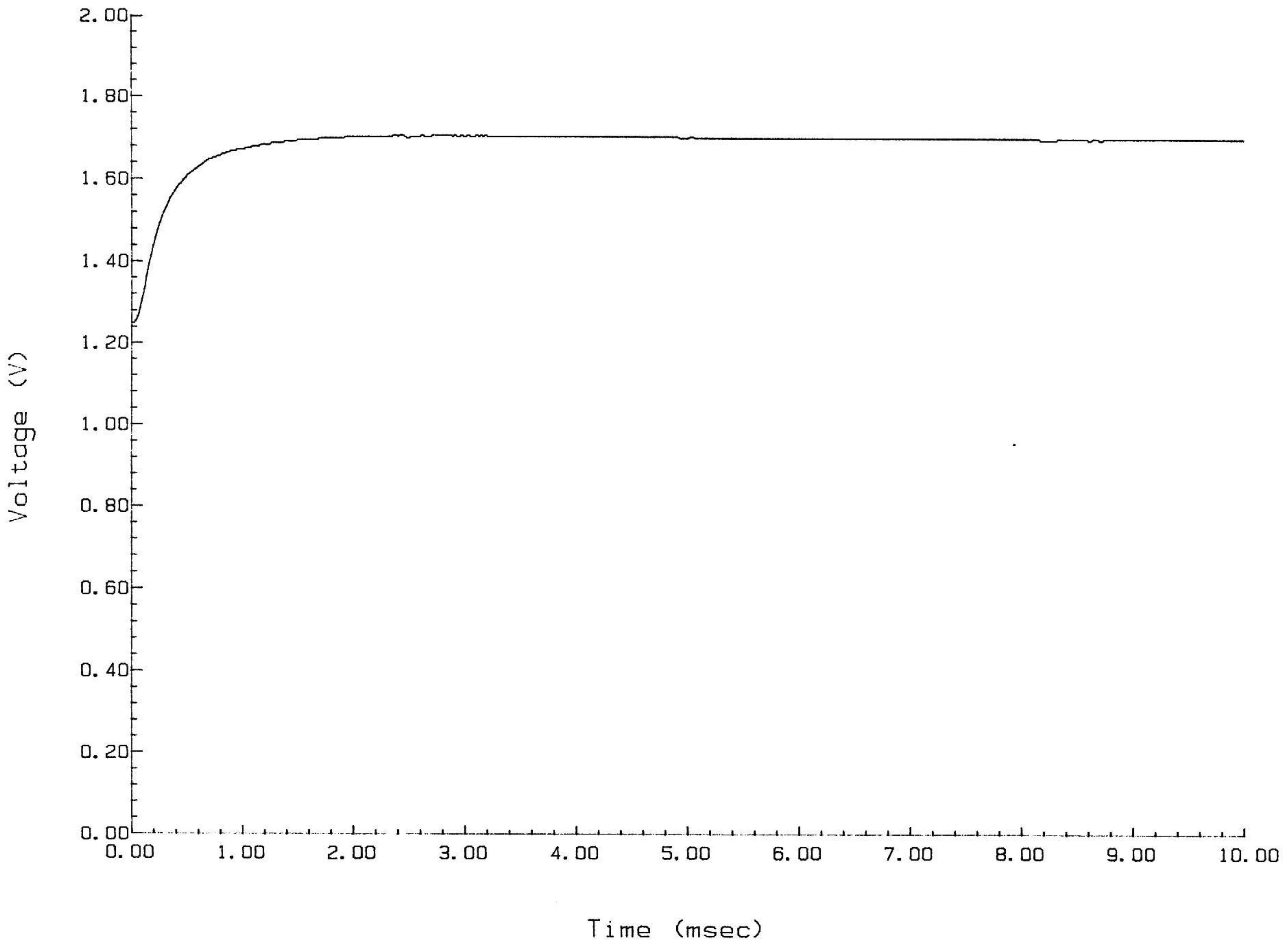


Fig 12. Transient at 300.1 K

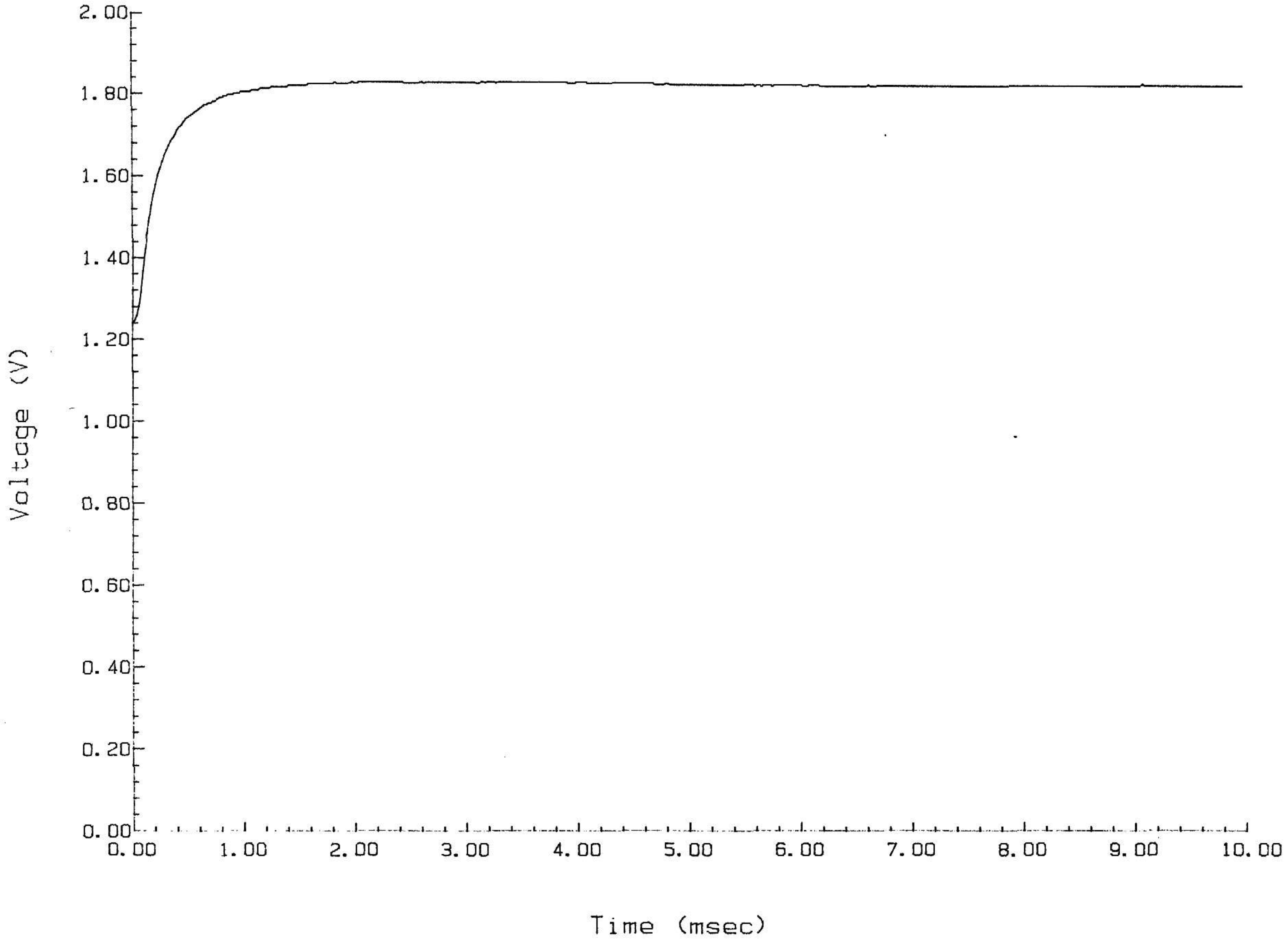


Fig 13. Transient at 305.5 K

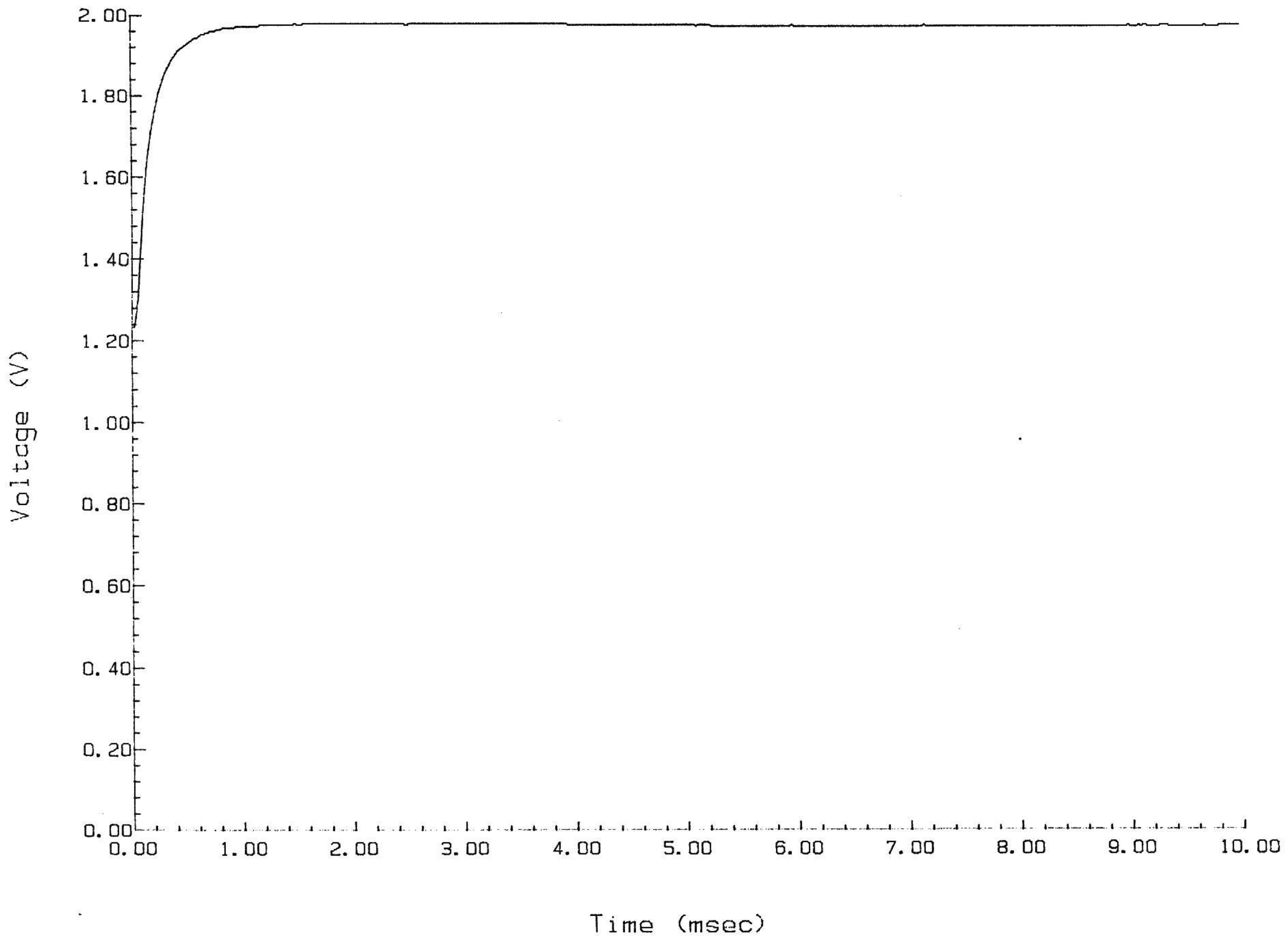


Fig 14. Conventional DLTS Spectra

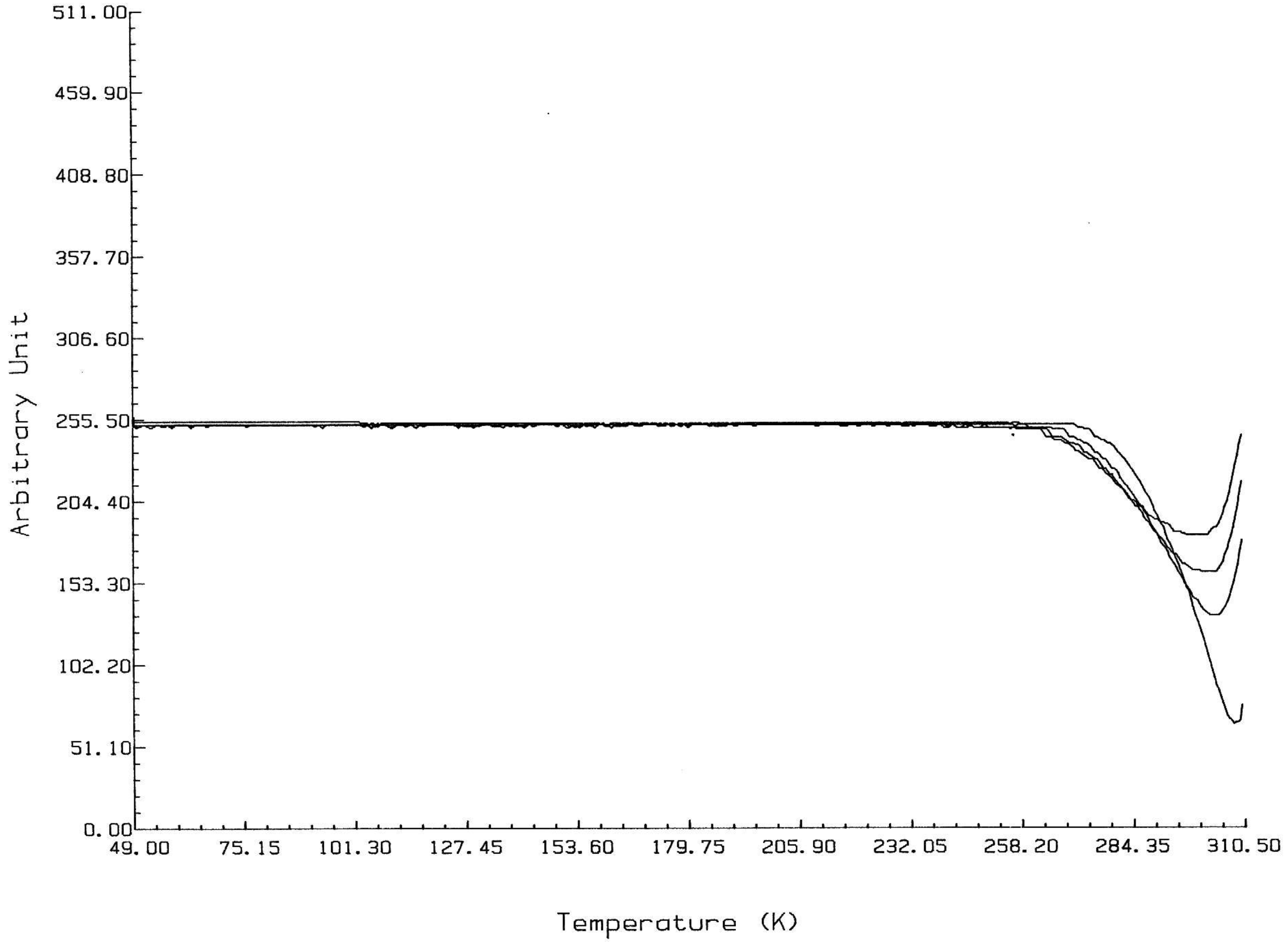


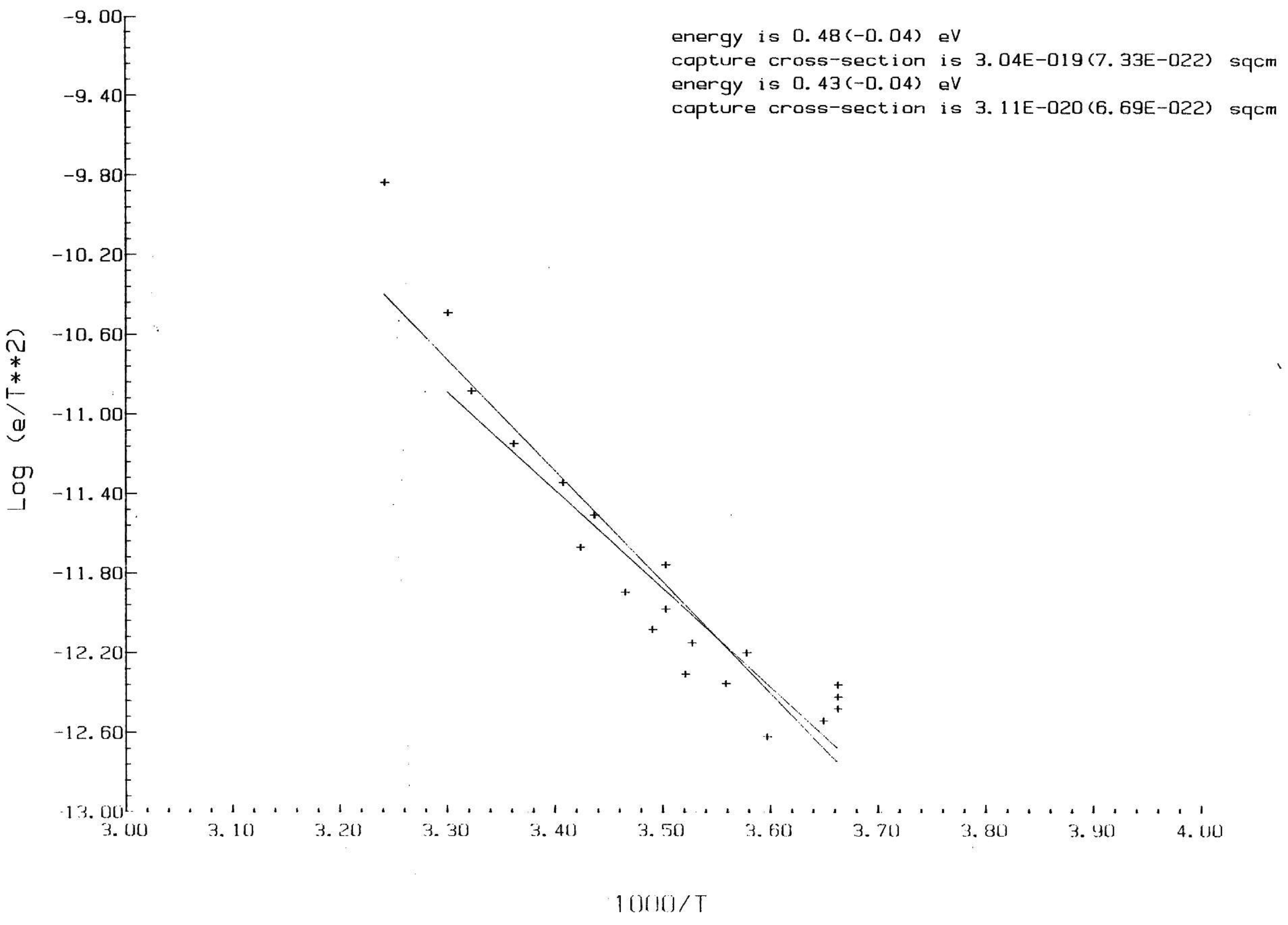
Figure (15) is the Arrhenius plot as discussed earlier. Each point on the plot represents a peak on DLTS spectra. Two attempts were made to fit a straight line through these points. The first line represents a fit through all the points whereas the second line was drawn after rejecting the first point. The software lets the user reject as many points as he wishes. This helps in removing the occasional discrepencies which may have occurred, in users judgement, by visual inspection.

From the Arrhenius plot, the slope and the y-intercept of the line are obtained. These values are used to calculate the activation energy and capture cross-section area as discussed in theory. These values are compared to the values obtained for similar traps using other techniques namely analog DLTS and the values are comparative.

Fig. (16) to (18) are the photographs of the signal captured from scope, XYZ display and TV monitor. The comparison of XYZ display picture and the raw data plot of the transient can give an idea about how good resolution, for digitization, we have.

Thus through the test runs, we could demonstrate that this digital apparatus has served its purpose, i.e., reduce the run time to perform the experiment and results are comparative and in some senses better than those obtained from conventional methods. It has better performance when comparing the presentation of the results, i.e., it generates nice plots and reduces the time it takes to do so. In no way is the accuracy of the results sacrificed. And as the raw data is available to the user at all times, he can even perform different waveform analysis than available through this software. So this system has distinct advantages over other conventional methods on time and analysis of the results.

Fig 15. Arrhenius Plot



```
energy is 0.48(-0.04) eV
capture cross-section is 3.04E-019(7.33E-022) sqcm
energy is 0.43(-0.04) eV
capture cross-section is 3.11E-020(6.69E-022) sqcm
```

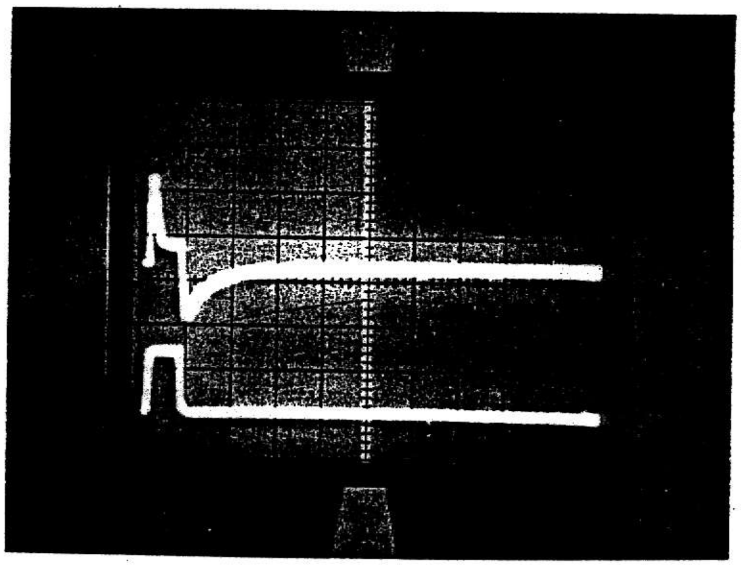
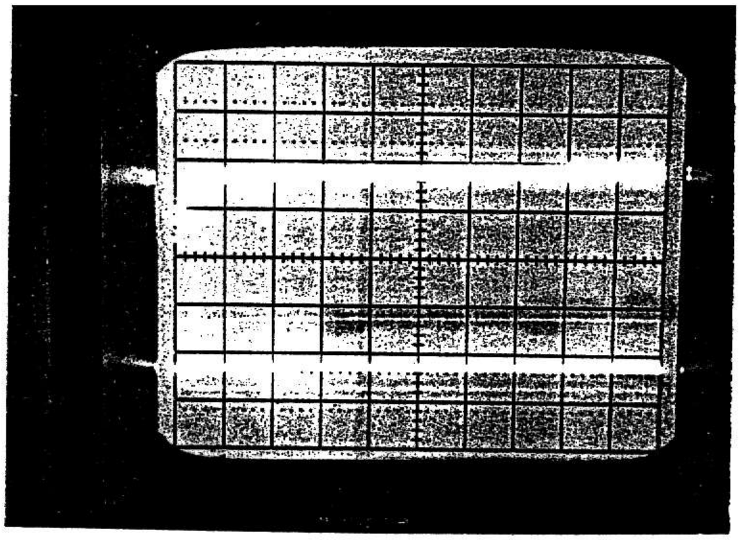


Fig 16. Signal as seen on Oscilloscope



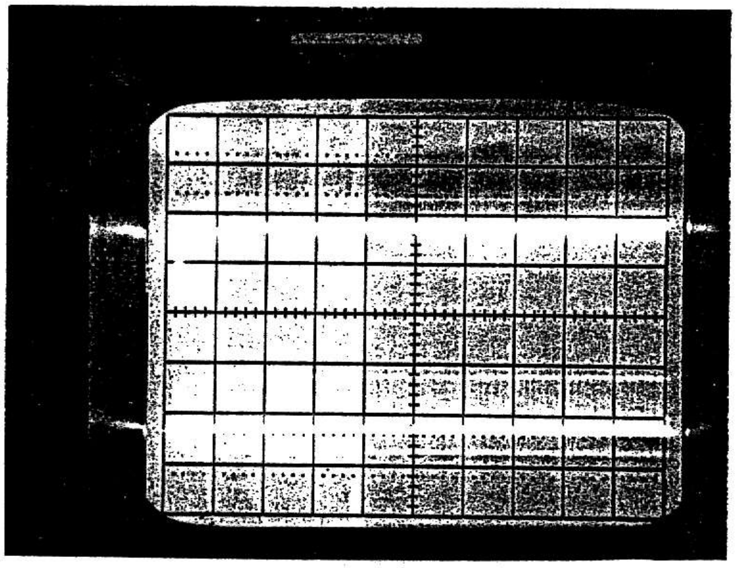


Fig 17. Signal as seen on XYZ Display

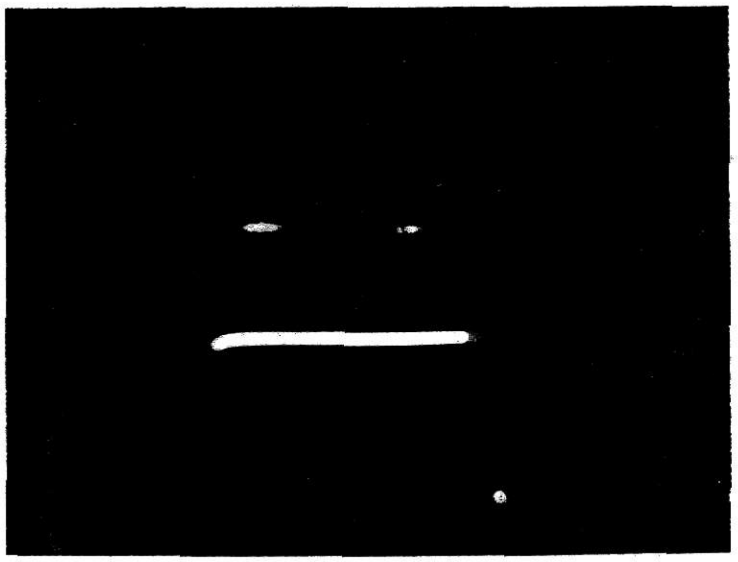


Fig 18. Signal as seen on TV Monitor

Literature Cited:

1. D. V. Lang, Deep Level Transient Spectroscopy: A new method to characterize traps in semiconductors. J. Appl. Phy. 45, 3023 (1974).

2. Wagner, Hiller, Mars, Fast Digital Apparatus for Capacitance Transient Analysis, Rev. Sci. Instrum. 51 (9), 1205 (Sept. 1980).

3. Miller, Lang, Kimerling, Capacitance Transient Spectroscopy, Ann. Rev. Master. Sci. 377 (1977).

4. Jack, Pack, Henriksen, A Computer Controlled Deep Level Transient Spectroscopy System, IEEE Trans. Electron Devices, Vol. ED 27, 2226, (1980).

5. Instruction Manual, 7912AD Programmable Digitizer, Tektronix.

6. Instruction Manual, DRC-81C, Temperature Controller, Lake Shore Cryotronics.

APPENDIX A

```
/*
*****************************************************************
*                                                               *
* Programmer: HEMANT MEHTA    Date: 6/25/86                     *
*                                                               *
* Course: EECE898 Masters Report                                *
*                                                               *
*****************************************************************/


#include <stdio.h>
#include <decl.h>
#include "var.c"

main()
{
char c;


/* call init to attach and initialize the instruments on IEEE bus */

     init();

     do {

/* display the main menu */

     dis_men();
     c = bdos(1) & 0xff;
     switch(c) {

/* switch to subsection of the display as desired by user */

     case '1' : dig_stat();  /* digitizer status disp */
                puts("Hit return to Continue");
                bdos(1);
                break;
     case '2' : drc_stat(); /* temp controller status disp */
                puts("Hit return to Continue");
                bdos(1);
                break;
     case '3' : dig_set(); /* digitizer set up routine */
                break;
     case '4' : drc_set(); /* temp controller set up routine */
                break;
     case '5' : run();    /* data acquisition routine */
                break;
     case '6' : exit(0);  /* quit */
                break;
     }
     } while(c != 'z');
}
```

```
init()
{

extern int dgtzr,drc;

     puts("Turn on the power of all the instruments and ");
     puts("hit return when ready.");
     bdos(1);

/* connect the digitizer on IEEE bus */

     if((dgtzr = ibfind("DEV1"))<0) finderr();

/* put digitizer online */

     ibonl(dgtzr,1);

/* send a dummy digitize command to digitizer */

     ibwrt(dgtzr,"DIG DAT;",8);
     if( ibsta & ERR ) error();

/* connect temp controller on IEEE bus */

     if((drc = ibfind("DEV12"))<0) finderr();

/* put it inline */

     ibonl(drc,1);

/* select display B, heater on full power range and set to
   communicate with gpib controller with EOI signal and no
   extra terminator for read or write.
*/

     ibwrt(drc,"Z0M2T3R4BS\0xA",11);
     if( ibsta & ERR ) error();

/* disable terminators set by default on controller for
temp controller */

     ibeos(drc,0);
}
```

```
dis_men()
{

     puts("   **** MAIN MENU ****");
     puts("        ---------\n\n");
     puts("     1) Digitizer Status Display");
     puts("     2) Temperature Controller Status Display");
     puts("     3) Digitizer Set-up");
     puts("     4) Temperature Controller Set-up");
     puts("     5) Run Data Aquisition Routine");
     puts("     6) Exit the Program");
     puts("\n\n     Enter your selection");

}
```

```
dig_stat()
{

extern int dgtzr,ibsta;
extern char set[];
int i;

/* read the settings from digitizer */

     ibwrt(dgtzr,"SET?",4);
     if( ibsta & ERR ) error();

     if( ibrd (dgtzr,set,90) & ERR ) error();

/* print routine prints the values of various settings as read into
   the string from digitizer.
*/


     puts("   **** DIGITIZER STATUS DISPLAY ****");
     puts("        -----------------------\n\n");

     printf("  a) Mode                        ");
     i=4;
     print(i);
     i += 5;
     printf("  b) Graticule                   ");
     print(i);
     i += 3;
     printf("  c) TV Scale Factor Display     ");
     print(i);
     i += 4;
     printf("  d) XYZ Display                 ");
     print(i);
     i += 3;
     printf(" *e) Device Trigger Function     ");
     print(i);
     i += 4;
     printf("  f) SRQ on REM                  ");
     print(i);
     i += 4;
     printf(" *g) SRQ on Operation Complete");
     print(i);
     i += 4;
     printf("  h) Main Intensity              ");
     print(i);
     i += 4;
     printf("  i) Graticule Intensity         ");
     print(i);
     i += 4;
     printf("  j) Focus                       ");
     print(i);
```

```
        i += 3;
        printf(" *k) Trace Width                      ");
        print(i);
        i += 3;
        printf(" *l) Ratio                            ");
        print(i);
}
```

```
drc_stat()
{

extern int drc,ibsta;
extern char setl[100];


     puts("   **** TEMPERATURE CONTROLLER STATUS ****");
     puts("        ----------------------------\n\n");

/* read temperature values (display, sensor and setpoint) */

     ibwrt(drc,"WO",2);
     if( ibsta & ERR ) error();

     if( ibrd (drc,setl,25) & ERR ) error();

     printf(" *1) Display  Temperature  = %.7s\n",setl);
     printf(" *2) Control  Temperature  = %.7s\n",&setl[8]);
     printf("  3) SetPoint Temperature  = %.7s\n\n",&setl[16]);

/* read the gain,rate reset and heater range settings */

     ibwrt(drc,"Wl",2);
     if( ibsta & ERR ) error();

     if( ibrd (drc,setl,25) & ERR ) error();

     printf("  4) Gain   Setting         = %.2s%%\n",&setl[10]);
     printf("  5) Rate   Setting         = %.2s%%\n",&setl[13]);
     printf("  6) Reset  Setting         = %.2s%%\n",&setl[16]);
     printf("  7) Heater Power           = %.1s \n",&setl[19]);
}
```

```
drc_set()
{
extern int gain,rate,reset,drc,range,ibsta;
extern char a[];
extern float set_point;
char x;

     do {

/* display the present status of controller */

     drc_stat();

/* now let user select what he wants to change some of the
   settings are program controlled and user can not change
   them if he selects to change one of these, it will be
   simply ignored
*/

     puts("\n\n  Enter your selection");
     puts(" Type 0 to return to menu");
     puts(" (you can't select * options)");

     x = bdos(1) & 0xff;
     puts("\n");

     switch(x)  {

     case '3' : puts("Enter new value");

                scanf("%f",&set_point); /* set point value */

                set_temp();

                break;

     case '4' : puts("Enter new value (0-99 %)");

                scanf("%d",&gain);      /* gain setting */

                sprintf(a,"P%d",gain);

                ibwrt(drc,a,3);
                if( ibsta & ERR ) error();

                break;

     case '5' : puts("Enter new value (0-99 %)");

                scanf("%d",&rate);      /* rate setting */

                sprintf(a,"D%d",rate);
                ibwrt(drc,a,3);
                if( ibsta & ERR ) error();

                break;

     case '6' : puts("Enter new value (0-99 %)");

                scanf("%d",&reset);   /* reset setting */

                sprintf(a,"I%d",reset);

                ibwrt(drc,a,3);
                if( ibsta & ERR ) error();

                break;

     case '7' : puts("Enter new value (0-4)");

                scanf("%d",&range);  /* heater range setting */

                sprintf(a,"R%d",range);

                ibwrt(drc,a,2);
                if( ibsta & ERR ) error();

                break;

     }

     } while(x != '0');

}
```

```
dig_set()
{

extern int dgtzr,mode,grat,tv,mai,gri,foc,rem,ibsta;
char s,x;


     do {

/* display present status of digitizer */

     dig_stat();

     puts("\n\n Enter your selection or x  to skip");
     puts(" (you can not change * options)");

     s = bdos(1) & 0xff;


     switch(s)  {

/* toggle the mode tv/digital */

     case 'a' : if((mode ^= 1) == 0)

                ibwrt(dgtzr,"MOD TV;",7);

                else

                ibwrt(dgtzr,"MOD DIG;",8);

                break;

/* toggle graticule on/off */

     case 'b' : if((grat ^= 1) == 0)

                ibwrt(dgtzr,"GRAT OFF;",9);

                else

                ibwrt(dgtzr,"GRAT ON;",8);

                break;

/* toggle tv scale factor display on/off */

     case 'c' : if((tv ^= 1) == 0)

                ibwrt(dgtzr,"TV OFF;",7);
                else

                ibwrt(dgtzr,"TV ON;",6);

                break;

/* select the display data type for XYZ display */

     case 'd' : puts("  XYZ Display\n");
                puts("1) On\n2) Off\n3) Raw");
                puts("4) ATC\n5) SA\n");

                puts(" Enter your selection");

                x = bdos(1) & 0xff;

                switch(x) {

                case '1' : ibwrt(dgtzr,"XYZ ON;",7);
                           break;

                case '2' : ibwrt(dgtzr,"XYZ OFF;",8);
                           break;

                case '3' : ibwrt(dgtzr,"XYZ RAW;",8);
                           break;

                case '4' : ibwrt(dgtzr,"XYZ ATC;",8);
                           break;

                case '5' : ibwrt(dgtzr,"XYZ SA;",7);
                           break;
                }

                break;

/* set srq on/off from remote mode through remote button */

     case 'f' : if((rem ^= 1) == 0)

                ibwrt(dgtzr,"REM OFF;",8);

                else

                ibwrt(dgtzr,"REM ON;",7);

                break;

/* set main intensity level */

     case 'h' : puts("Enter new value (0-1023)");
```

```
                   scanf("%d",&mai);

                   sprintf(a,"MAI %d;",mai);

                   ibwrt(dgtzr,a,9);
                   if( ibsta & ERR ) error();

                   break;

/* set graticule intensity level */

        case 'i' : puts("Enter new value (0-255)");

                   scanf("%d",&gri);

                   sprintf(a,"GRI %d;",gri);

                   ibwrt(dgtzr,a,8);
                   if( ibsta & ERR ) error();

                   break;

/* set focus level */

        case 'j' : puts("Enter new value (0-63)");

                   scanf("%d",&foc);

                   sprintf(a,"FOC %d;",foc);

                   ibwrt(dgtzr,a,7);
                   if( ibsta & ERR ) error();

                   break;
        }

        } while(s != 'x');

}
```

```
/* this routine acts as user timer interrup service in DOS
   and sets a flag every one second
*/

int_serv()
{

extern int flag,time_count;

     time_count++;
     if(time_count < 18) return;
     time_count = 0;
     flag = 1;

}



/* this routine sets the address of the above interrupt
   service routine in DOS interrupt vector table
*/

int_set()
{
     time_count = 0;
     intrinit(int_serv,256,0x1c);

}
```

```
/* This routine waits for no of seconds as determined in run
   routine and after the delay checks the temperature against
   set point and changes the gain setting if required.
*/


wait()
{

extern int drc,dgtzr,flag,count,gain;
extern float set_point;
extern char *fp2;
float k_new;
int i;
char a[10];

/* interrupt service routine sets the flag every one sec.
   This routine continuously checks the flag and decrements
   the count every time flag is set.
*/

     i = 0;
     flag = 0;
     do {
       if(flag != 0) {
         flag = 0;
         i++;
       }
     } while(i < count);


/* now read the temp from controller */

     ibwrt(drc,"WC",2);
     ibrd(drc,a,10);
     sscanf(a,"%f",&k_new);

/* if the temp is wwithin 0.1 degree of the set point return
   else change the gain to reduce/increase heating rate.
*/

     if((k_new <= set_point+0.1) && (k_new >= set_point-0.1)) return;

     if(k_new > set_point)
       gain--;
     else
       gain++;

/* write out the new value of gain */

     sprintf(a,"P%d",gain);
     ibwrt(drc,a,3);
```

```
    printf("%d%%  %.2f\n",gain,k_new);

}
```

```
/* This is the main data acquisition routine */


run()
{

extern int dgtzr,drc,count,ibsta,ibcnt;
extern float set_point,k;
float k_init,k_final,k_inc;
char name[25],name1[25],name2[25],spr,str[10],str2[50];
int i,direc,strglen,num_write,gndref,safe[2],k_slope,samples;
extern char *fp,*fp1,*fp2,*fopen();
unsigned char *buffer;


/* Get the data,temp and status file names. Open the files.
   Make sure data file is opened in binary mode,whereas the
   other two are in ascii mode same should be true while
   reading the files.
*/

try:

     puts("Enter the Data File Name");

     scanf("%s",name);

     if((fp = fopen(name,"wb")) == NULL) {

       printf("ERROR! can not open %s\n",name);
       goto try;
     }

retry:

     puts("Enter Temp File Name");

     scanf("%s",name1);

     if((fp1 = fopen(name1,"w")) == NULL) {

       printf("ERROR! can not open %s\n",name);
          goto retry;
     }


     puts("Enter Status File Name");

     scanf("%s",name2);

     if((fp2 = fopen(name2,"w")) == NULL)
       printf("Error in opening File %s\n",name);
```

```
/* write the data and temp file names in status file for
   later checking during processing.
*/

     fprintf(fp2,"%s\n%s\n",name,name1);

/* get the initial and final temps */

     puts("Enter Initial Temperature");

     scanf("%f",&k_init);

     puts("Enter Final Temperature");

     scanf("%f",&k_final);

/* read the step to change the temp */

     puts("Enter Set-Point increment/decrement");

     scanf("%f",&k_inc);

/* set the direction flag for temp 1, if increasing 0, if
   decreasing and in that case make step -ve.
*/

     direc = 1;

     if((k_final - k_init) < 0) {
           k_inc *= (-1);
           direc = 0;
     }


/* get the rate of rise of temp */

     puts("Enter slope(degree per min)");

     scanf("%d",&k_slope);

/* calculate the time for rise of temp by step size in secs */

     count = (int)(60/k_slope*k_inc);

/* get the no of signals to be averaged for digitizer */

     puts("Enter no of samples per set(max 64)");

     scanf("%d",&samples);

/* save it in status file */
```

```
     fprintf(fp2,"%d\n",samples);

/* allocate the buffer in memory to read data from digitizer */

     buffer = calloc(1029,1);

/* set graticule and main intensities and enable digitizer to
   assert srq when digitize operation is completed
*/

     ibwrt(dgtzr,"OPC ON;GRI 0;MAI 540;",21);

/* put digitizer in local mode */

     ibloc(dgtzr);

     puts("Do any setting,if needed, of V/D & T/D and hit return");
     puts("Do not change intensity levels");
     bdos(1);

/* this part digitizes the gnd level setting on digitizer */

     puts("put digitizer input to GND and position GND level");
     puts("hit return when ready");
     bdos(1);

/* give the digitize command and average to center */

     ibwrt(dgtzr,"DIG DAT;ATC;",12);

/* wait for digitizer to assert srq */

     ibwait(dgtzr,RQS);

/* read the status byte */

     ibrsp(dgtzr,&spr);

/* read the averaged signal */

     ibwrt(dgtzr,"REA ATC;",8);
     ibrd(dgtzr,buffer,1029);

     gndref = 0;

/* take the average of a portion of digitized signal */

     for(i=415; i<615; i+=2)
           gndref += (buffer[i]*256+buffer[i+1])>>1;

     gndref = gndref/100;
```

```
/* save it in status file as gnd ref level */

     fprintf(fp2,"%d\n",gndref);

/* put the digitizer in local mode again */

     ibloc(dgtzr);
     puts("put digitizer input to DC and hit return when ready");
     bdos(1);

/* set the delay time count for wait routine. count off the data
   acquisition time for digitizer (calculated by product of no of
   samples and pulse rate, which is assumed to be 30 msec here)
   and take off another sec for all other read write operations
*/

     count = count - 1 - (int)(samples*30.0/1000.0);

/* set the first set point a step lower(or higher) than initial
   temp
*/

     set_point = k_init - k_inc;

     puts("Turn on cooling system and hit return when ready.");

     bdos(1);

/* this routine writes out the setpoint value to temp controller */

     set_temp();

/* set the temp controller to send control temp whenever read
   command is issued
*/

     ibwrt(drc,"WC",2);
     if( ibsta & ERR ) error();

/* read the scale setting V/D and T/D from digitizer */

     ibwrt(dgtzr,"REA SCl;",8);
     ibrd(dgtzr,str2,30);

/* save them in status file */

     fprintf(fp2,"%s\n",str2);

/* wait till temp is close enough to set point */

      do {
            read_temp();
      } while(k < (set_point-0.2));
```

```
/* Generate the command string for digitizer to do digitize
   operation
*/

     sprintf(str,"DIG SA,%d;",samples);
     strglen = strlen(str);

/* save the original timer interrupt address */

     safe[0] = peek(0x70,0);
     safe[1] = peek(0x72,0);

/* set the new timer interrupt address */

     int_set();

     do {

/* change the set point */

        set_point += k_inc;

        set_temp();

/* wait for the temperature to start to change */

        wait();

/* trigger the digitizer */

        ibwrt(dgtzr,str,strglen);
        if(ibsta & ERR) error();

/* wait for the digitizer to set srq line */

        ibwait(dgtzr,RQS);

        ibrsp(dgtzr,&spr);

/* if digitizer request is valid */

        if((spr & 0xAF) == 0x02) {

/* read the temperature */

           read_temp();

/* send the command to digitizer to load the data */

           ibwrt(dgtzr,"REA SA;",7);
           if(ibsta & ERR) error();
/* read the data */

           if( ibrd (dgtzr,buffer,1029) & ERR ) error();

/* write out the data to disk */

           i = 0;
           if(ibcnt == 1029) {

              do {
                 num_write = fwrite(&buffer[i],1,ibcnt,fp);
                 if(num_write != ibcnt)
                       ibcnt -= num_write;
                       i += num_write;
              } while(ibcnt != num_write);

/* write out the temperature to file */

              fprintf(fp1,"%.2f\n",k);

           }

/* if there is a error in reading from the digitizer */

           else {

              ibrsp(dgtzr,&spr);
              printf("Read error status for dgtzr %02x",spr);
           }

        }

/* if the digitizer request is invalid */

        else
           printf("dgtzr error status %02x",spr);

/* loop till the final temperature is reached */

   } while(((set_point<k_final+k_inc)&&direc) ||
                      ((set_point>k_final+k_inc)&&!direc));

/* free the buffer */

     free(buffer);

/* close the files */

     fclose(fp);
     fclose(fp1);
     fclose(fp2);
```

```
/* restore the timer address */

     pokew(0x70,0,safe[0]);
     pokew(0x72,0,safe[1]);

}
```

```
#define ERR (1<<15)

print(i)
int i;
{

extern char set[],c;

     do {

     sscanf(c,"%c",&set[i]);
     putchar(c);

     } while(c != ';');

}




set_temp()
{

extern int drc,ibsta;
extern float set_point;
extern char a[];


     sprintf(a,"S%06.2f",set_point);

     ibwrt(drc,a,7);
     if( ibsta & ERR ) error();

}


read_temp()
{

extern int drc;
extern char a[];
extern float k;

     ibwrt(drc,"WC",2);
     if( ibrd (drc,a,7) & ERR ) error();
     sscanf(a,"%f",&k);

}
```

```
error()
{

extern int ibsta,iberr,ibcnt;

     puts("GPIB Function Call Error!");

     printf("ibsta = %x  iberr = %x  ibcnt = %x\n",ibsta,iberr,ibcnt);

     exit(0);

}



finderr()
{

     puts("Device open Error! Please check Power Condition");
     exit(0);
}
```

```
/*
 *************************************************
 *                                               *
 *  Programmer: HEMANT MEHTA    Date: 06/25/86   *
 *                                               *
 *  Course: EECE898 Masters Report               *
 *                                               *
 *************************************************
*/


/* This routine is the data processing routine.
   This routine calls some supporting routines
   for the purposes of reading data, ploting sorting
   etc.

*/


#include <stdio.h>

/* variable.c contains definitions of global variables */

#include "variable.c"


main()
{
extern int trial,count,reject,c12[750][25],buff[],*plotter;
extern float temp[750],peak_t[25],p_s_t[25];
extern double expn[25],e_s[25],log(),fit(),siga,sigb,a,b,exp();
extern FILE *fp,*fp1,*fp2,*fopen();
int down,read_f,sign,n_plot,l,k,i,c1[25],c2[25],flag,j,itemp;
int sample,gndref,amplf,offset;
char name[80],name1[25],response,select;
float t_unit,v_unit,b_temp,s_temp,ftemp,t1[25],t2[25],min_val;
float max_val,ratio,t_inc;
double c_cross,sigma,gamma,deltae,ftemp1,ftemp2,point1,point2;

/*
   Read the status file name. Status file contains names of
   of data and temp files and also the no of averages, gnd
   referrence value and scale factors on digitizer.
*/

      puts("enter stats file name");
      scanf("%s",name);
      if((fp2 = fopen(name,"r")) == NULL) {
            puts("Error in opening the file");
            exit(0);
      }
/* Read the data and temperature file names */

      fscanf(fp2,"%s%s",name,name1);

/* Read the no of samples for averaging and gnd ref */

      fscanf(fp2,"%d%d",&sample,&gndref);

      n_plot = 0;
      puts("do you want raw data plots");
      response = bdos(1) & 0xFF;
      if (response == 'y') {
         puts("\nHow many?");
         scanf("%d",&n_plot);
         puts("enter begin temperaure");
         scanf("%f",&b_temp);
         puts("enter temp step");
         scanf("%f",&s_temp);
         puts("enter voltage and time setting");
         scanf("%f%f",&v_unit,&t_unit);
      }

      puts("\ndo you want any amplification of data? enter
                                  a no or 1 if not");
      scanf("%d",&amplf); gets();

      puts("do you want any offset in the plot? enter a
                                               no (0-256)");
      scanf("%d",&offset); gets();

      puts("The following options are available for time
                                              settings:\n");
      puts("  1) Keep T1 fixed, vary T2");
      puts("  2) Keep T2 fixed, vary T1");
      puts("  3) Keep T1/T2 fixed, vary both\n\n");
      puts("     Enter your option");
      response = bdos(1)&0xFF;

      puts("\nenter the starting values of T1 and T2 (T2>T1)
                                             (0-10 msec)");
      scanf("%f%f",&t1[0],&t2[0]); gets();
      ratio = t1[0]/t2[0];

      puts("enter the time increment step");
      scanf("%f",&t_inc); gets();

/* no of trials decides how many points will be there on
   Arrhenius plot
*/

      puts("enter no of trials (1-25)");
      scanf("%d",&trial); gets();
      puts("Please wait...");

      for(i=1; i<trial; i++) {
            switch(response) {

/*
   calculate all the time settings for which data points from
   transient signal will be taken. (no of trials times)
*/

               case '1' : t2[i] = t2[i-1]+t_inc;
                          break;
               case '2' : t1[i] = t1[i-1]+t_inc;
                          break;
               case '3' : t1[i] = t1[i-1]+t_inc;
                          t2[i] = t1[i]/ratio;
                          break;
            }
      }

      i = -1;
      response = 'y';

   do {

      if((fp = fopen(name,"rb")) == NULL) {
               puts("error opening the file");
               exit(0);
      }
      if((fp1 = fopen(name1,"r")) == NULL) {
            puts("error opening the file");
            exit(0);
      }

         sign = 0;

         do {

/* read the temp value */

            read_f = fscanf(fp1,"%f",&ftemp);

/* if not end of file */

            if(read_f != 0) {

/* call routine to read the set of data points for a transient */

               readdata();

/* if data is ok go ahead and process */
```

```
                if(reject == 0) {

                   i++;

/* store the temp in temp_array */

                   temp[i] = ftemp;

/* check if raw data is to be plotted. if yes go ahead
   and plot
*/

                   if(n_plot != 0) {
                         if(ftemp > b_temp) {
                         puts("ready plotter and hit return");
                         bdos(1);
                         plot_init();

/* signal average the data and subtract the ground ref level */

                         for(j=0; j<512; j++)
                         buff[j] = buff[j]/sample-gndref;
                         plot_raw();
                         printf("temperature is %.2f\n",ftemp);
                         plot_write(0.0,t_unit*10,0.0,v_unit*10);
                         n_plot--;
                         b_temp += s_temp;
                         }
                   }

/* calculate the value of C[t1]-C[t2] for all t1 and t2 */

                   for(j=0; j<trial; j++) {

/*
   there are 51.2 datapoints per division in digitizer output
   assuming the setting for 1msec/div calculate the index of
   datapoint to be extracted
*/

                      c1[j] = buff[(int)(t1[j]*51.1)];
                      c2[j] = buff[(int)(t2[j]*51.1)];

/*
   devide by any averaging factor as read from status file,
   amplify if user wishes
*/

                      c12[i][j] = (c1[j]-c2[j])/sample*amplf;

                      if(c12[i][j]<0)
                         sign -= 1;
```

```
                    if(cl2[i][j]>0)
                       sign += 1;

/* and add offset if desired by user */

                    cl2[i][j] += offset;
                  }

                }
             }

          } while(read_f != 0);

          fclose(fp);
          fclose(fpl);


/* check if user wants to include any other set of files */

          puts("do you want to read any other file");
          response = bdos(1) & 0xFF;
          if(response == 'y') {
                puts("\nenter data file name");
                scanf("%s",name);
                puts("enter temp file name");
                scanf("%s",namel);

          }

    } while(response == 'y');


          count = i+1;
          printf("\n%d files read\n",count);

/*
   smoothing routine for 3 or 5 point smoothing. matrix used for
   3 point smoothing is (1/4,1/2,1/4) and for 5 point (1/10,1/5,
   1/2.5,1/5,1/10)
*/

       puts("do you want smoothening of the data");
       response = bdos(1) & 0xFF;
       if(response == 'y') {

        do {
          puts("\nhow many point smoothening (3 or 5)");
          scanf("%d",&l);

          for(j=0; j<trial; j++) {
             for(k=(int)(l/2); k<(i+1-(int)(l/2)); k++) {
                if(l == 3)
        cl2[k][j] = cl2[k-1][j]/4+cl2[k][j]/2+cl2[k+1][j]/4;

        else {

        cl2[k][j] = cl2[k-2][j]/10+cl2[k-1][j]/5+cl2[k][j]/2.5;
        cl2[k][j] += cl2[k+1][j]/5+cl2[k+2][j]/10;
        }
       }
     }
     puts("do you want smoothening again");
     response = bdos(1) & 0xFF;

     } while(response == 'y');

     }


/*
   here if user wishes, conventional DLTS spectrums are plotted
   for various time settings as specified by user
*/

        puts("\ndo you want the DLTS plots");
        response = bdos(1) & 0xFF;

        if(response == 'y') {
              puts("\nhow many");
              j = trial;
              scanf("%d",&trial);

/*
   call sort_f to sort the data by temperature values as they
   will be plotted against temperature.
*/

              puts("\nsorting the data");
              sort_f();
              puts("ready plotter and hit return");
              bdos(1);

/* initialize the plotter (draw the axes and put tick marks) */

              plot_init();

/* plot the actual data */

              plot();
              getchar();

/* write division markings on axes and titles */
```

```
                plot_write(temp[0],temp[count-1],0.0,511.0);

                trial = j;
        }

/* check if user wants arrhenius plots */

        puts("\ndo you want arrhenius plot");
        response = bdos(1) & 0xFF;

        if(response == 'y') {

/*
   find the peaks for various time settings from conventional
   DLTS data if majority of c[t1]-c[t2] values were -ve then we
   will have a -ve peak otherwise a +ve peak.
*/

                for(j=0; j<trial; j++) {
                   itemp = c12[0][j];
                   for(i=1; i<count; i++) {
                      if(sign<0) {

/* look for a -ve peak */

                           if(itemp > c12[i][j]) {
                               itemp = c12[i][j];
                               peak_t[j] = temp[i];
                           }
                      }

/* if not look for a +ve peak */

                      else {
                           if(itemp < c12[i][j]) {
                                 itemp = c12[i][j];
                                 peak_t[j] = temp[i];
                           }
                      }
                   }




/*
   calculate e = 1/tmax = log(t1/t2)/(t1-t2)  and calculate
   exp = e/T**2 for every t1 and t2
*/

                   expn[j] = log(t1[j]/t2[j])/(t1[j]-t2[j]);
                   expn[j] /= (peak_t[j]*peak_t[j]);
/* now take log of exp */

                        expn[j] = log(expn[j]);

/* also calculate 1000/T */

                        peak_t[j] = 1000.0/peak_t[j];

                     }

/* sort them by 1000/t values for plotting . use bubble sort */

                     do {
                        flag = 0;
                        for(i = 0; i < trial-1; i++) {
                              if(peak_t[i] > peak_t[i+1]) {
                                    ftemp = peak_t[i];
                                    peak_t[i] = peak_t[i+1];
                                    peak_t[i+1] = ftemp;
                                    ftempl = expn[i];
                                    expn[i] = expn[i+1];
                                    expn[i+1] = ftempl;
                                    flag = 1;
                              }
                        }
                     } while(flag == 1);

                     min_val = peak_t[0];
                     max_val = peak_t[trial-1];

/* look for min and max of exp values */

                     ftempl = expn[0];
                     ftemp2 = expn[0];
                     for(j=1; j<trial; j++) {
                           if(ftempl > expn[j]) ftempl = expn[j];
                           if(ftemp2 < expn[j]) ftemp2 = expn[j];
                     }

/*
   print min and max of 1000/T. They are already sorted so top
   is min and bottom is max
*/

    printf("\nminimum value of 1000/T is %f\n",peak_t[0]);
    printf("maximum value of 1000/T is %f\n",peak_t[trial-1]);

/* check if user wants to change the range for plotting the
   data
*/

     puts("do you want to enter the new values of min and max");
     response = bdos(1) & 0xFF;
              if(response == 'y') {
                 puts("\nenter new min and max values");
                 scanf("%f%f",&min_val,&max_val);
              }

/* print min and max of exp values */

        printf("\nminimum value of exp is %g\n",ftemp1);
        printf("maximum value of exp is %g\n",ftemp2);

/* check if user wants to change the range for plotting */

     puts("do you want to enter the new values of min and max");
        response = bdos(1) & 0xFF;
              if(response == 'y') {
                 puts("\nenter new min and max values");
                 scanf("%lf%lf",&ftemp1,&ftemp2);
              }

/* save the original values */

              for(j=0; j<trial; j++) {
                    p_s_t[j] = peak_t[j];
                    e_s[j] = expn[j];
              }

/*
   scale the values as specified by user or else scale the
   values to map from 0 to full scale on plotter
*/

      for(j=0; j<trial; j++) {
      peak_t[j] = (peak_t[j]-min_val)*511.0/(max_val-min_val);
      expn[j] = (expn[j]-ftemp1)*511.0/(ftemp2-ftemp1);
      }

              l = trial;

              puts("\nready the plotter and hit return");
              bdos(1);

/* initialize the plotter if not already done so */

              plot_init();

/* plot the points */

              plot_c();

/* let user choose the value of gamma */
```

```
                puts("The following gamma values are available");
                puts("\n 1) Si-n-type    7.20E21");
                puts(" 2) Si-p-type    2.94E21");
                puts(" 3) GaAs-n-type  2.28E20");
                puts(" 4) GaAs-p-type  1.70E21");
                puts("\n\nEnter your selection");
                select = bdos(1) & 0xFF;
                switch(select) {
                   case '1' : gamma = 7.20E+21;
                              break;
                   case '2' : gamma = 2.94E+21;
                              break;
                   case '3' : gamma = 2.28E+20;
                              break;
                   case '4' : gamma = 1.70E+21;
                              break;
                }
                printf("\nGamma is %g\n",gamma);
                puts("do you want to enter new value");
                select = bdos(1) & 0xFF;
                if(select == 'y') {
                   puts("\nEnter the new value");
                   scanf("%lf",&gamma);
                }


             down = (-1);
             do {

/* call line fitting routine */

                fit();

/*
   line fit returns values for a and b in a+b*x
   calculate two points from equn a+b*x and draw the line
*/
                point1 = b*p_s_t[0]+a;
                point2 = b*p_s_t[trial-1]+a;
                point1 = (point1-ftemp1)*511.0/(ftemp2-ftemp1);
                point2 = (point2-ftemp1)*511.0/(ftemp2-ftemp1);

  sprintf(name,"SP1;PA,PU,%.4f,%.4lf;",peak_t[0]*50,point1*50);
  ibwrt(plotter,name,strlen(name));
  sprintf(name,"PA,PD,%.4f,%.4lf;",peak_t[trial-1]*50,point2*50);
  ibwrt(plotter,name,strlen(name));


/*
   calculate the y-intercept which is nothing but a, scale it
   back to actual value . then calculate capture cross section
   area as
```

```
                 sigma = 1/gamma*(exp a )

   and slope b is proportional to delta E and calculate delta E
   as

                 delta E = -1000*K*b

   where K is boltzman constant.
*/

/* scale back the value of a, siga, deltae sigb(deviation in
   deltae)
*/

                sigma = exp(a)/gamma;

                siga = exp(siga)/gamma;

                deltae = (-1000.0)*(8.62E-5)*b;

                sigb = (-1000.0)*(8.62E-5)*sigb;

/* write the energy, capture cross-section and their deviations */

    sprintf(name,"PA,PU,13052,25550;CP0,%d;",down);
    ibwrt(plotter,name,strlen(name));
    sprintf(name,"LBenergy is %.2g(%.2g) eV\3;",deltae,sigb);
    ibwrt(plotter,name,strlen(name));
    down--;
    sprintf(name,"PA,PU,13052,25550;CP0,%d;LBcapture ",down);
    ibwrt(plotter,name,strlen(name));
    sprintf(name,"cross-section is %.2g(%.g) sqcm\3;"sigma,siga);
    ibwrt(plotter,name,strlen(name));
    down--;

/* return the pen to 0,0 position */

                ibwrt(plotter,"PA,PU,0,0;SP0;",14);

/* check if user wants to remove any point */

                puts("\ndo you want to reject any point");

                response = bdos(1) &0xFF;

/* if yes get the point number and remove it */

                if(response == 'y') {

                   puts("\nenter the point number");
                   scanf("%d",&k);
                   for(j=k-1; j<trial-1; j++) {
```

```
                              expn[j] = expn[j+1];
                              e_s[j] = e_s[j+1];
                              p_s_t[j] = p_s_t[j+1];
                              peak_t[j] = peak_t[j+1];
                        }
                        trial -= 1;
                     }

/*
   go back and fit another line and check if user wants still
   more changes
*/

                 } while(response == 'y');

                     ibwrt(plotter,"PA,PU,0,0;SP0;",14);

/* call plot write routine to lable the axes and title the
   plot
*/
                     getchar();

                     plot_write(min_val,max_val,ftemp1,ftemp2);
               }


}
```

```
readdata()
{
   extern int buff[512],reject;
   extern char *fp;
   int i,total,number,num_read;
   unsigned char buffer[1028],str[3];

/* digitizer sends data in the following format -> % count data
   checksum ; so when reading the data first look for % sign and
   then read 1028 bytes-two for count,1024 for 512 datapoints, a
   checksum and one terminator(;)
*/

         do {
             fread(str,1,1,fp);
         } while(str[0] != '%');
         i = 0;
         total = 1028;
         do {

            num_read = fread(&buffer[i],1,total,fp);
            total = total-num_read;
            i += num_read;

         } while(total > 0);


/* generate integer values for data points. First byte is high
   byte and second is low.
*/

         number = buffer[0]*256+buffer[1];

         reject = 0;

         for(i=0; i<1024; i+=2) {

/* if the number is greater than 15 bits, it is garbage and
   reject the data set. set reject flag
*/

            buff[i/2] = (buffer[i+2]*256+buffer[i+3]);
            if((buff[i/2] < 0) || (buff[i/2] > 32767))
                     reject = 1;
         }

}
```

```
double fit()

{

extern double e_s[25],a,b,siga,sigb,sqrt();
extern float p_s_t[25];
extern int trial;
double exp_s,chi2,sigdat;
float temp_s,st2,diff,temp_avg;
int j;

      a = 0;   /* set constant term to 0 */
      b = 0;   /* set slope to 0 */
      st2 = 0;
      exp_s = 0;   /* set y-data sum to 0 */
      temp_s = 0;  /* set x-data sum to 0 */

/* sum x and y data */

      for(j=0; j<trial; j++) {
            exp_s += e_s[j];
            temp_s += p_s_t[j];
      }

/* calculate average of x data */

      temp_avg = temp_s/(float)(trial);

/* find the difference of every x data point from average value
   sum the square of difference and calculate sum the diferrence
   times corresponding y value.
*/

      for(j=0; j<trial; j++) {
            diff = p_s_t[j]-temp_avg;
            st2 += diff*diff;
            b = b+diff*e_s[j];
      }

/* get slope by dividing sum of product of ydata and diferrence
   of x data from avg with sum of square of diferrence
*/

      b = b/st2;

/* calculate y-intercept or constant term */

      a = (exp_s-temp_s*b)/(float)(trial);

/* calculate the standard deviations */

      siga = sqrt((1.+temp_s*temp_s/(trial*st2))/(float)(trial));
      sigb = sqrt(1./st2);
      chi2 = 0;
      for(j=0; j<trial; j++) {
         chi2 += (e_s[j]-a-b*p_s_t[j])*(e_s[j]-a-b*p_s_t[j]);
      }

      sigdat = sqrt(chi2/(trial-2));
      siga *= sigdat;
      sigb *= sigdat;

}
```

```
sort_f()
{

extern float temp[750];
extern int cl2[750][25],count,trial;
int i,itemp,flag,j;
float ftemp;


/* sort the temp and data arrays by temperature in ascending
   order using bubble sort
*/

            do {
               flag = 0;
               for(i = 0; i < count-1; i++) {
                     if(temp[i] > temp[i+1]) {
                           ftemp = temp[i];
                           temp[i] = temp[i+1];
                           temp[i+1] = ftemp;
                           for(j=0; j<trial; j++) {
                                 itemp = cl2[i][j];
                                 cl2[i][j] = cl2[i+1][j];
                                 cl2[i+1][j] = itemp;
                           }

                           flag = 1;
                     }
               }
            } while(flag == 1);

}
```

```
plot_init()
{
   extern int *plotter;
   int i,j;
   unsigned char strl[80];

   /* call up the plotter using the National Instrument
      routines.
   */

   plotter = ibfind("dev5");

   /* initialize, default, select penl, window size, tick
      length 1
   */

   ibwrt(plotter,"IN;DF;SP1;IP 1000,1000,9000,7000;TL1;",37);

   /* scale to data */

   ibwrt(plotter,"SC0,25600,0,25600;",18);

   /* draw x-axis */

   ibwrt(plotter,"PA,PU,0,0;PA,PD,25600,0;",24);

   /* draw y axis */

   ibwrt(plotter,"PA,PU,0,0;PA,PD,0,25600;",24);

   /* draw tic marks, x-axis and then y-axis */

   for ( i = 0; i < 25600 ; i+=2560) {
      for ( j = i; j <= i + 2560 - 2*512; j += 512 ) {
         sprintf(strl,"TL.5,0;PA,PU,%d,0;XT;",j+512);
         ibwrt(plotter,strl,strlen(strl));
      }
      sprintf(strl,"TL1,0;PA,PU,%d,0;XT;",i+2560);
      ibwrt(plotter,strl,strlen(strl));
   }

   for ( i = 25600; i > 0 ; i-=2560) {
      sprintf(strl,"TL1,0;PA,PU,0,%d;YT;",i);
      ibwrt(plotter,strl,strlen(strl));

   for ( j = i; j > i - 2560 + 512; j -= 512) {
         sprintf(strl,"TL.5,0;PA,PU,0,%d;YT;",j-512);
         ibwrt(plotter,strl,strlen(strl));
      }
   }
}
```

```
plot_raw()
{

extern int *plotter;
extern int buff[512];
int i;
char strl[80];


   /* move to first location before plotting the rest of the
      data
   */

         sprintf(strl,"SP1;PA,PU,0,%d;",buff[0]*50);
         ibwrt(plotter,strl,strlen(strl));

   /* now continue plotting the rest of the data */

         for ( i = 1; i < 512; i++ ) {
            sprintf(strl,"PA,PD,%d,%d;",i*50,buff[i]*50);
            ibwrt(plotter,strl,strlen(strl));
         }



         ibwrt(plotter,"PA,PU,0,0;SP0;",14);

}
```

```
plot()

{

extern int count,trial,cl2[750][25],*plotter;
extern float temp[750];
unsigned char strl[81];
float min_val, max_val,tempf;
int i,j;

   /* plot routine for conventional DLTS plots */

      for(j=0; j<trial; j++) {

   /* move to first location before plotting the rest of the
      data
   */

         sprintf(strl,"SP1;PA,PU,0,%d;",cl2[0][j]*50);
         ibwrt(plotter,strl,strlen(strl));

   /* now continue plotting the rest of the data */

       for ( i = 1; i < count; i++ ) {
       tempf = (temp[i]-temp[0])*511.0/(temp[count-1]-temp[0]);
       sprintf(strl,"PA,PD,%.4f,%d;",tempf*50,cl2[i][j]*50);
       ibwrt(plotter,strl,strlen(strl));
       }


         ibwrt(plotter,"PA,PU,0,0;SP0;",14);
      }

}
```

```
plot_c()
{

extern int trial;
extern double expn[25];
extern float peak_t[25];
extern int *plotter;
int j;
unsigned char strl[80];

      ibwrt(plotter,"SP1;",4);

      for(j=0; j<trial; j++) {

   /* plot the individual points on arrhenius plot*/

    sprintf(strl,"PA,PU,%.4f,%.4lf;",peak_t[j]*50,expn[j]*50);
    ibwrt(plotter,strl,strlen(strl));
    ibwrt(plotter,"CP-.25,-.25;LB+\3;",17);
    }


         ibwrt(plotter,"PA,PU,0,0;SP0;",14);
}
```

```
plot_write(xmin,xmax,ymin,ymax)

float xmin,xmax,ymin,ymax;

{
extern int *plotter;
extern char *stdin;
int times,i;
unsigned char str1[81],str2[80],str3[80],str4[80];
float offset,x_scale,x_var,y_scale,y_var;



   ibwrt(plotter,"PA,PU,0,0;SP1;",14);

/* divide x and y axes in ten parts */

   x_scale = (xmax-xmin)/10;
   y_scale = (ymax-ymin)/10;

   /* label x and y tick marks */

      times = 0;
      for (i=0; i<=25600; i+=2560) {
         x_var = x_scale*times+xmin;

/* generate the character string to write to plotter */
         sprintf(str2,"%.2f",x_var);
         sprintf(str1,"PA,PU,%d,0;CP-%d,-1;LB%s\3;",i,
                                      (strlen(str2))/2,str2);
         ibwrt(plotter,str1,strlen(str1));
         times +=1;
      }


      offset = 0;
      times = 0;
      for (i=0; i<=25600; i+=2560) {
         y_var = y_scale*times+ymin;

/* generate character string to write to plotter */

         sprintf(str2,"%.2f",y_var);
         sprintf(str1,"PA,PU,0,%d;CP-%d,-.25;LB%s\3;",i,
                                         strlen(str2),str2);
         ibwrt(plotter,str1,strlen(str1));
         times +=1;
      }

   /* write the title in the middle at the top */

      puts("enter the plot title");
      fgets(str4,80,stdin);

      sprintf(strl,"SI.25,.4;PA,PU,12800,26800;CP-%d,0;",
                                              strlen(str4)/2);
      ibwrt(plotter,strl,strlen(strl));

      sprintf(strl,"LB%s\3;",str4);
      ibwrt(plotter,strl,strlen(strl));

/* write x axis lable */

      puts("enter x-axis units");
      fgets(str4,80,stdin);

      sprintf(strl,"SI.2,.32;PA,PU,12800,0;CP-%d,-3;",
                                              strlen(str4)/2);
      ibwrt(plotter,strl,strlen(strl));

      sprintf(strl,"LB%s\3;",str4);
      ibwrt(plotter,strl,strlen(strl));

/* write y axis lable */

      puts("enter y-axis units");
      fgets(str4,80,stdin);

      sprintf(strl,"PA,PU,0,12800;DI0,1;CP-%d,3;",
                                              strlen(str4)/2);
      ibwrt(plotter,strl,strlen(strl));

      sprintf(strl,"LB%s\3;",str4);
      ibwrt(plotter,strl,strlen(strl));

      ibwrt(plotter,"PA,PU,0,0;SP0;",14);
}
```

COMPUTER CONTROLLED
DEEP LEVEL TRANSIENT SPECTROSCOPY
SYSTEM

by

HEMANT MEHTA

B.E. (Hons) EEE

Birla Institute of Technology and Science
Pilani, India

1983

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of

the requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

Durland Hall

Kansas State University

Manhattan, Kansas 66506

1986

Abstract:

The analysis of transient phenomena originating from relaxation process is a tool for material and device characterization. A convenient method for the analysis of exponential transients is the use of correlation spectroscopic techniques. An example is deep level transient spectroscopy (DLTS). It is demonstrated here how higher flexibility is achieved by separating signal processing from the data acquisition. A flexible microcomputer based automated DLTS system is described. The hardware is a high resolution capacitance measurement system. As a result of measurement which is considerably faster than analog methods, a data array describes the response of the sample with regard to time and temperature. Signal processing routines are then applied to data. Critical trap parameters may be simultaneously measured during one thermal scan.

1430-227
CA96