Modeling, Control, and Experimentation of a Scanning Tunneling Microscope

by

Jungmok Bae

B.S., Mechanical Engineering (1992)

Seoul National University

Submitted to the Department of Mechanical Engineering

in Partial Fulfillment of the Requirements for the degree of

Master of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

May, 1994

Signature of Author _____

Department of Mechanical Engineering
May 6, 1994

Certified by _____

Dr. Kamal Youcef-Toumi
Associate Professor
Thesis Supervisor

Accepted by _____

Dr. Ain A. Sonin
Chairman, Department Committee on Graduate Students

Modeling, Control, and Experimentation of a Scanning Tunneling Microscope

by

Jungmok Bae

Submitted to the Department of Mechanical Engineering
on May 6, 1994 in partial fulfillment of the
requirements for the degree of Master of Science in
Mechanical Engineering

ABSTRACT

The purpose of this thesis work is to integrate the high performance digital control system to the scanning tunneling microscope that has been built in the Laboratory of Manufacturing and Productivity at MIT. In the proposed control system, the 80486 personal computer was used to handle the user interface and the ADSP21000 family digital signal processor embedded in a slave board was used to implement the real-time controls. The new control system can execute 100 instruction control code in 3 μsec. The system is also capable of 14 bit data acquistion at a 1 MHz rate and a 16 bit output resolution. The experiments of taking atom image in both the constant height mode and the constant current mode are performed and the results are evaluated. The model of the PID feedback loop control implemented in constant current mode is built and tested. The derived model is used to identify each component's effect on the overall system's response.

Thesis Supervisor: Dr. Kamal Youcef-Toumi

Title: Associate Professor of Mechanical Engineering

Acknowledgement

I would first like to thank my research advisor Prof. Youcef-Toumi for his generous guidance and support. Without him, I would have never learned so many things as I did here.

I also want to thank Tetsuo for giving me this great opportunity of working on such exciting topics and supporting me in many ways through the end of the thesis work. I really appreciate such invaluable help he gave to me. I also thank his wife, Renee for transfering the good quality atom iamges.

I also want to thank T.J. on his generous advices. I really appreciate that he was never been busy when I had the questions. I enjoyed and will enjoy working in the same lab with him.

I want to thank also other lab members Jason (Yong-jun), Tarzen, Woosok, Shigeru, Francis, and Doug. I appreciate the help of Doug and David in finishing the 2.171 course safely. I also appreciate the help of Jim Gort in learning the DSP.

I would like to thank Prof. Chun and Prof. Suh for the help of my settlement here at MIT.

I thank Geehun for making my life a lot enjoyable at the MIT campus. That goes to all my friends I made here at Boston.

I want to thank also my little brother and sister who always have been cheerful to me.

Lastly and most importantly, I would like to thank my parents who gave their continued love, cheers, and support.

Contents

List of Figures

Chapter 1
Introduction

1.1 Motivation

The project of developing the high precision system based on the scanning tunneling microscope technology is being currently underway at the Laboratory for Manufacturing and Productivity at Massachusetts Institute of Technology. Prior to the application development, the prototype of low cost in-house scanning tunneling microscope was built and tested in order to acquire the related technology and to investigate other possible areas of applications. Within this scope, the object of this thesis was to enhance the performance of the previous built prototype by integrating the digital signal processor based control system.

Recently, as digital signal processor technology has grown very fast, direct emulation of analog circuits using digital signal processing has become possible.[10,11,12,13] Such DSP-based system provides greater flexibility in the choice of the system parameters and control algorithms. Thinking that DSP based STM control system will facilitate the development of such complicated applications that are being planned in the project, our group laid out and initiated the work done in this thesis. The DSP hardware provided by JRG is based on the modified Harvard architecture of the implemented digital signal processor. The board features the on-board RAM memory, the digital I/O port, the A/D and D/A converters, and the communication paths between the host computer and the DSP. The board is one of the fastest version of data conversions and the computations among the commercially available products

present.

1.2 Contents of Thesis

This thesis is organized as follows. Chapter 2 lists all the neccessary information in running the instrument. The detailed explanation is given for the hardware of the STM built by Ohara. After a brief introduction to the STM fundamentals, the procedure of atom surface scanning is described.

Chapter 3 provides a mathematical model of the scanning tunneling microscope control system, especially focused on the PID feedback loop performed during constant current scanning. The model was derived by matching the frequency response. The derived model is assessed by comparing its step responses with the system's real responses.

In Chapter 4, the algorithms and the functions of the high-level softwares are described. The currently developed control software implemented in the host PC computer is explained in depth. The steps neccessary for developing the application software is also explained.

In Chapter 5, the algorithms and the roles of each modules of low-level software are described. The descibed module includes the real-time control software such as coarse positioning, PID control of the z-axis, and scanning. In this chapter, the line by line description of the software is avoided. Instead, the design considerations and some related techniques used in software implementation are discussed. The detailed description of the software is again given in the appendix B and C.

In Chapter 6, the results of the scanning experiments are presented. The description of the graphite atom structure is given in order to give better understanding of the atomic image we have taken.

Finally, the conclusion is provided in the last chapter.

Chapter 2
STM Operations

2.1 Introduction

This chapter provides a basic understanding of a scanning tunneling microscope that has been built in the laboratory of flexible automation at Massachusetts Institute of Technology. In the first section, the overall description of the scanning tunneling microscope is given. The second section discusses the piezoelectric tip positioner. This section is divided into two subsections which explain the characteristics of the mechanical part and the electrical part of the piezo scanner. In the third section, the pre-amplifier and filters are described. Data is provided to show the circuit's ability in rejecting noise. The next section discusses the log conversion of the tunneling voltage data and the following section describes how the tip is prepared. In the last section, the procedure of atom surface scanning is explained.

2.2 Description of STM system

The scanning tunneling microscopes that are currently in use in various laboratories and other research areas have been designed to meet specific purposes. Different technologies and ideas are put into each one of them. However, the author found out that their overall structures are quite similar and can easily be described with a general terminology. The following paragraphs provide the definition of such terms and explain how they can be fit to our scanning tunneling microscope.

The system can be divided into four units. These are the system control unit,

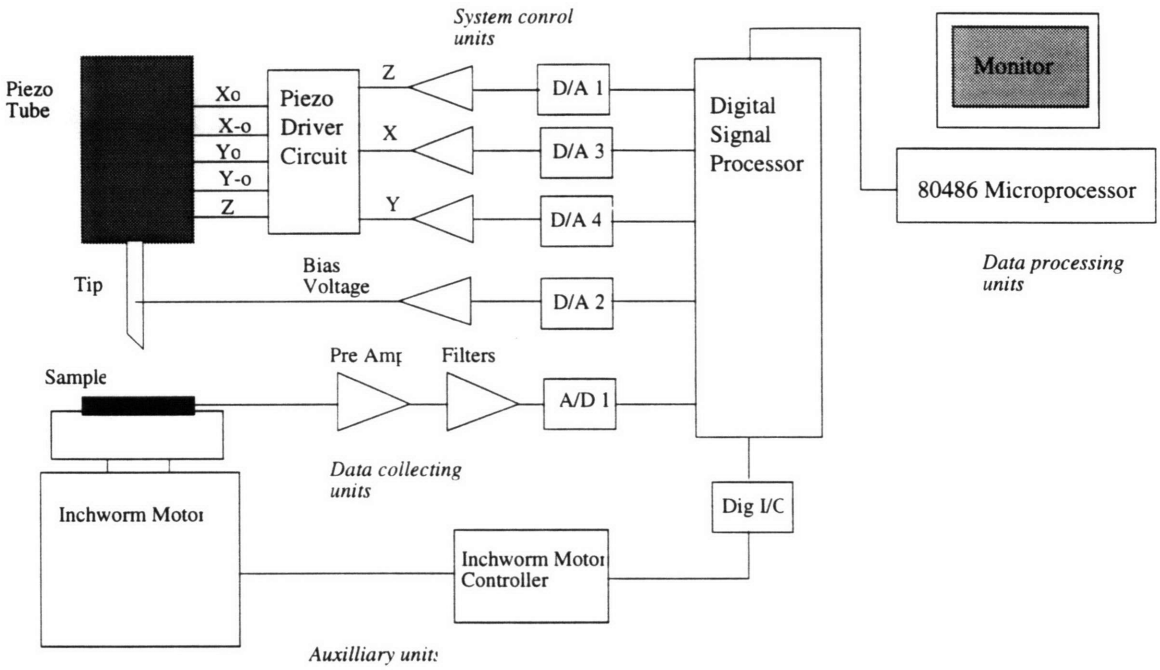


Figure 2.1: Overall STM Configuration

the data acquisition unit, the data processing unit, and the auxiliary units. These are shown schematically in the figure 2.1. The system control unit controls the tip position. In our system, the digital signal processor(DSP in short), the piezoelectric tube and its driver circuit fall into this category. When the user inputs the x, y, z voltages of the piezoelectric tube, the digital signal processor produces the corresponding control voltages and sends the voltages through the D/A port to a circuit called a piezoelectric tube driver where, again, the five output voltages, -x, x, -y, y, z are produced. Each output voltage is directly applied to the electrodes on the piezoelectric tube. This voltage causes the piezo tube to expand, contract, or bend. The detailed descriptions are given in section 2.3.1.

The data acquisition unit collects the atom image data and transfers it to the computer for analysis. Generally, the unit consists of the tip, the sample, the ampli-

fiers, and the filter circuits. During the scanning process, the tunneling current flows through the sample from the tip. Then, the signal corresponding to the tunneling current, is amplified by a pre-amplifier up to a level appropriate for the DSP board. The signal is filtered again before input to the DSP. A notch filter at 60 Hz and a 4th order low pass filter cutoff frequency at 1 kHz are used in our system. Section 2.4.2 presents an in depth discussion of the characteristics of the filters.

The data processing unit analyzes and displays data. The unit is mainly the computer. During scanning, the acquired data are stored temporarily in the DSP's memory. When the scanning is completed, the data is immediately transferred to the personal computer. Then, the computer analyzes the data, displays it to the user, and stores the result into the disc. Chapter 5 is devoted to the computer program written for such purposes.

The auxiliary unit conducts the coarse positioning of the tip and the sample. In our case, it corresponds to the inchworm motor. The motor has its own controller which takes commands from the DSP and sends the corresponding control current to the inchworm motor. The reader should refer to section 2.7.2 and the manufacturer's note for more descriptions of the inchworm motor control.

Each unit has to be carefully considered in order to produce good outputs. Although the structural design is a critical factor in producing high quality output, it is not taken into account in this diagram.

2.3 Piezoelectric Scanner

This section is devoted to the piezoelectric scanner used for the tip positioning. In the first part of the section, the piezoelectric tube is described. The sensitivity of the tube was calculated from the experimental data. Likewise, the natural frequency of the piezoelectric tube was also measured experimentally. The detailed descriptions

of the experimental procedures are provided.

The second part of the section describes the piezoelectric tube drive circuitry. The charcterization of the circuit has been done in this part through the frequency response plot of each input and output.

2.3.1 Piezoelectric Tube Transducer

The most widely used form of piezoelectric tip positioner in a scanning tunneling microscope is the tube. This is because the tube has a higher resonant frequency, greater range, and greater thermal resistance than other form of piezoelectric actuators. Note that the high resonant frequency allows fast scanning speed and greater rejection of ambient vibration.[9] The piezoelecrtic tube used in the experiment is manufactured by Stavely Sensors, Inc. The length of the tube is 1 inch with outer diameter of 0.25 inches, and thickness of 0.02 inches. The piezoeletric material inside the tube is the EBL#2 version of PZT.

As shown in figure 2.2 a), the outer surface and the inner surface are surrounded by the electrodes while the inside is filled with piezoelectric material. When the voltage is applied to the inner and outer electrodes, the piezoelectric material expands to the longitudinal direction of the tube.

The outer electrode of the piezo tube is sectioned into four quadrants. Each quadrant is connected to one of the voltage sources from the driver circuit as shown in figure 2.2 b). Two voltage outputs, Vxo and V-xo, are connected to the opposite quadrants while Vyo and V-yo are connected to the remaining two. This configuration of the voltage connections is to improve the linearity.[15] The Vxo and V-xo, when applied, cause the bending motion of the tube in x-direction and, Vyo and Y-yo cause the bending motion in the direction perpendicular. The Vzo is connected to the inner quadrants. The Vzo is held constant except when the piezoelectric tube is to expand

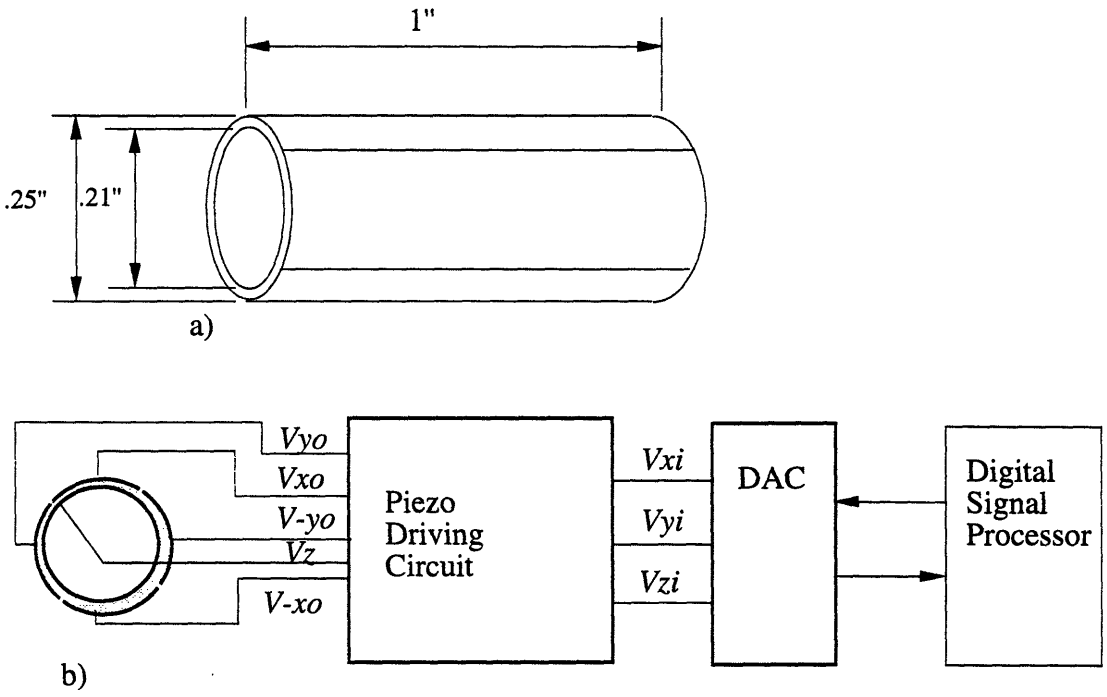


Figure 2.2: a) Piezo Tube b) Piezo Tube Voltage Connection

in the longitudinal direction.

In positioning the tip to the precise location, it is important to know the accurate relationships between the applied voltage and the displacement of the piezoeletric tube in x, y, z direction. The relationships was found by conducting the experiments and calculating from the equations provided by the manufacturer. A detailed explantion is given below.

X and y direction sensitivity

The x and the y direction sensitivities were calibrated from the atom image that the piezoeletric tube had produced. The atom image used in the calibration is shown in figure 2.3. From the figure, the distance from the apex to nearest apex were measured. The distance was measured to be 27.6 mm. The figure 2.3 a) shows the data that

has been measured and averaged. Next, the ratio between the atomic image of figure 2.3 and the real atomic dimension were derived. The real dimension of the distance between the apex is known to be 2.46 Å.[21] The ratio was calculated by just dividing this value by the measured value. The result was as follows.

$$\frac{\text{Real World Dimension}}{\text{Diagram Dimension}} = 8.6956 \times 10^{-9}. \tag{2.1}$$

The next step was the calculation of the total x direction and y direction displacement. First the x direction length of the figure 2.3 was measured. The measured value was 95 mm. Then, using the ratio caluculated above, the real dimension of the x displacement was derived. The calculated total x direction displacement in real world dimension was 8.26 Å. During the calculation process, it was assumed that the angle between the x direction and the line connecting the apex of the atom were negligible. Finally, the x direction sensitivity was calculated by dividing the x direction displacement by the applied voltage which was known. The net voltage applied by the DSP was, read from the scanning program. The applied voltages were 15.57 mV for the x direction and 14.04 mV for the y direction. The calculated sensitivity in the x direction was

$$\frac{\text{Total x Direction Displacement}}{\text{Net Applied Voltage}} = 53.1\frac{nm}{V}. \tag{2.2}$$

By following the same procedure, the y direction sensitivity was at

$$\frac{\text{Total y Direction Displacement}}{\text{Net Applied Voltage}} = 50.14\frac{nm}{V}. \tag{2.3}$$

The averaged sensitivity of the piezoelectric tube was calculate at

$$\frac{\text{Displacement}}{\text{Applied Voltage}} = 51.62\frac{nm}{V}. \tag{2.4}$$

a)

Data	The Measurement
1.	26.5 mm
2.	29 mm
3.	28 mm
4.	27 mm
5.	27.5 mm
6.	26.5 mm
7.	28.5 mm
Average	27.6mm

b)

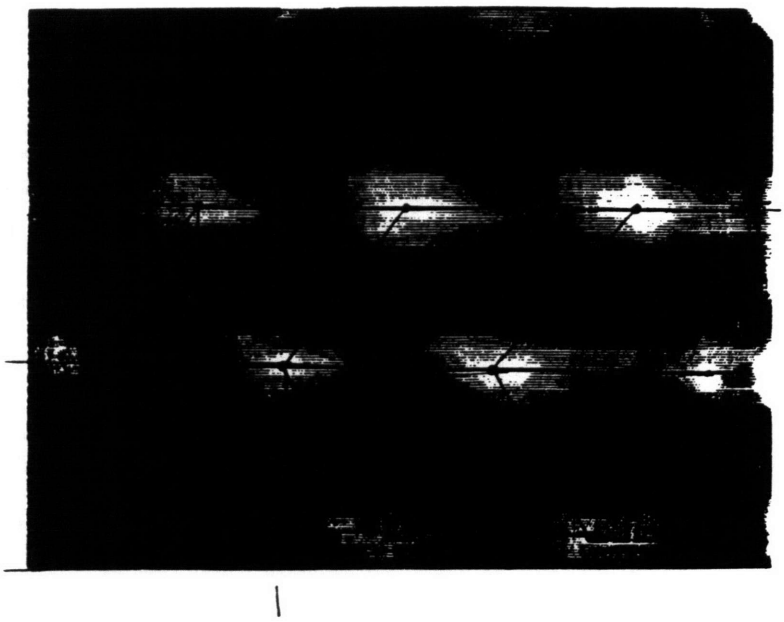


Figure 2.3: The Atom Image Used in the Calibration of the Piezo Tube

Z direction sensitivity

For the z direction sensitivity, the equation provided by the manufacturer was used. The equation is as follows.

$$\Delta L = \frac{2 d_{31} V L}{OD - ID} \tag{2.5}$$

The manuafacturer also provided the following material constants and the dimensions.

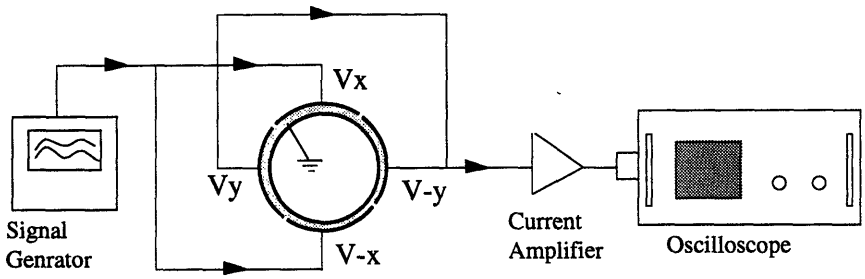
$d_{31} = -173 \times 10^{-12} \frac{meter}{volt}$.

$L = 25.4mm$.

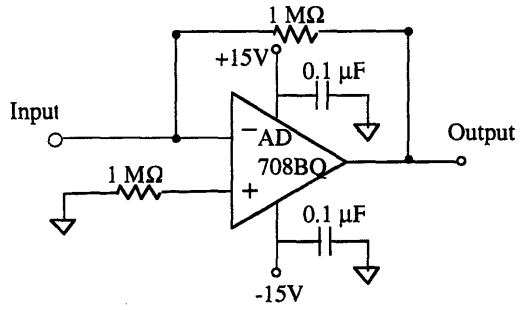
$OD - ID = 1.016mm$.

Substituting these known values to equation (2.5), we got the relationship between the applying voltage and the expanded length described as follows.

$$\Delta L(meter) = 8.7 \times 10^{-9} L \tag{2.6}$$



a)



b)

Figure 2.4: a) The Experiment Setups for the Measurement of the Piezotube's Natural Frequency b) The Current Amplifier Circuit Diagram

From the above equation, the z direction sensitivity was found to be 8.7 nm/volt.

Natural frequency

The resonant frequency of the piezo tube in longitudinal direction was also measured by conducting a simple experiment. Figure 2.4 a) shows the experiment set up for the measurement of the piezo tube's natural frequency. A sinusoidal signal was applied to the y electrodes using the function generator while the amount of the current generated in the x electrodes were measured. Because only a small amount of current is generated, usually the order of 10nA, the signal was amplified before the mea-

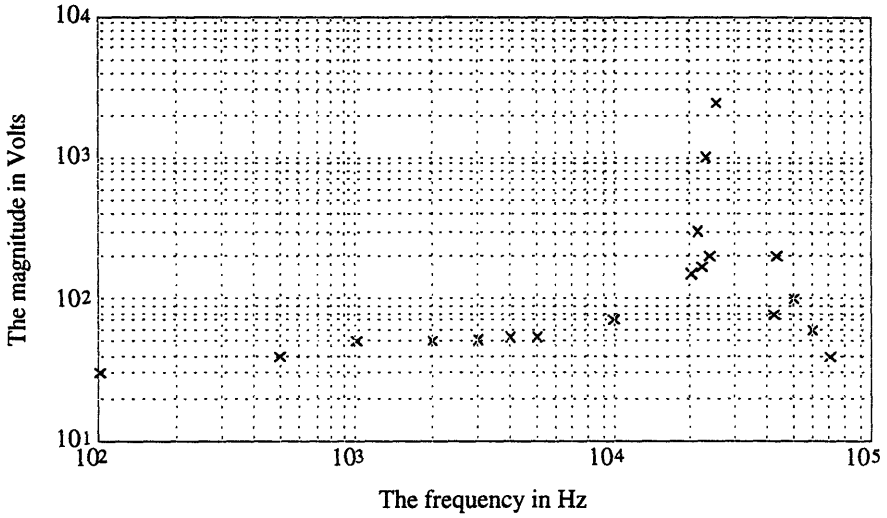


Figure 2.5: The Frequency Response of the Piezo Tube

surement. The current amplifier whose multiplying factor is 1 million, is shown in figure 2.4 b). The amplitude change of the output signal was measured through the oscilloscope while the frequency of the input is varied. The result was the frequency response plot of the piezotube as shown in figure 2.5. Each dot in the figure represents an experimental data point. One may refer to more detailed plot in chapter 3. The frequency response in this chapter was plotted using a signal analyzer.

The first resonant frequency was measured at 21kHz where the first peak in the magnitude plot appeared. The high frequency sound was produced from the piezotube when the input frequency moved close to the resonant frequency.

2.3.2 Piezoelectric Tube Driver Circuit

The circuit diagram is shown in figure 2.6. The diagram shows that the circuit has three inputs and five outputs. The two inputs, Vxi and Vyi are divided, inverted

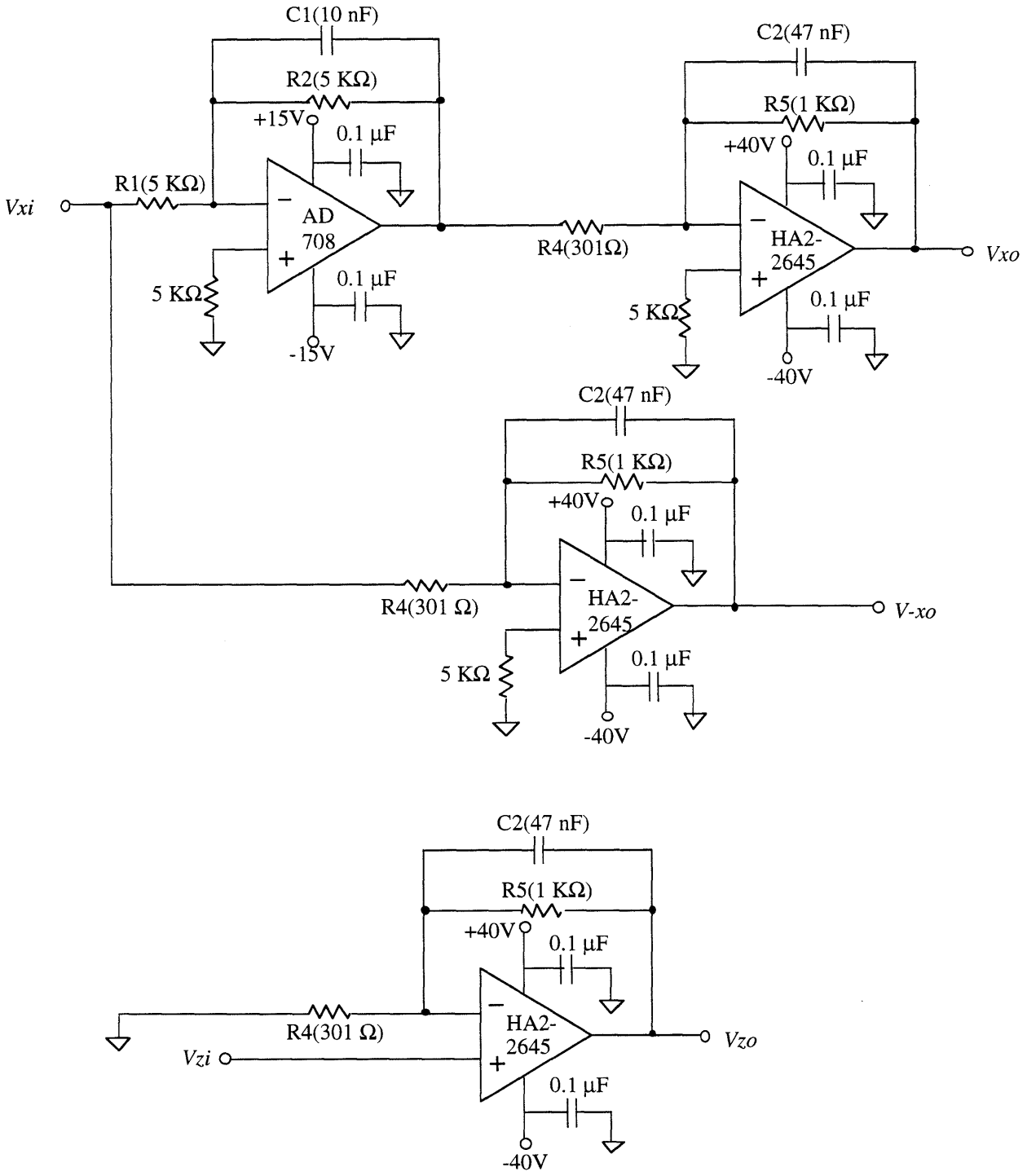


Figure 2.6: The Piezo Tube Driver Circuit

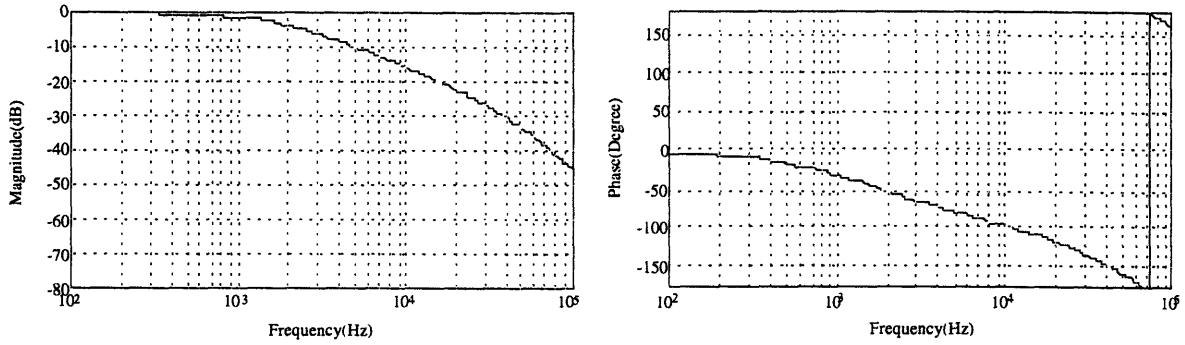


Figure 2.7: The Frequency Response of Vxo/Vxi

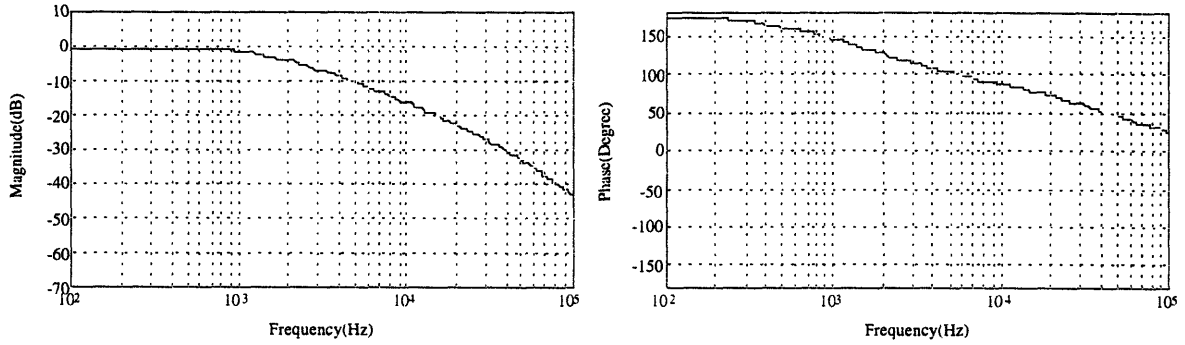


Figure 2.8: The Frequency Response of V-xo/Vxi

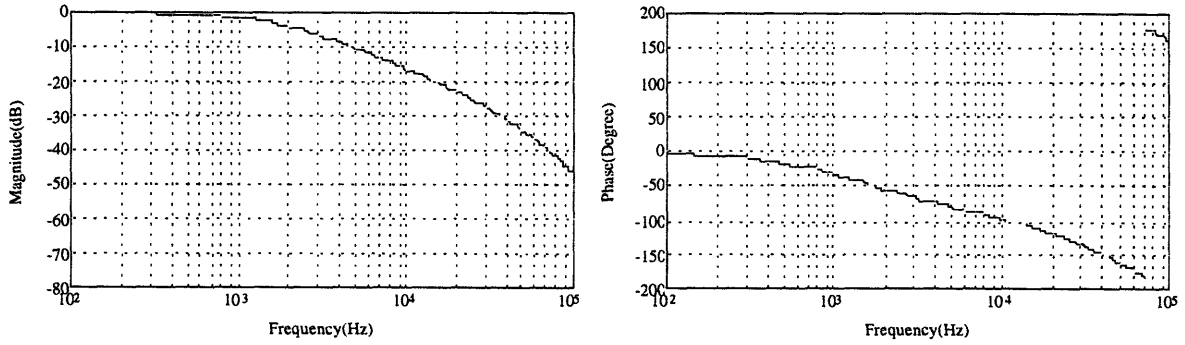


Figure 2.9: The Frequency Response of Vyo/Vyi

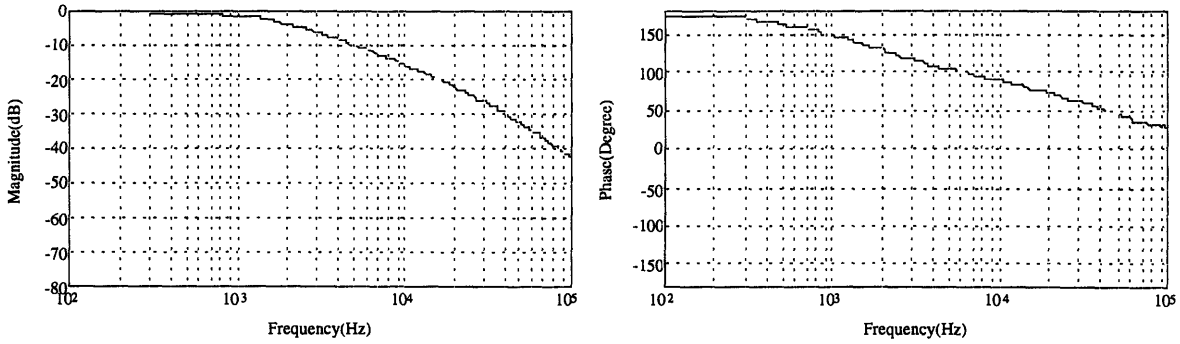


Figure 2.10: The Frequency Response of V-yo/Vyi

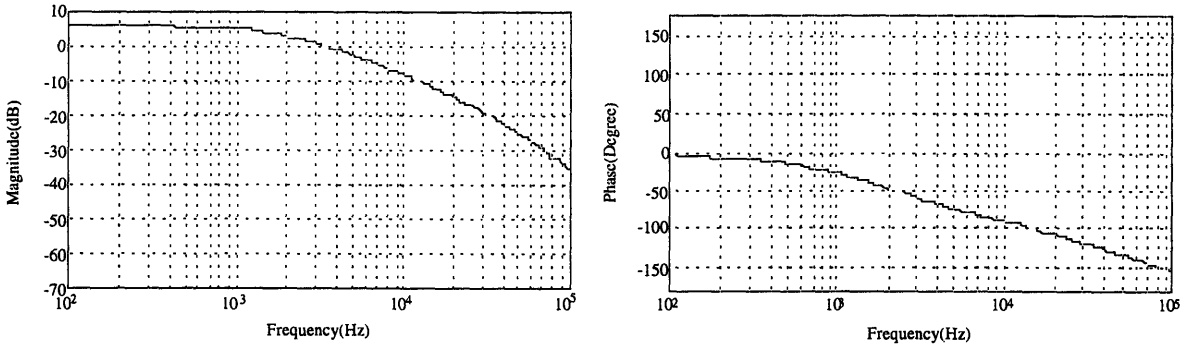


Figure 2.11: The Frequency Response of Vzo/Vzi

and amplified in the circuit. The result is the four outputs, Vxo, V-xo, Vyo, V-yo. Note that the voltages are amplified by a factor of 2. The Vzo is not inverted or divided, but it is also amplified by a factor of 2.

The frequency response of the piezoelectric driver circuit was measured for its characterization. The measured frequency responses of the five outputs, Vxo, V-xo, Vyo, V-yo, and Vz are shown in figures, 2.7 to 2.11. The average cut-off frequencies of the responses was measured at 1 kHz where the corresponding phase shift was 35 degree.

2.4 Output Amplifier Circuit

The amplifier circuit amplifies the tunneling current to a appropriate level so that the A/D converter can recognize the current's change. The tunneling current, usually generated in a range of 0.1 nA to 100 nA, is amplified up to a range of 10 mV to 10 V by the pre-amplifier in the circuit. Because such a small current is amplified, the outside noise has large effect on the signal. The low-pass filter is used to filter out the high frequency part of noise and the notch filter is used to remove the noise at 60 Hz. The first subsection is devoted to this pre-amlifier and the next section discusses the filters.

2.4.1 Pre-amplifier

The circuit diagram of the preamp is shown in figure 2.12. The preamp amplifies the input current by a gain of 100 million. The resulting output voltage is in the range of 10 mV to 10 V which is a reasonable voltage level for the A/D converter.

In order to reduce the noise while the small current flows through the circuit the path between the sample and the preamp is reduced to the smallest distance. The

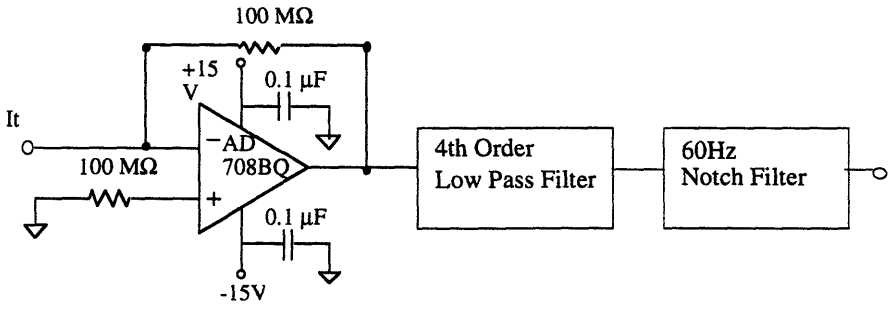


Figure 2.12: The Pre-amplifier and the filters

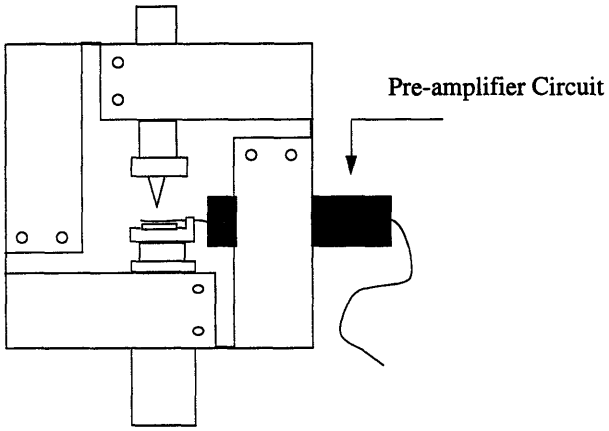


Figure 2.13: The Location of the Pre-amplifier

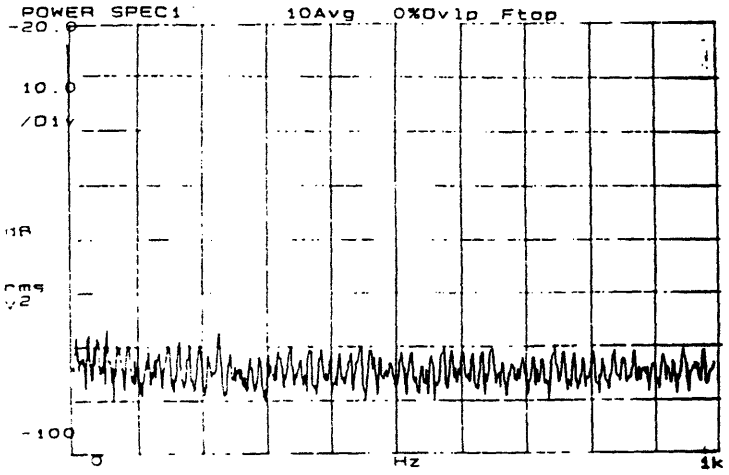
diagram 2.13 shows the location of the preamp in the scanning tunneling microscope.

2.4.2 Filters

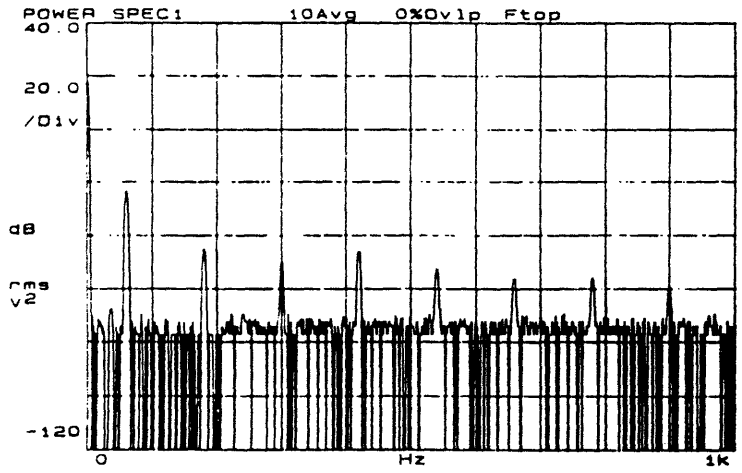
The low pass filter used in the circuit is a 6th order filter. The filter has a cutoff frequency at 1 kHz and shows a significant phase delay at the frequency of 200 Hz. The phase delay of the low pass filter shows that the speed is low compared to other part of the circuit. From the information provided in the previous section, one can point out that the response is even slower than the response of the piezoelectric tube whose lowest natural frequency is at several kHz. The speed can be increased by simply replacing a resistor but at the cost of amplifying noise in the high frequency range. Thus, there exists a trade off between the speed and the filter's performance.

The twin-T type notch filter is used for removing the noise at 60 Hz. Filtering out the 60 Hz noise is important because the signal representing the atomic image usually comes in at the same order of frequency.

a)



b)



c)

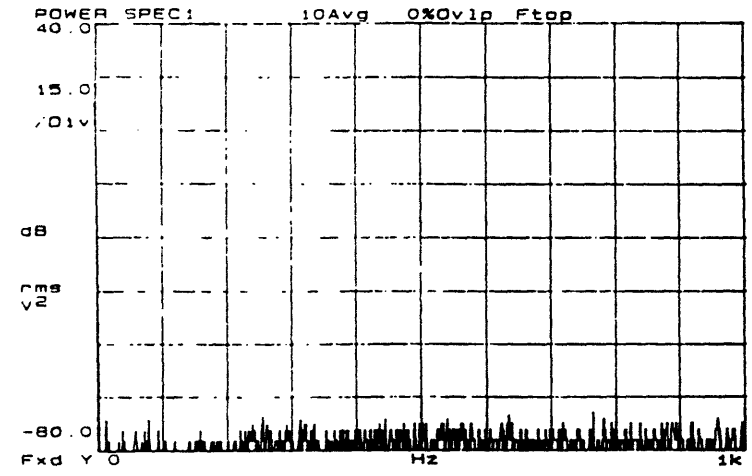


Figure 2.14: The Noise Level of the Amplifier Circuit a) FFT plot of the voltage source b) FFT plot of the signal before the filters c) FFT plot of the signal after the filters

In order to see the filters' performances, the FFT of the input signal before the filter and after the filter were measured. Figure 2.14 a) shows the noise level of the DC source input from the function generator. Figure 2.14 b) shows the noise level just after the current is amplified. In the figure, the noise of the input signal in harmonic form can be seen. The noise contains the frequency, and multiples of 120 Hz starting from 60 Hz. (The noise at 60 Hz is usually present in every circuits due to a nearest power supply). Notice that the noise components are completely eliminated in the output signal of the filter as seen in the FFT plot c).

As mentioned before, the low pass filter turns out to be the dominant component which determines the characteristics of the frequency response of the overall amplifier circuit. The actual measurement of the frequency response is shown in figure 2.15. It

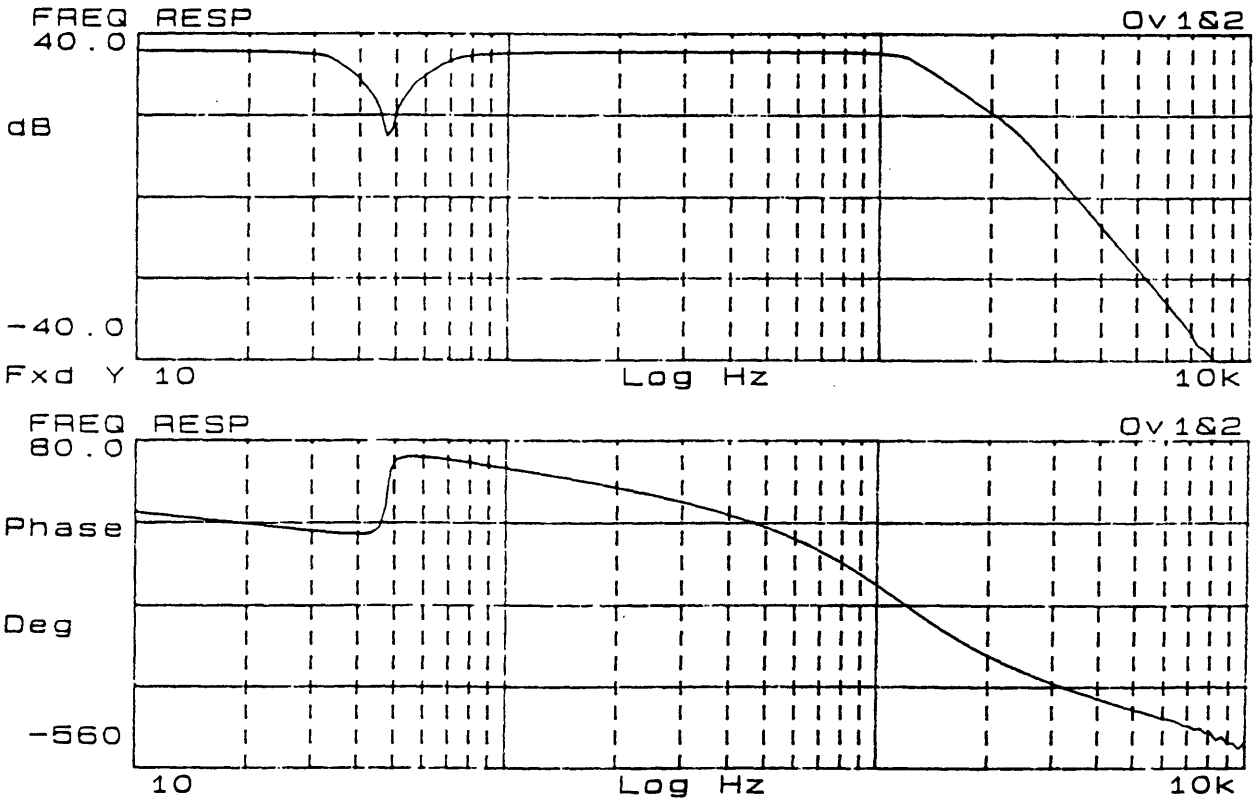


Figure 2.15: The Frequency Response of the Amplifier Circuit

clearly shows the characteristic of the low pass filter and the notch filter. Note that the phase has been shifted by 180 degrees up around the peak frequency of the notch filter. The typical phase characteristic of the notch filter due to the second order highly underdamped zeros in its transfer function.

2.5 Log Converter

The tunneling current and the gap are related by an exponential form. The detail explanation of the relationship is given in section 2.7.1. In order to calculate the gap distance from the measured tunneling voltage the logarithmic sheme is used to compensate for the exponential relation between the tunneling current and the gap.

The log conversion is performed digitally in the digital signal processor. At the very start of the program, the precalculated log table is downloaded to the specified address of the memory residing in the DSP board. Whenever the input value needs to be log converted, the matching value in the logtable is taken.

The address of each item in the log table represents the A/D converted value of the input voltage. Because only the positive values in the range of 0 to 5 volt are taken during the experiment, only the numerical values from 0 to 0x2000 are considered to be log converted. The memory space from adrress 0x2000 to 0x4000 in the data memory in the DSP board is used for the log table. Whenever data is to be log-converted comes in, its numerical value is increased by 0x2000. The result is the address which stores the corresponding log value.

The log converting equation implemented is based on the analog log-amplifier circuit that is no longer used in the system. The equation given in the specification of the chip is as follows.

$$V_{out} = K \left[\frac{log_{10}(V_{ref} R_{in})}{R_{ref}} \right] V_{in}, \tag{2.7}$$

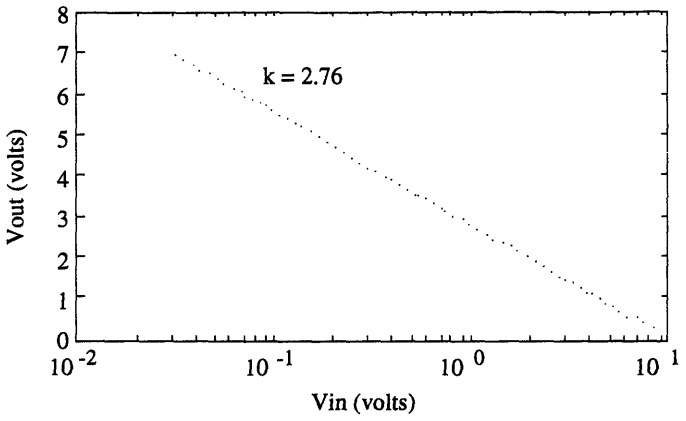


Figure 2.16: The Log Conversion

where K=2.76, R_{in}/R_{ref}=2, and V_{ref}=5V. These parameters were given from the circuit diagram of the analog amplifier.[16] The relationship between the output voltage and the input voltage of the log conversion has been plotted in the figure 2.16.

2.6 Tip Preparation

Before using the etching method, the tip was produced by a mechanical cutting method. The mechanical cutting of the tip was easy to implement but didn't provide consistent tip shapes. Many times the tip generated a unstable tunneling current and had to be replaced for the proper experiment.

The electrochemical etching method, which was adopted later, consistently produced a sharp tip. There are several articles[2,3,1] devoted to sharpening the tip by etching. In figure 2.17, our setup for the etching is shown. The tip wire is connected to the AC voltage from the function generator while the electrode is connected to the ground. When an AC voltage is applied, a bubble stream soon appears and start to form the tip shape. The bubble disappears after it finishes forming the tip shape.

During the etching process, there are several important parameters to consider.

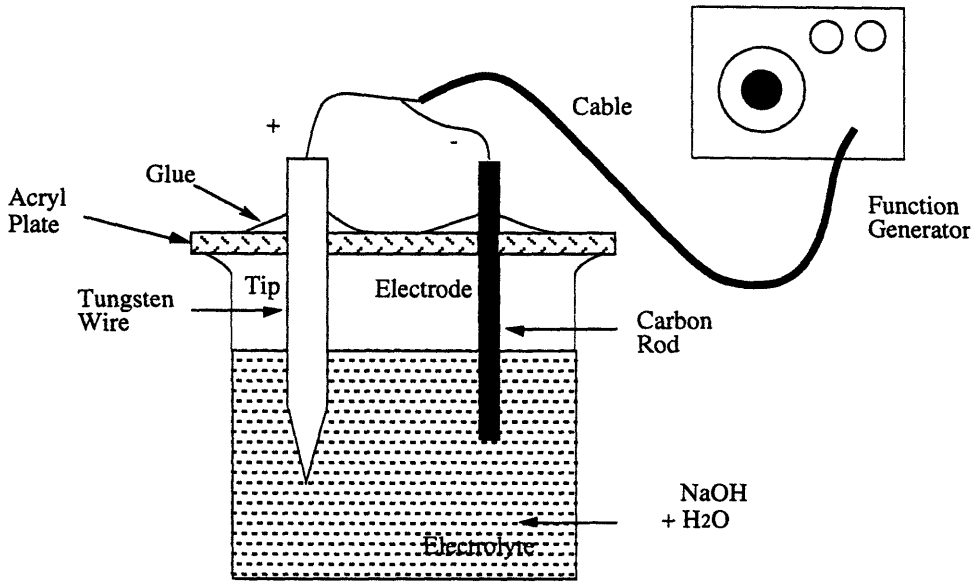


Figure 2.17: The Setups for the Etching of the Tip

The fllowing parameters, suggested by *Mircea Fotino*, are important in this procedure: ac voltage and current size, wave shape, frequency, phase angle, number of waves, depth of immersion, electrode shape, bubble dynamics, and concentration, temperature, and viscosity of the electrolyte.[1] These parameters have a large effect on the final tip shape and should be considered carefully.

Two materials, tungsten and Pt-Ir were used in making the tips. The Pt-Ir wire tip was produced by the former way of mechanical cutting. The wire was provided by Omega Engineering Inc. The tungsten wire was produced by etching. A tungsten wire of 0.27 mm in diameter was used. Before applying the AC voltage, the tungsten wire and its counter electrode, a carbon rod, were immersed into the electrolyte.(The electrolyte consists of 8 g of NaOH and 100 ml of H_2O.) The AC voltage ranged from 6 to 7 V was applied between the tungsten wire and its counter electrode. The bubble appeared immediately on the part of the tip immersed into the liquid and

lasted about 1 to 2 minutes. The etching process was finished when the bubble completely disappeared and the sharp edge of the tip was formed.

2.7 Atomic Image Acquisition Process

This section discusses the procedure of taking atomic image using the scanning tunneling microscope whose hardware description is given in the previous sections. This section is to cover the general idea of the overall process of scanning and leave the details of the software implentations in chapter 5 and chapter 6. Before going through each step of the scanning procedure, a brief review is given below.

The procedure of scanning the atomic surface is summarized as follows.

1. Coarse positioning of the gap using the inchworm motor only.
2. Fine positioning of the gap using the inchworm motor and the piezo tube.
3. Initial scanning height positioning using PID control scheme.
4. Scanning.
5. Display of the data recorded.

The first step is the positioning of the tip and the sample. In this STM, the inchworm motor is used to bring up the sample to the tip within the scanning range. There are two modes that are used to accomplish the positioning. The first mode, called coarse positioning, uses only the inchworm motor to reduce the gap. Using this positioning method, the gap is reduced at a fast rate. The process continues until the tunneling current is detected. The next mode, called the fine positioning, uses also the piezotube with the inchworm motor for the gap reduction. Knowing the tunneling current is detectable within closer range, the gap is reduced by a smaller step at a slower rate.

As soon as the tunneling current is detected the fine positioning is stopped and the tip is fixed to that position. In the next stage the tip is positioned to the specified initial height and finally the system starts scanning the surface.

2.7.1 Gap and Tunneling Current

To understand the relationship between the gap and the tunneling current one needs a broad knowledge of the field of solid state physics. Since such theories are not of our main concern, only brief descriptions are given below. The interested reader should refer to other detailed papers. [8,4,5]

The tunneling current equation starts from the model of the barrier Hamiltonian. The first order equation of the barrier Hamiltonian model, formalized by Bardeen[5], is given as follows.

$$I = \frac{2\pi e}{h} \sum_{\mu,\nu} f(E_\mu)[1 - f(E_\nu + eV)]|M_{\mu\nu}|^2 \delta(E_\mu - E_\nu), \tag{2.8}$$

where f(E) is the Fermi function, V is the applied voltage, $M_{\mu\nu}$ is the tunneling matrix element between states between the probe, ψ_μ, and the surface, υ_ν. E_μ is the energy of state ψ_μ for the tip in the absence of tunneling and E_ν is the energy of state ψ_ν for the surface in the absence of tunneling.

The above equation can be readily simplified if we assume that the voltage and the temperature are small.[4] The result is

$$I = \frac{2\pi e^2 V}{h} \sum_{\mu,\nu} |M_{\mu\nu}|^2 \delta(E_\nu - E_F)\delta(E_\mu - E_F). \tag{2.9}$$

The $M_{\mu\nu}$ is the one which reveals the exponential relationship. In order to calculate the $M_{\mu\nu}$, the wave functions for both the surface and the tip are needed. It is assumed that the the tunneling is one-dimensional planar in deriving the wave functions.[7,6] The assumptions result in a much simpler tunneling equation,

$$I_T \approx \rho_T V_T e^{-2\kappa s} \tag{2.10}$$

where,

I_T Tunneling current.

s Gap between the tip and the surface of the material.

ρ_T Tunneling conductance; the local density of electronic states at the Fermi level.

V_T Bias voltage applied between the tip and the sample.

κ Decay constant of a sample state near the Fermi level in the barrier region.

The parameter κ again can be represented by the following expression.

$$\kappa = \frac{\sqrt{2m\phi_\omega}}{h} \tag{2.11}$$

$$= 0.51\sqrt{\phi_\omega(eV)}\mathring{A}^{-1} \tag{2.12}$$

where m is the mass of an electron, h is Plank's constant, and ϕ_ω is the work function. The work function ϕ_ω can be described as the minimum energy required to remove an electron from the bulk to the vacuum level.

By using equation (2.10), we can find a relationship between the gap and the pre-amplifier's output voltage. If we define M as a gain of the pre-amplifier, we get the pre-amplifier's output voltage as a function of the gap distance as in the equation,

$$V_{pre} \approx M\rho_T V_T e^{-2\kappa s}. \tag{2.13}$$

The relationship between the pre-amp's output and the gap is plotted in figure 2.18. For the gain M the value, 10^8 is set as was stated in the section 2.4. Both variables, ρ_T and ϕ are dependent to the gap distance. However, because the gap is relatively small we assume this variable to be constant. The tunneling conductance, ρ_T is set to 30k Ω^{-1} which is a rough approximation for a typical STM.[25] The

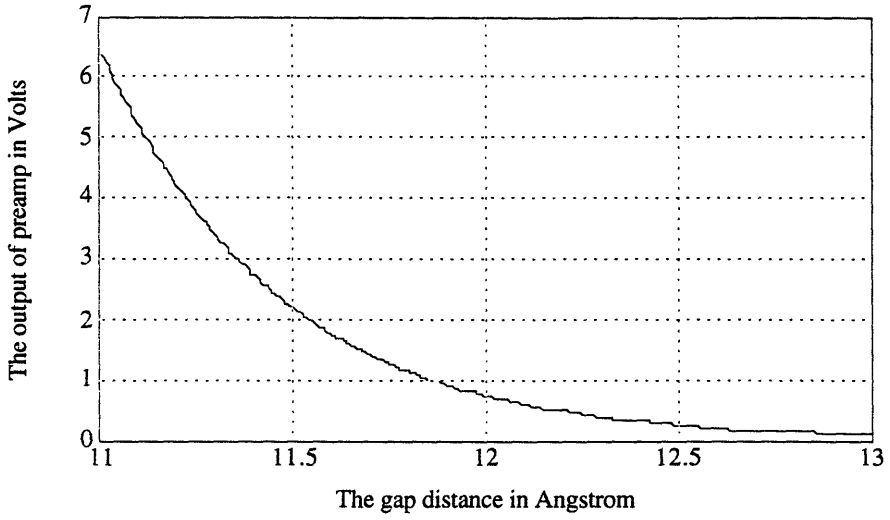


Figure 2.18: The Relationship between the Gap and the Output Voltage from the Pre-Amp

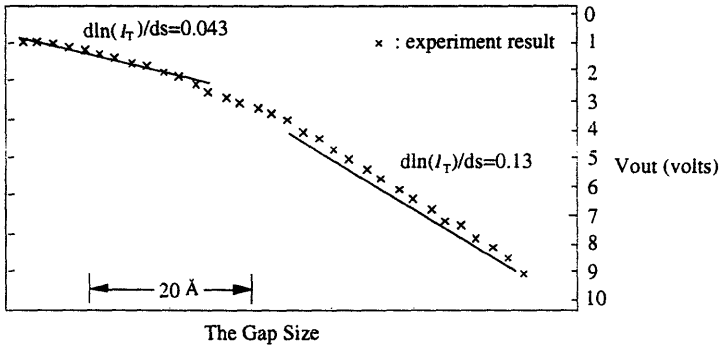


Figure 2.19: The Relationship between the Gap and the Output Voltage after Log Conversion

approximation has been made also for the work function ϕ to be 4.5 eV.[26] These are the typical values for a constant bias voltage with the elements such as platinum, tungsten, carbon, etc. The bias voltage, V_T is set to 50 mV, the usual value used in the experiment.

Next, the relationship between the gap and the output value after the log-conversion

is derived. Equation (2.13) is substituted in the log-conversion equation (2.7). The result is,

$$V_{log} = K(1 - log(M\rho_T V_T) + 2\kappa s log(e)) \tag{2.14}$$

In order to prove that the above equation truly holds, the experiment results of *Pahng* are used.[16] The experiment result is shown in figure 2.19. The slope of the experiment data turned out to be steeper than the theoretical data. The main causing factors can be the contaminations from surrounding air and the deformation of the graphite surface.

2.7.2 Approach Method

Just after the user fixes the tip and the sample, there is a wide gap between the tip and the sample. If the system is manually set by the user, the gap would be reduced up to around 1 mm at best, which means that several hundreds of micron have to be overcome before the atom scanning can be done. Because the piezo tube's working range is usually limited to around 100 nm, it is alone not suitable for initial positioning of the tip and the sample. Another means of reducing the gap is needed.

The inchworm motor is used to overcome this initial gap. The device has been purchased from the Burleigh Instrument, Inc. By using the walking mechanism of 3 pieces of the piezo materials, the motor achieves high resolution and wide moving range at the same time. The smallest single step is about 4 nm and the total range of motion is 25 mm. It can move at the maximum speed around 2 $\frac{mm}{sec}$.

In our system the inchworm motor is located under the working area facing upward as shown in figure 2.20. The motor is moving in a vertical direction to bring the sample up or down. The sample is fixed on top of the sample holder which is again glued at the top of the moving part of the inchworm motor.

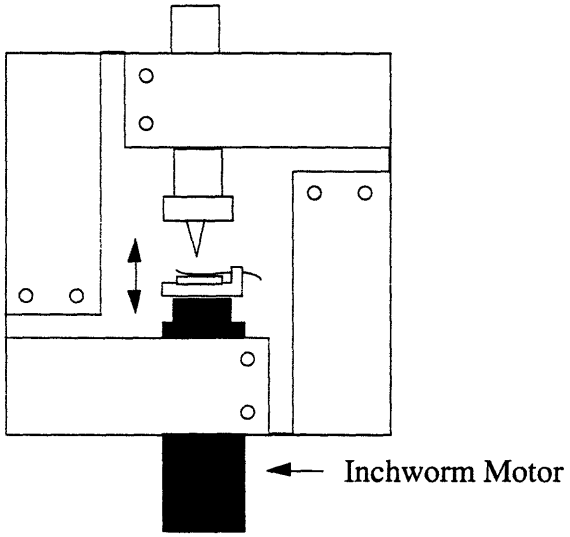


Figure 2.20: The Placement of the Inchworm Motor

Two modes, coarse positioning and fine positioning modes, are used for the gap reduction. The reason for using two mode operation is to minimize the total approaching time. During the first mode, the gap is reduced at a very fast speed but with low resolution. The motor usually operates at 2 $\frac{\mu m}{sec}$ in this mode. (In another words it takes approximately 8 minutes to move 1 mm.) The resolution at this mode is 4 nm which is very low considering that the tunneling current is generated within the range of less than 2 nm. During the positioning, the voltage output from the pre amplifier is measured as the inchworm motor is moved up from the initially set position. As soon as the tunneling current is detected, the inchworm motor stops its upward motion and pulls itself back several steps. Such motion is needed because otherwise the tip crashes to the sample. (It is observed that the tip crashes onto the sample usually by amount of 60 nm.) The reason is that the inertia caused by the constant forward motion may change the state in the piezo material inside the inchworm motor even though the voltage is held constant. Such change of the state may

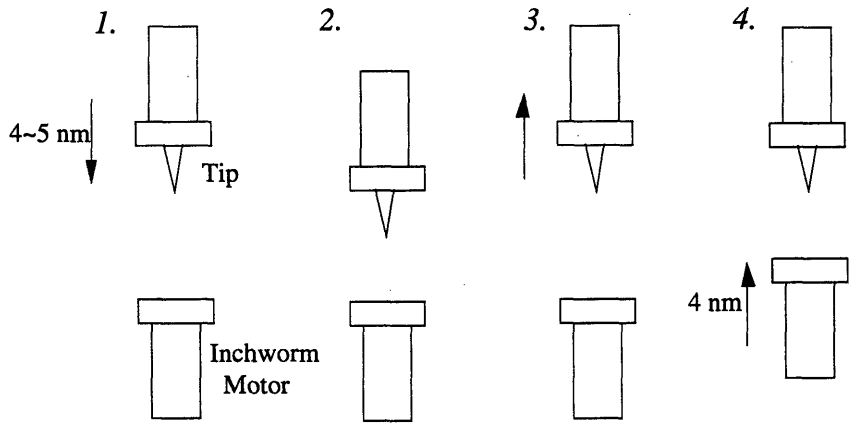


Figure 2.21: The Fine Positioning Mechanism

result in thermal expansions or plastic deformations. However the detail phenemena of the piezo material is still unknown.

The next mode, the fine positioning, uses both the inchworm motor and the piezo tube. This time, the gap is reduced with much slower rate but with higher resolution. To provide a good understanding of the positioning mechanism, a diagram is given in figure 2.21.

The first step of the fine positioning is to move the tip downward a little more than a single step size of the inchworm motor, usually around 4 nm - 5 nm. During this step, the downward motion is stopped whenever the tunneling current is detected. Note that the resolution of the overall operation can be determined in this stage of the fine positioning. Since the piezotube's resolution can be assumed as infinite, the resolution is determined from the resolution of the D/A which applies the control voltage. The resolution is found out to be 1.328 pm. The detailed information relevant to the D/A's characteristics is provided in section 5.3.

The next step, step 2 and step 3 in the figure, is to stop the tip and pull back the

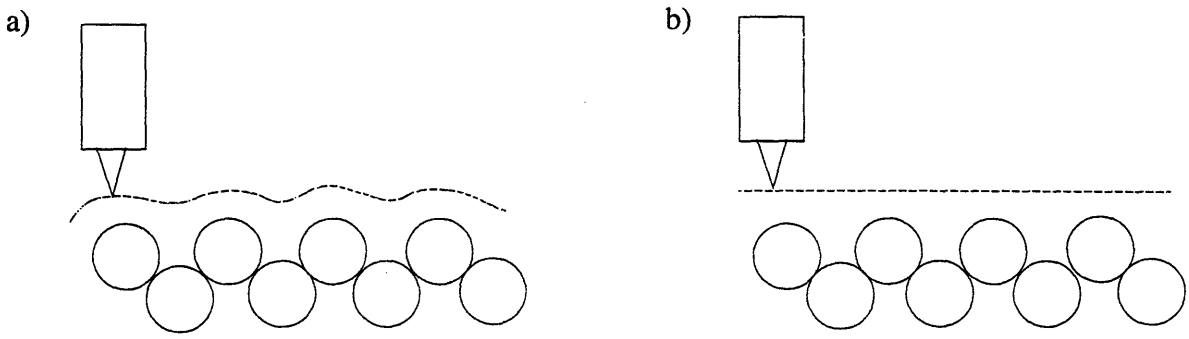


Figure 2.22: a) Constant Current Scanning b) Constant Height Scanning

tip to the original position knowing that the tunneling current is not detectable in this range. Then in the last step which is step 4 in the figure, the sample is moved up one step of the inchworm motor and the whole process is repeated.

2.7.3 Scanning

There are two common ways of scanning an atomic surface. One is called constant height scanning and the other is called constant current scanning. Both ways are adopted in our system for different purposes. In figure 2.22, the schematics for both modes are shown.

The constant height mode fixes the tip in z-direction throughout the scanning area. At the same time, the tunneling current generated is recorded. Later in the computer, the records are interpreted as atomic image data. The computer uses the logarithmic sheme of tunneling equation, discussed in section 2.7.1, to convert the tunneling current data to a gap distance value. The converted value represents the topology of the scanned surface and it is directly transfered to the display routine to be shown on the computer screen.

The constant height scanning allows a fast scanning speed which makes the system having less chance of being interrupted by outside noise at low frequencies such as thermal drifts. However, the scanning area of this scanning mode is limited up to only a few nanometers. If one scans over the limit, there should be a good chance of crashing the tip onto the sample. The fact that the data interpretation depends on the over simplified equation can be also one of the disadvantages of the constant height mode.

On the other hand, the constant current scanning isn't affected by such tunneling equation's nonlinearity. The tunneling current is controlled constant instead of the z-direction of the tip during the scanning. This time, the z-voltage applied to the piezo tube is taken as the atomic image data. A PID feedback control sheme is used to keep the tunneling current constant.

One of the advantages of the constant current scanning is that the scanning area is not limited as in the constant height mode. The constant current mode is used when the wide area needs to be scanned. However, overall scanning speed is slower than the speed of the constant height mode so that it can cause the low frequency noise disturbance. Also, the difficulty arises in the process of the control gain tuning when the noise is prevalent in the system. The control gain has to be set in such a way that the closed loop system well rejects the noises.

During scanning, the tip follows the path defined in the computer. There are numerous choices of path shapes. The ramp pattern is the one usually used because of its simpleness in software implementation while producing the satisfying result. However it is suspected that its sudden change of scanning direction causes the vibration of the piezoelectric tube and generates the high frequency component of the noise in the system. To solve the problem, triangular, circular, and even sinusoidal shapes are adopted. Such patterns should provide a continuous tip motion. For scanning a wide

area the ramp pattern is never used because of sudden back and forth motion could damage the scanned sample.

Chapter 3
System Modeling

3.1 Introduction

This chapter deals with the modeling of the system and, especially the PID feedback loop implemented in the constant current mode of scanning. The purpose of this work is to allow more in depth analysis of the overall system's behavior, the system's components' behavior and their contribution, and lastly the controller's performance.

The first section is devoted to the modeling of the piezoelectric tube dynamics. The second section discusses the building of the block diagram for the feedback loop. Here, each component in the loop is defined and identified by the frequency response approximation. The last section assesses the model derived.

3.2 Piezoelectric Tube Modeling

Modeling of the piezoelectric tube is important in analyzing the system's characteristics in that it is the only system's mechanical component whose dynamic behavior is usually difficult to define. Furthermore, the mechanical components usually have a slower response than the electrical components and therefore it has greater effect on the system's transient state behavior. This section discusses the fundamentals, the measurements, and the modeling of its dynamics.

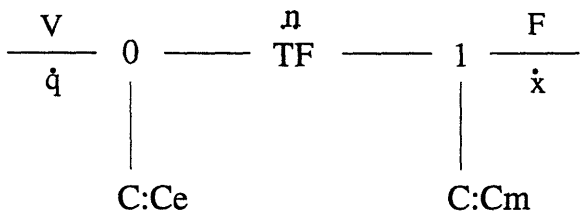


Figure 3.1: The Bond Graph Representation of Fundamental Piezoeletricity Relation

3.2.1 Background Physics

The piezoelectricity effect is understood to be an interaction between mechanical and electrical system. It is considered here that this relation is one in which two different variables in the system are linearly coupled. The linearly coupled system uniquely determines the free energy which can be expressed as,

$$E(x,q) = \frac{1}{2}a_{11}x^2 + a_{12}xq + \frac{1}{2}a_{22}x^2. \tag{3.1}$$

Here, x is the displacement and q is the charge by the definition of piezoelectricity. The exact differential of the above equation leads to the constitutive relation, given by two linear equations as below.

$$F = a_{11}x + a_{12}q \tag{3.2}$$

$$V = a_{12}x + a_{22}q \tag{3.3}$$

where F is force and V is voltage. For better understanding of this relationship, the bond graph can be used to represent the system. The bond graph is shown in figure 3.1. The graph shows a 2-port element, transformer, which converts the energy from the electrical to the mechanical domain and vice versa. It also has two 1-port capacitance at each domain connected by 0 junction and 1 junction respectively. These elements represent the capacitance and elasticity of the piezoelectric material. From this bond graph, the linearly coupled equations as we saw in the equations (3.2)

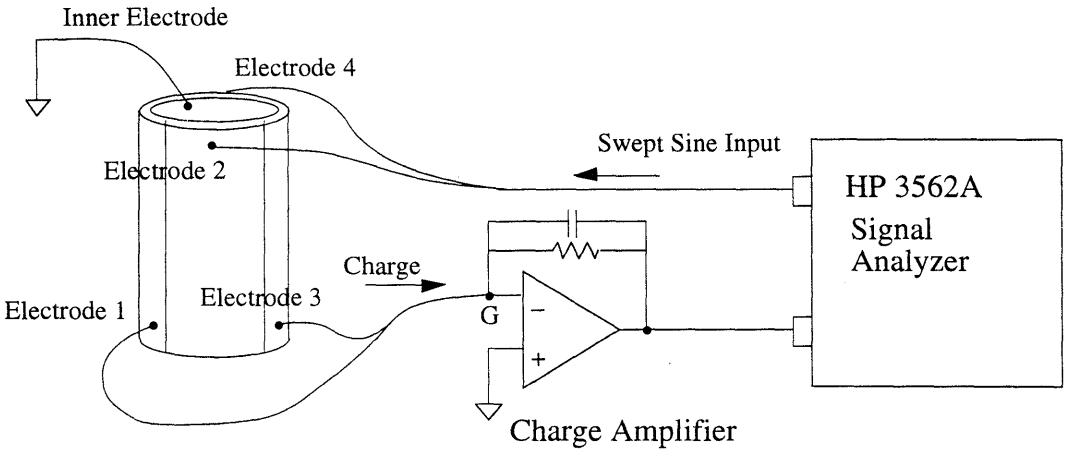


Figure 3.2: The Experiment Setup for the Frequency Response Measurement of the Piezoelectric Tube in Longitudinal Direction

and (3.3) can be derived.

$$F = (\frac{1}{C_m} + \frac{n^2}{C_e})x + (-\frac{n}{C_e})q \tag{3.4}$$

$$V = -\frac{n}{C_e}x + \frac{1}{C_e}q \tag{3.5}$$

3.2.2 Frequency Response Measurement

The frequency response of the piezoelectric tube was measured with a scheme similar to the one used in section 2.3.1 for the purpose of modeling its dynamics. The charge amplifer was used instead of the current-amplifier because of the known fact that the charge is linearly coupled with the displacement of the piezotube while the current is related to the speed with which the piezo tube is moving. The bond graph model provided in section 3.2.1 should help in understanding of this relationship.

Figure 3.2 shows the experiment setup for the frequency measurement. Note the connection of the piezo tube and the voltage source. The voltage source from the signal analyzer is directly connected to two electrodes of the piezo tube. During the

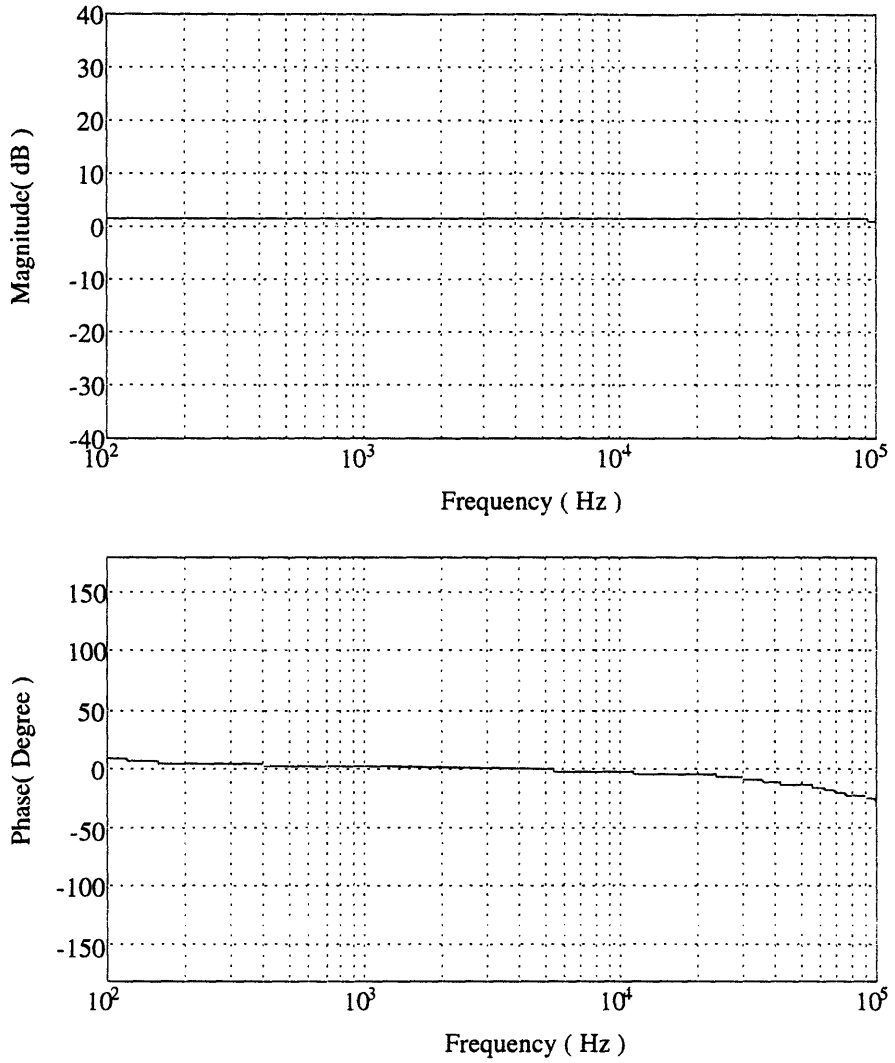


Figure 3.3: The Frequency Response of the Charge Amplifier (Experimental Result)

measurement process, the source increases the sine sweep to vary the frequency in which it vibrates the piezo tube. The output voltage after the charge amplifer is measured and the frequency response is plotted. We used an *HP 3562A Dynamic Signal Analyzer* for the frequency response measurement.

The frequency response of the charge amplifier was measured in order to remove its effect on the measured frequency response. Figure 3.3 shows the frequency response of the charge amplifier. The result shows that its gain stays constant in the measured

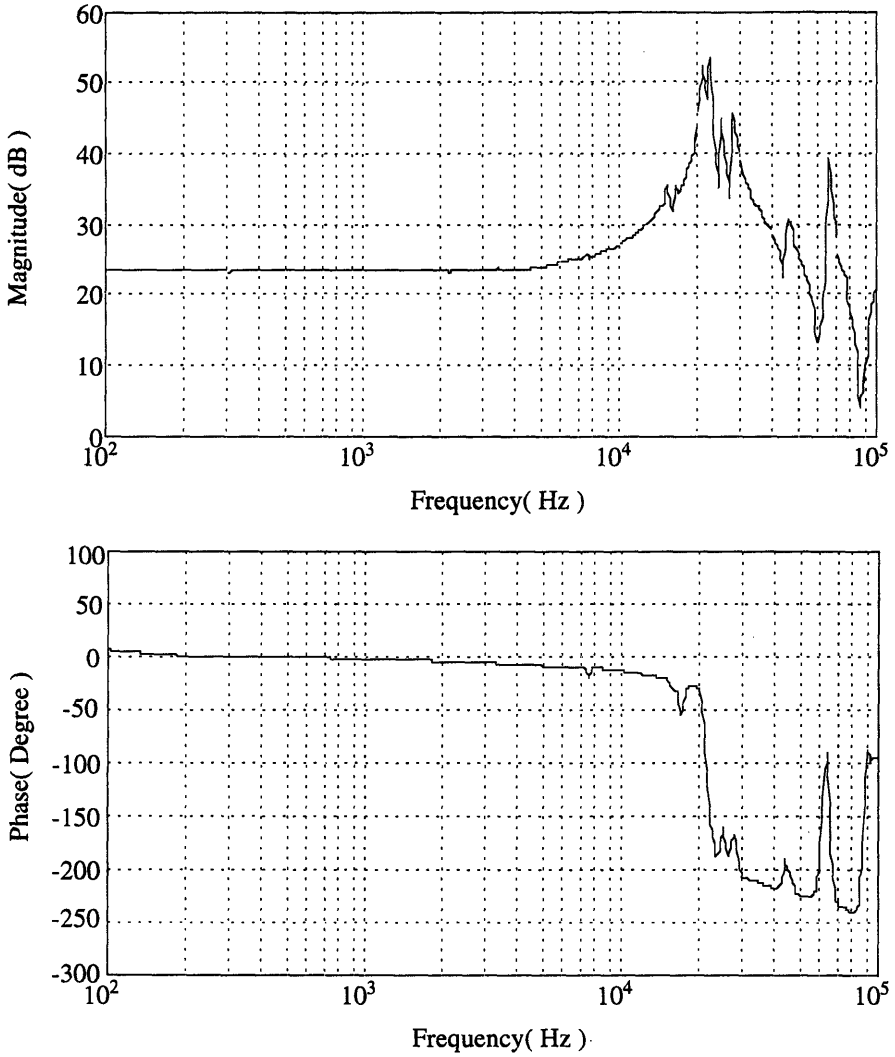


Figure 3.4: The Frequency Response of the Piezoelectric Tube in Longitudinal Direction (Experimental Result)

frequency which ranges from 100 Hz to 100 kHz. However the phase delay appears after 6 kHz and increases up to 30° at the maximum. Therefore its effect on the measurement should be considered in this frequency range of 6 kHz to 100 kHz.

Figure 3.4 shows the result of frequency response measurement of the piezoelectric tube from the experiment setup explained previously. The plot shows several vibrating modes in the high frequency region represented by small and large peaks.

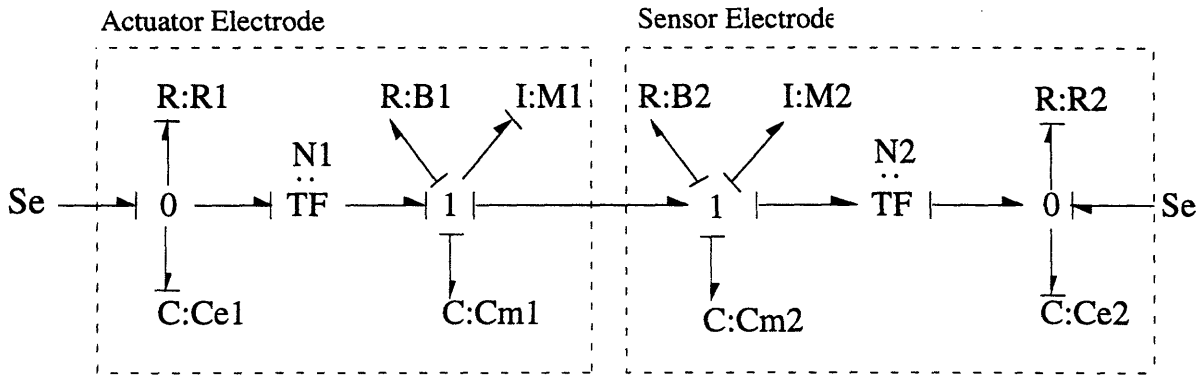


Figure 3.5: The Bond Graph Model of the Piezoelectric Tube Frequency Response Measurement

Although the vibration modes are frequent, the dominant peak located at around 20 kHz in the gain plot and the 180° shift at the same frequency in the phase plot imply that the system is 2nd order. Note how the phase at the high frequency region are distorted towards the bottom. This comes from the charge amplifier's phase delay.

3.2.3 Bond Graph Model

Figure 3.5 shows the bond graph model of the piezoelectric tube of which the frequency response is measured. The model comprises two main parts, the acutator electrodes to which the swept sine voltage is applied and the sensor electrodes from which the charge is measured. The applied voltage is represented as 1-port effort source shown on the left hand side of the graph. The voltage source is applied to the resistance and the capacitance connected in parallel represented as 0 junction. The 2-port element transformer then converts the energy from the electrical domain to the mechanical domain and thus affects the displacement of the piezo tube. In the mechanical domain the tube is assumed to have compliance, damping, and mass connected in parallel. The connection between the actuator part and the sensor part

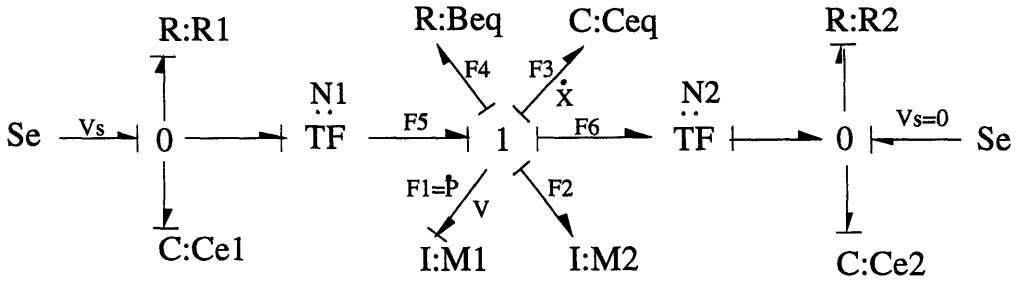


Figure 3.6: The Simplified Bond Graph Model

is assumed to be rigid. In the bond graph, it is represented as 1 junction to 1 junction connection .

From the sensor part of the electrodes, the variation of the charge is measured. The model includes 1-port elements(capacitance, resistance, and inertia) same as the actuator has. The 1-port effort source is inserted into the 0 junction of the electrical domain and its effort value is set to zero. This represents the connection between the sensor electrode and the virtual ground of the charge amplifer. In figure 3.2, this ground is specified as G. This causes the effect of the resistance and the capacitance of the sensor electrode to be removed and enables the charge to directly represent the displacement.

3.2.4 State Space Representation

The figure 3.6 is a simplified form of the previous bond graph model. The two adjacent 1 junctions are reduced to a 1 junction with the equivalent 1-port elements, B_{eq} and C_{eq} defined as,

$$B_{eq} = B_1 + B_2 \tag{3.6}$$

$$C_{eq} = \frac{1}{\frac{1}{C_{m1}} + \frac{1}{C_{m2}}} \tag{3.7}$$

Two state variables are chosen from the causality assignments shown in figure 3.6. Those are the momentum P of the actuator electrode and the displacement X of the piezoelectric tube that includes both electrodes as specified in the figure. The state equations are derived and are shown below,

$$\frac{dX}{dt} = V \tag{3.8}$$

$$\frac{dP}{dt} = F_1 \tag{3.9}$$

where V is the velocity of the piezoelectric tube and F_1 is the inertia force associated with the mass of the actuator electrode. The first set of derivatives can be further developed using constitutive relation of the inertia.

$$\frac{dX}{dt} = \frac{1}{M_1}P. \tag{3.10}$$

The M_1 is the mass of the actuator electrode. The second set can be also developed with the 1 junction relations,

$$\frac{dP}{dt} = -F_2 - F_3 - F_4 + F_5 - F_6. \tag{3.11}$$

Here, F_2, F_3, F_4, F_5, and F_6 represent the inertia force of the sensor electrode, the spring force, the damping force, the converted mechanical force from the voltage source, and the applied force to the sensor electrode respectively. Again by using the 1-port element relation for the inertia, the compliance and the damper defined as $F_2 = M_2\frac{dV}{dt}$, $F_3 = \frac{1}{C_{eq}}X$ and $F_4 = B_{eq}V$, and the transformer's constitutive relations defined as $F_5 = \frac{1}{N_1}V_s$, equation (3.9) can be written as,

$$\frac{dP}{dt} = -M_2\frac{dV}{dt} - B_{eq}V - \frac{1}{C_{eq}}X + \frac{1}{N_1}V_s. \tag{3.12}$$

F_6 becomes zero since it is connected to the 0 junction with the ground input at the other end. The constant N_1 in the above equation is the transformer modulus of the actuator electrode. Since we know that the 1 junction is a common flow junction, we can replace V's with $\frac{1}{M_1}P$ using the constitutive relation of inertia as shown below.

$$\frac{dP}{dt} = -\frac{M_2}{M_1}\frac{dP}{dt} - B_{eq}\frac{1}{M_1}P - \frac{1}{C_{eq}}X + \frac{1}{N_1}V_s \tag{3.13}$$

The above equation can be reorganized in the explicit form as,

$$\frac{dP}{dt} = -\frac{B_{eq}}{M_1 + M_2}P - \frac{M_1}{(M_1 + M_2)C_{eq}}X + \frac{M_1}{(M_1 + M_2)N_1}V_s \tag{3.14}$$

The state equations derived above can be put into matrix form,

$$\frac{d}{dt}\begin{pmatrix} X \\ P \end{pmatrix} = \begin{bmatrix} 0 & \frac{1}{M_1} \\ -\frac{M_1}{(M_1+M_2)C_{eq}} & -\frac{B_{eq}}{M_1+M2} \end{bmatrix}\begin{pmatrix} X \\ P \end{pmatrix} + \begin{bmatrix} 0 \\ \frac{M_1}{(M_1+M_2)N_1} \end{bmatrix} V_s \tag{3.15}$$

$$Q = \begin{bmatrix} \frac{1}{N_2} & 0 \end{bmatrix}\begin{pmatrix} X \\ P \end{pmatrix} \tag{3.16}$$

where the output Q is the charge being measured and constant N_2 is the transformer modulus of the sensor electrode. From the state equations, the transfer function with the voltage source as an input and the charge as an output is formulated. The resultant transfer function is,

$$\frac{Q}{V_s} = \frac{1}{(M_1 + M_2)N_1N_2} \times \frac{1}{s^2 + \frac{B_{eq}}{M_1+M_2}s + \frac{1}{(M_1+M_2)C_{eq}}}. \tag{3.17}$$

3.3 Construction of Block Diagram Model

Figure 3.7 shows the block diagram representing the PID feedback loop implemented in the constant current scanning mode. In the block diagram, the system's components of both the electrical and mechanical are defined. Here, each component in the block diagram is modeled by measuring of its frequency response and approximating with the transfer function of an appropriate order.

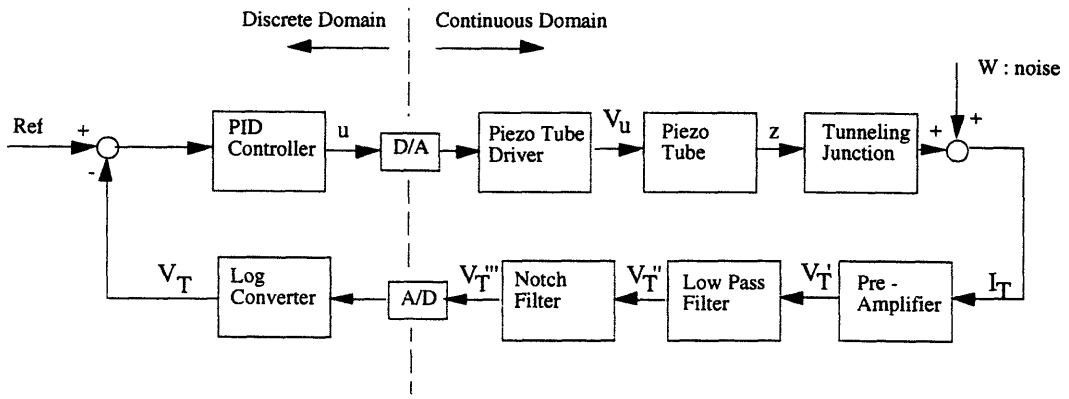


Figure 3.7: The Block Diagram of the PID Feedback Loop

3.3.1 Piezoelectric Tube

Based on the 2nd order model derived in section 3.2.4, the transfer function of the piezo tube is found by curve fitting of the measured frequency response. In the block diagram model, the input in this case is u, the control voltage from the piezo tube driver circuit and the output is x which represents the z-direction displacement of the piezo tube. The resultant transfer function turned out to be.

$$G_{pt}(s) = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{3.18}$$

where, K=14.96, ω_n=8.42×10$^5 \frac{rad}{sec}$, and ζ=0.021. The coeffecients $K\omega_n^2$, $2\zeta\omega_n$, and ω_n^2 correspond to $\frac{1}{(M_1+M_2)N_1N_2}$, $\frac{B_{eq}}{M_1+M_2}$, and $\frac{1}{(M_1+M_2)C_{eq}}$ of equation (3.17) respectively. Both the frequency response of the approximated transfer function and the measured frequency response are plotted in figure 3.8. In the plot, the derived 2nd order model matches well with the highest peak in the low frequency region but leaves the high frequency region peaks not identified.

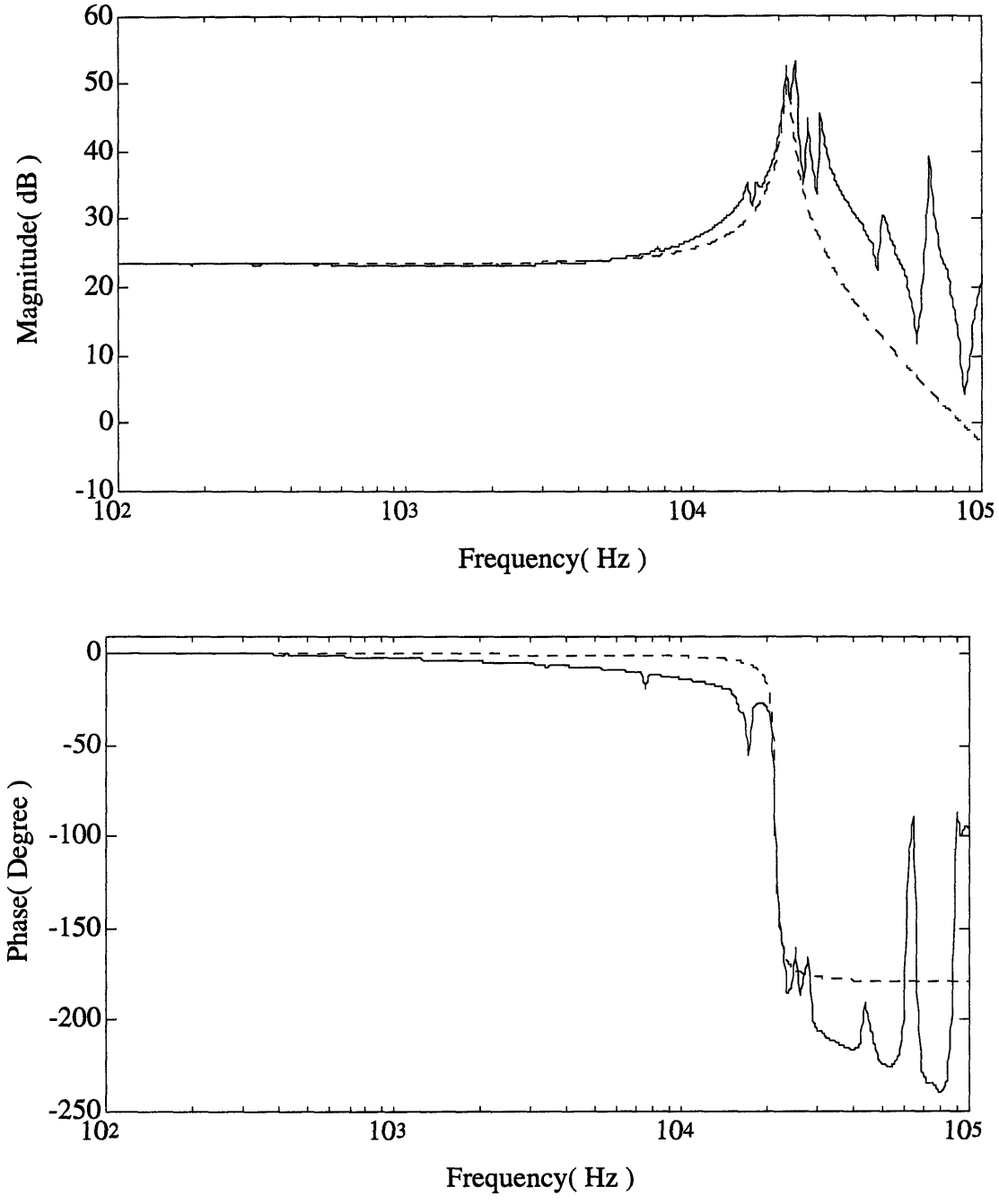


Figure 3.8: The Frequency Response of the Piezoelectric Tube in Longitudinal Direction(Curve Fitting Result) : Experiment ——, Curve Fit - - -

One should note that this simulation is based on the model in which the actuation of two electrodes during the PID feed back is assumed. In real case, the control voltage is applied to all four electrodes of the piezo tube for the longitudinal expansion or compression. This model assumption has been made because the corresponding experiment is easier to implement. Although the model does not exactly represent the real system, this should give some picture of the piezo tube's dynamics and its contribution to overall system's response.

3.3.2 Low-pass Filter

The frequency response of the low pass filter of figure 2.15 is approximated with a 6th order system. The approximated transfer function with V_T' as an input and V_T'' as an output resulted in following equation,

$$G_{lpf}(s) = \frac{63.1 \times \omega_n^2}{(T_1 s + 1)(T_2 s + 1)^3 (s^2 + 2\zeta\omega_n s + \omega_n^2)} \tag{3.19}$$

with ω_n=8800 $\frac{rad}{sec}$, ζ=0.3, T_1=$\frac{1}{2\Pi \times 1400}$ $\frac{sec}{rad}$, and T_2=$\frac{1}{2\Pi \times 2500}$ $\frac{sec}{rad}$. The bode plot of the above transfer function is drawn in figure 3.9. The model comprises two first order components with the cutoff frequency at 1.4 kHz and 2.5 kHz.

3.3.3 Notch Filter

The following transfer function of the notch filter is the curve fitting result of the frequency response previously shown in figure 2.15. The input of this component is V_T'' which is the output of the low pass filter. The output is specified as V_T''' in the block diagram.

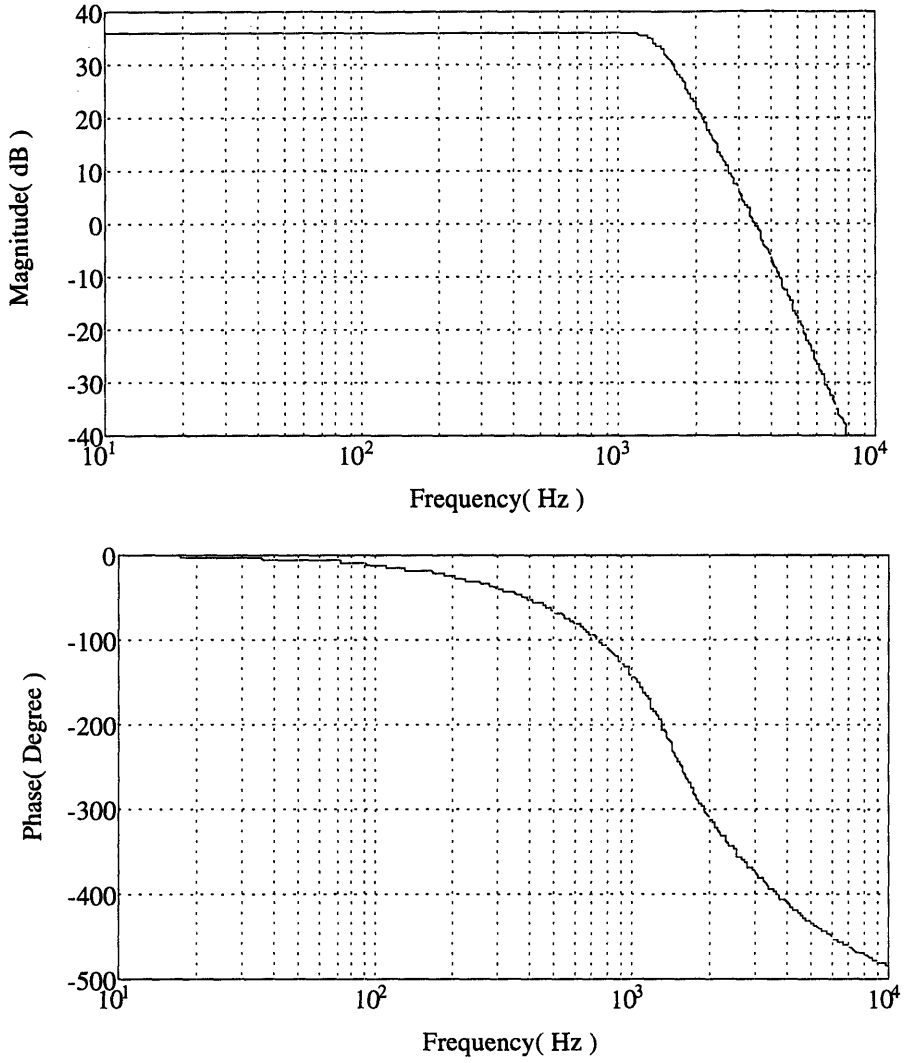


Figure 3.9: The Frequency Response of the Low-Pass Filter (Curve Fitting Result)

$$G_{nf}(s) = \frac{\omega_{n1}^2\omega_{n2}^2(s^2 + 2\zeta_3\omega_{n3}s + \omega_{n3}^2)^2}{\omega_{n3}^4(s^2 + 2\zeta_1\omega_{n1}s + \omega_{n1}^2)(s^2 + 2\zeta_2\omega_{n2}s + \omega_{n2}^2)} \tag{3.20}$$

where, ζ_1=0.23, ζ_2=0.23, ζ_3=0.09, ω_{n1}=251 $\frac{rad}{sec}$, ω_{n2}=361 $\frac{rad}{sec}$, and ω_{n3}=302 $\frac{rad}{sec}$. The figure 3.10 shows the frequency response of the transfer function found above.

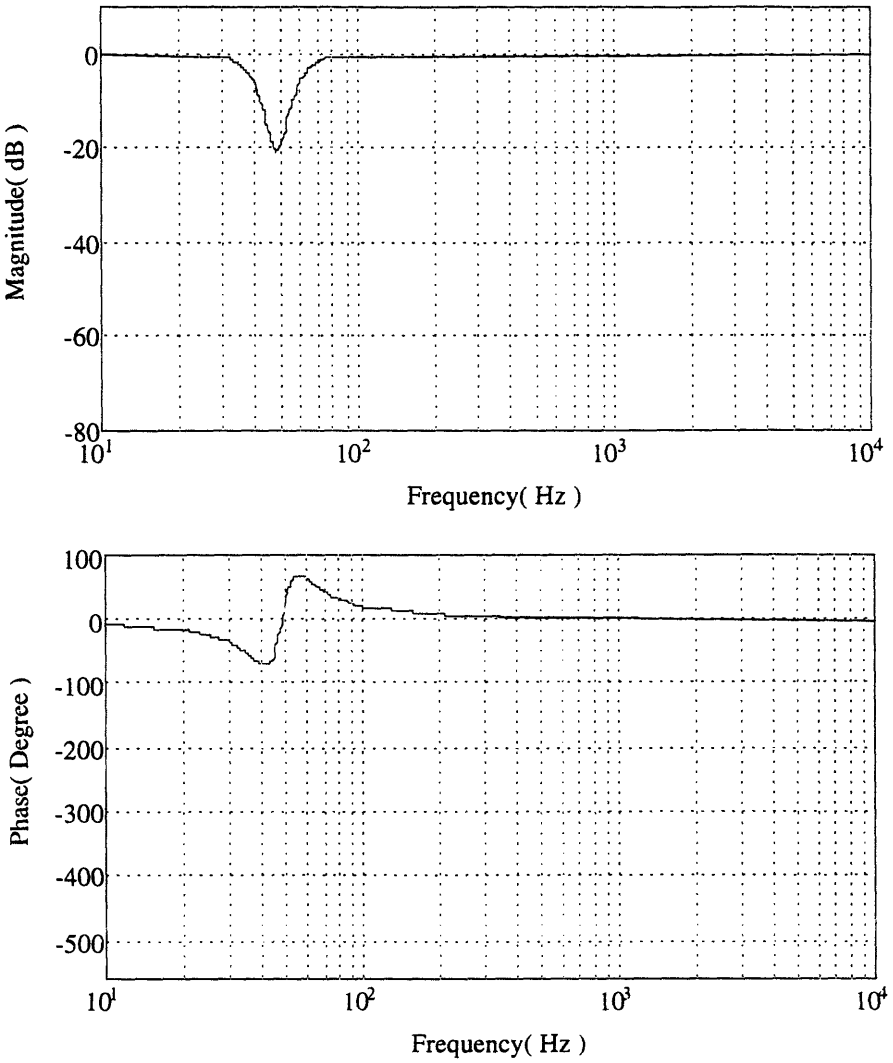


Figure 3.10: The Frequency Response of the Notch Filter (Curve Fitting Result)

3.3.4 Piezoelectric Driver

The frequency response plot of the piezoelectric driver circuit shown in figure 2.11 is approximated. The input here is the controller's command voltage u, and the output is V_u sent to the piezo tube. The following transfer function which comprises two first order system is the result of the curve fitting.

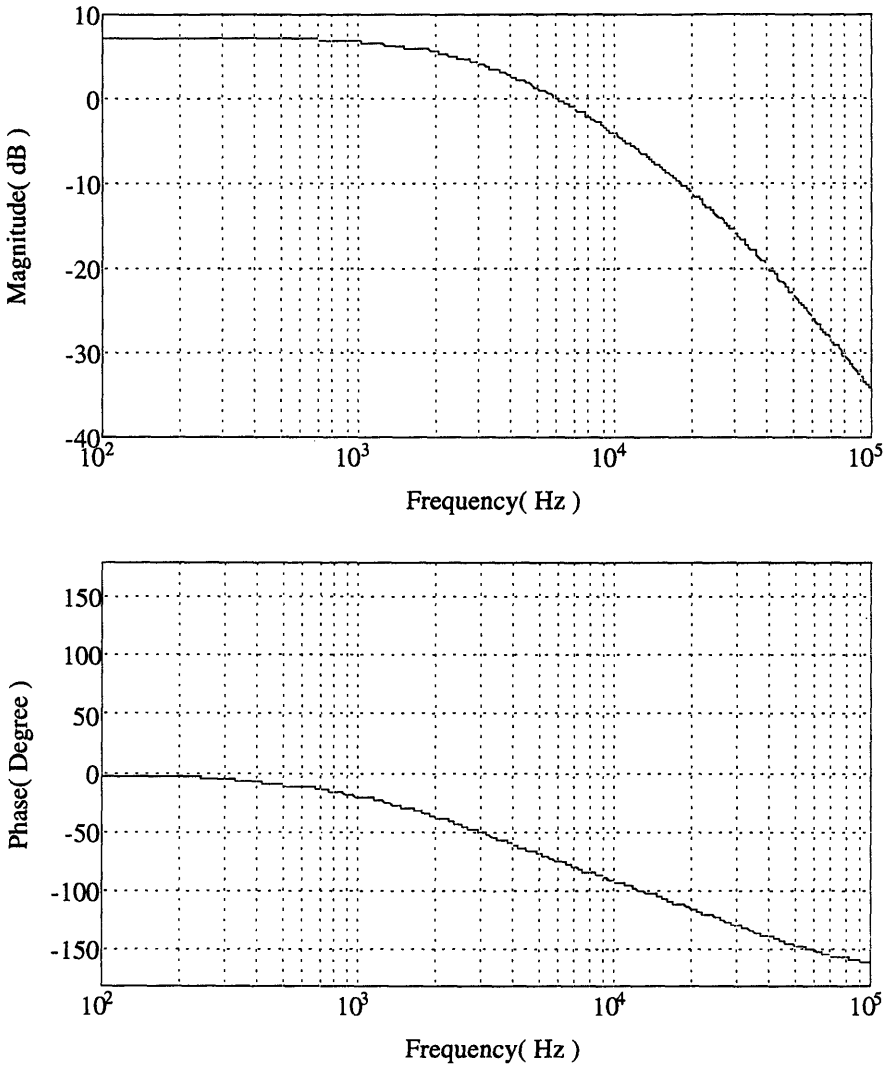


Figure 3.11: The Frequency Response of the Piezoelectric Tube Driver(Curve Fitting Result)

$$G_{pd}(s) = \frac{2.2387}{(T_1 s + 1)(T_2 s + 1)} \tag{3.21}$$

where, $T_1 = \frac{1}{2\Pi \times 3000}\ \frac{sec}{rad}$ and $T_2 = \frac{1}{2\Pi \times 30000}\ \frac{sec}{rad}$.

3.3.5 Pre-Amplifier

The pre-amplifier circuit diagram is shown in 2.12. Ideally the pre-amplifier should amplify the input voltage by 100 milion times with no time delay. In real world, there exists a stray capacitance and it causes some amount of the phase delay and the magnitude attenuation. However, this effect is negligible compared with the filters' and thus ignored here.

3.3.6 Tunneling Junction and Digital Log Converter

The time delay is caused by the capacitance and resistance effect in the tunneling junction. This can be also ignored with the same reason stated above. The digital converter also causes the time delay due to its computation time. However, this also is negligible if compared with other components. The DC gain can be found from figure 2.19. Here, the relationship between the gap and the output voltage can be assumed to be linear in the measurement range from the fact that the tip varies the gap by relatively small distance during the feedback loop.

3.3.7 A/D Converter and D/A Converter

The A/D Converter has 14 bit resolution and the 10 V overall voltage range. The gain can be calcuated as,

$$\frac{2^{14}}{10} = 1638\frac{counts}{volts} \tag{3.22}$$

With the same manner, the D/A Converter's gain with its 16 bit resolution and 10 V output range can be found as,

$$\frac{10}{2^{16}} = 1.5259 \times 10^{-4}\frac{counts}{volts} \tag{3.23}$$

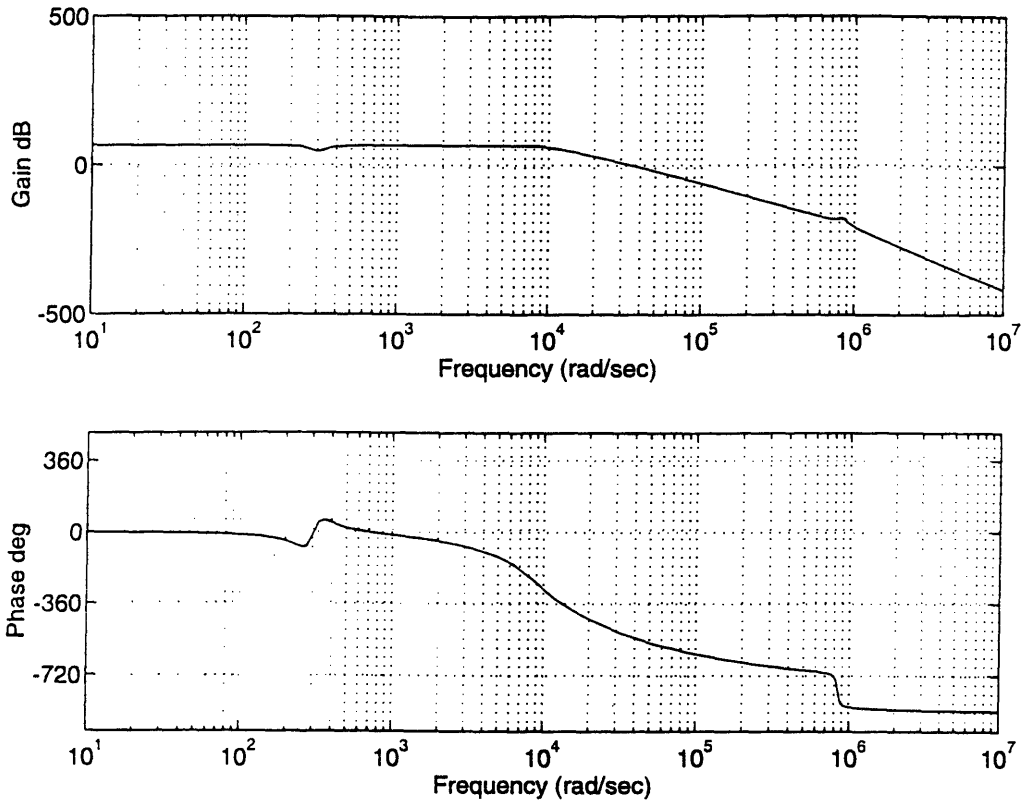


Figure 3.12: The Frequency Response of the Open Loop System (Simulation Result)

3.4 Discussion

In this section, the model derived from the previous section is characterized with its frequency response and is assessed by comparing the model's step responses and the experimental step responses.

3.4.1 Frequency Response

The frequency response of the system's loop transimission transfer function is shown in figure 3.12. The plot clearly shows the characteristics of the two filters, the notch at 60 Hz and the cut off freguency at 1 kHz where the downward slope is suddenly introduced. The plot also shows the small peak at the high frequency region which probably accounts for the piezo tube's dynamics. It can be ensured by the 180° phase shift at the same frequency in the phase plot of the frequency response. This

implies that the effect of the piezoelectric tube dynamics is small compared to other components.

3.4.2 Step Response

The step response of the simulated transfer function of the closed loop system of the block diagram 3.7 and the experiment results are plotted in figure 3.13 and 3.14 respectively. The step input here is applied to *Ref* which is the reference and the output is measured from V_T which represents the tunneling voltage after the log conversion. Both parameters are specified in the block diagram of figure 3.7. The experiments result is from the PID gain tuning program described in section 5.5. As one can see in the plots, two drawn figures are quite similar. Especially the two main features are noticeable. First one is the initial peak that rises at $\frac{1}{1000}$ sec. Both plot shows this behavior of step response. The other one is the response after the first rises. The response in that region shows typical underdamped response. Surpisingly the shape of the curve of it in both plots matches very well, specifically if we compare the number of oscillation, and the size of overshoot.

The experimental step response shows the high frequency oscillations which can not be seen in simulated step response. This should an effect of high frequency noise from the outside environment which we did not modeled in the simulation. The oscillation has the constant frequency whose average value was measured at 500 Hz. Note that the measured frequency was lower than the cutoff frequency of the low pass filter at 1 kHz.

To determine what the main contributers to the characteristics of the step response are, the open loop step responses for four major components in the block diagram are drawn. Each step response is drawn in a same span of time except the piezo tube's response. One can clearly see from the figure that the initial peak in the step

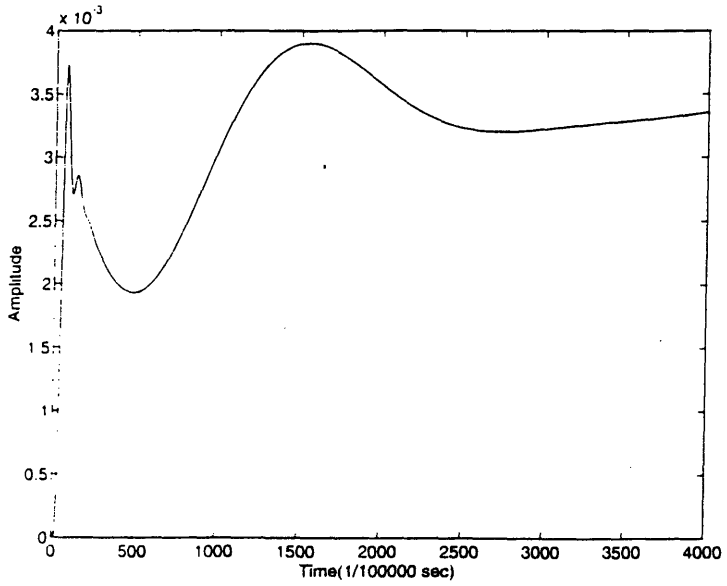


Figure 3.13: The Step Response of the Closed Loop System (Simulation Result)

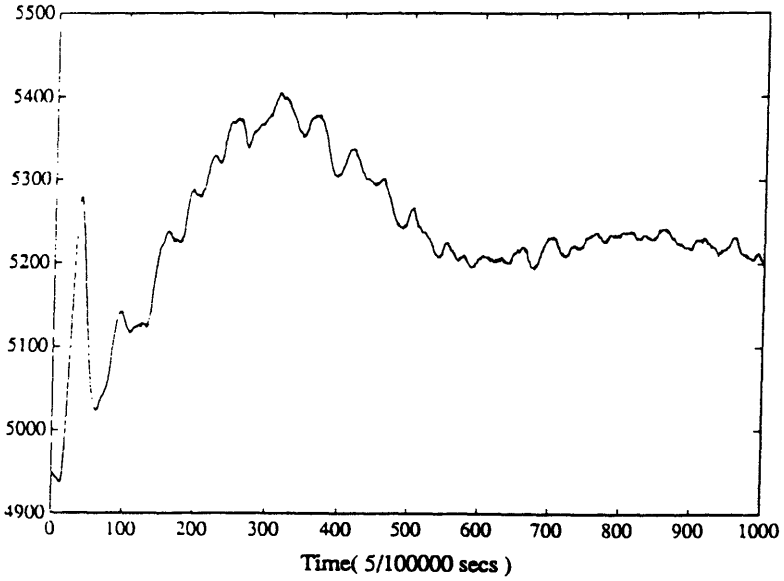


Figure 3.14: The Step Response of the Closed Loop System (Experimental Result)

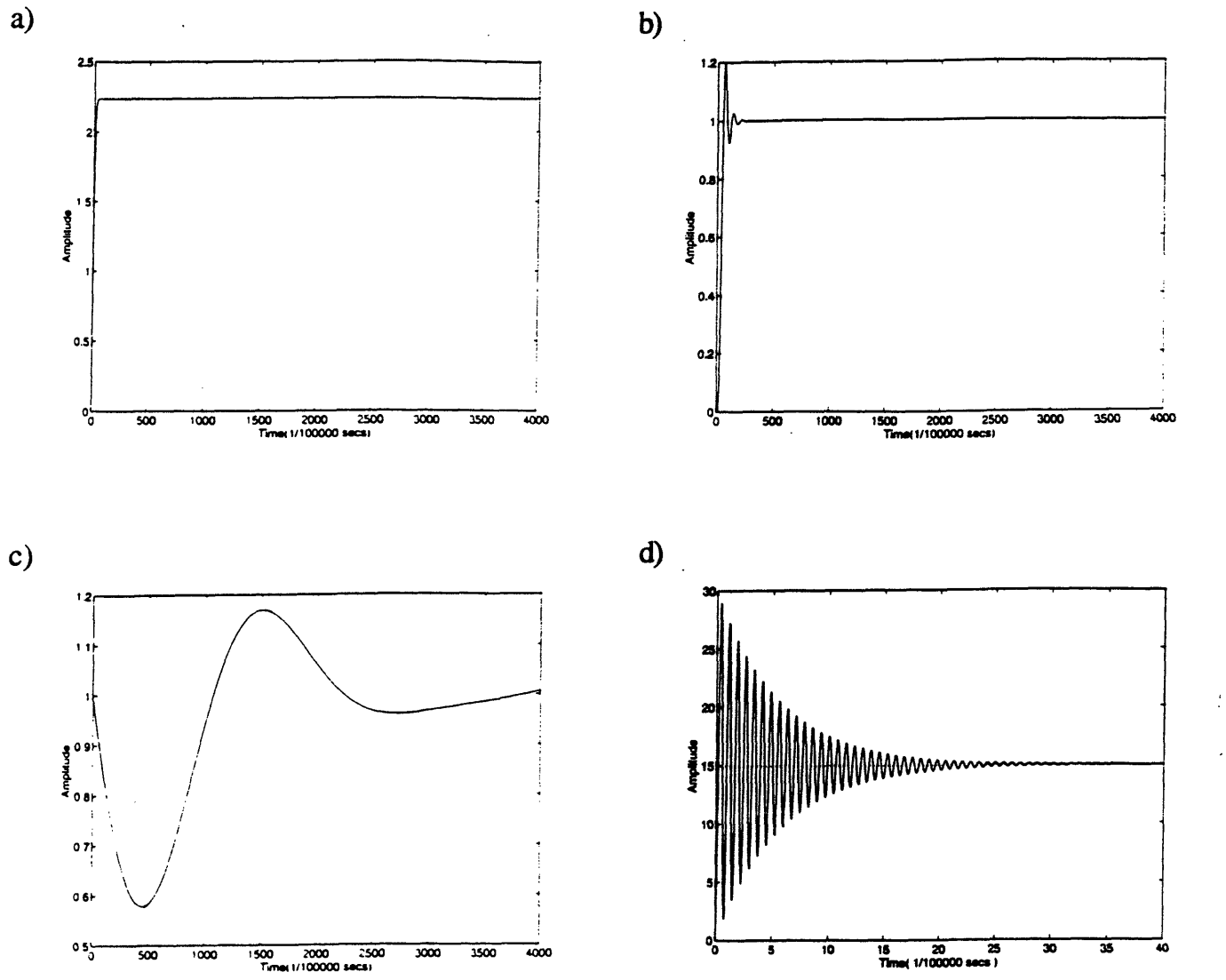


Figure 3.15: (a) The Step Response of the Piezoelectric Tube Driver (b) The Step Response of the Low Pass Filter (c) The Step Response of the Notch Filter (d) The Step Response of the Piezoelectric Tube

response matches with the low pass filter's response and the response that comes after matches with the notch filter's response. Since most of the transient response parameters are determined in the later part of the step response, this leads to the conclusion the effect of the notch filter's characteristics are the most dominant in the system's response.

Chapter 4

High Level Software for System Operation

4.1 Introduction

The scanning tunneling microscope control software is implemented in two levels, a high level and a low level. The high level software deals with the non-real-time operations such as the user interfaces and the data analysis. The software runs in the personal computer. The low level software operates the digital signal processor to control the instrument in real time.

This chapter deals with the high level software. This chapter first discusses the functions of the presently developed software. Only a brief description for each functions is given. One can find more detailed information in the code listing in the appendix B. The second part of this chapter addresses the environment provided for further development of the application software. The basic concept and the procedure of the development scheme are explained.

The C language is used in the development of this high-level software. The language is chosen because it is widely used and the language itself provides a good environment in developing such multi-task software.

4.1.1 Replacement of the STM Control Scheme

The initial scheme in controlling the scanning tunneling microscope has been changed since this thesis work started. One goal of the thesis was to enhance the sytsem's

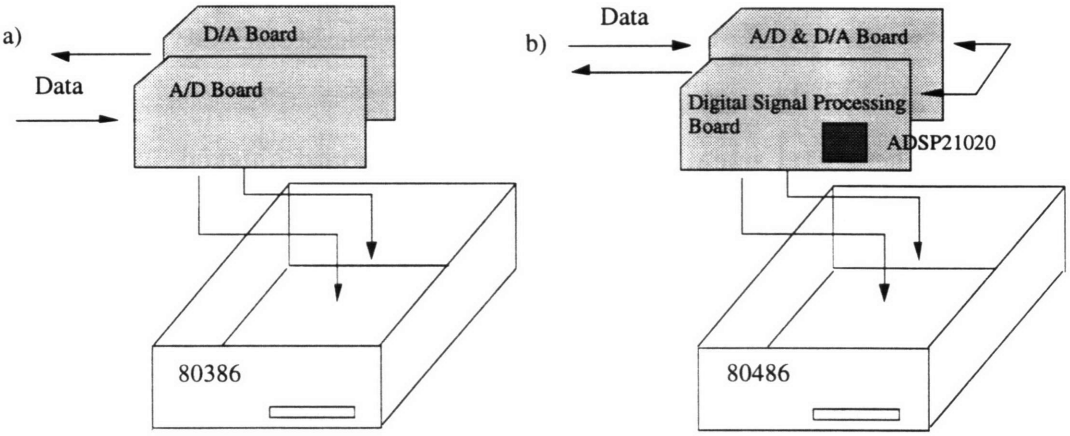


Figure 4.1: a) Initial Configuration b) Enhanced Configuration

performance by replacing the old sheme with a new improved one. This section discusses the changes that have been made to the system.

At the time the STM control software development was initiated on a 386 PC which was the only host computing system. The separate A/D board and the D/A board were attached to the PC via 16 bit ISA slot. For the A/D board, the DAT 2801 of Data Translation Inc. was used and for the D/A board, the DDA-06 of Metra Byte corp. was used. The A/D converter and D/A converter had a same 12 bit resolution. Also the A/D converter had the maximum sampling rate at less than 10 kHz. For the display, the 386 PC is hooked up with a CGA monitor.

The above configuration gave a satisfying result initially in scanning sample surfaces. However, as more advanced applications are being planned, the capability of the control system was limited because of the low conversion speed and the low computational power. With that in mind, a new control hardware system was developed. The two configurations of both old and new are shown in the figure 4.1.

The new configuration uses the two microprocessors, the ADSP 21020 digital signal processor of Analog Devices Inc. and the 80486 processor of Intel corp. Each processors are dedicated to their independent assignments. The digital signal processor is used for the real-time control part of the software, and the PC processor is assigned to the user interface part.

The digital signal processor resides in a board plugged into a PC expansion slot. The digital signal processor communicate with the user through this path. Its the other end is directly connected to the controlled unit. The separate board for the A/D and D/A conversion are also plugged into the PC next to the DSP board. This board is controlled by the DSP board through the bus connection. The data transfering speed between these boards is 60 nanosec per a 16 bit data. The board has the high speed A/D converters that has the 14 bit resolution and the maximum sampling rate at approximately 1.6 MHz. The board also has the D/A converter that has 16 bit resolution. Both boards are designed and manufactured by JRG Corp.

The new control system provides higher computing power and higher conversion speed than the previous system. The resolution of the D/A and A/D also became higher than before. (Note that high resolution of the D/A means high resolution of the piezotube's movement.)

4.2 Functionality Overview

There are three major modes in the main menu of STM control software. Those three modes are the constant height mode, constant current mode, and the data interpretation mode.

Constant Height Mode

The hardcopy of the menu for the constant height mode is provided in figure 4.2. The listed items in the menu are explained briefly below.

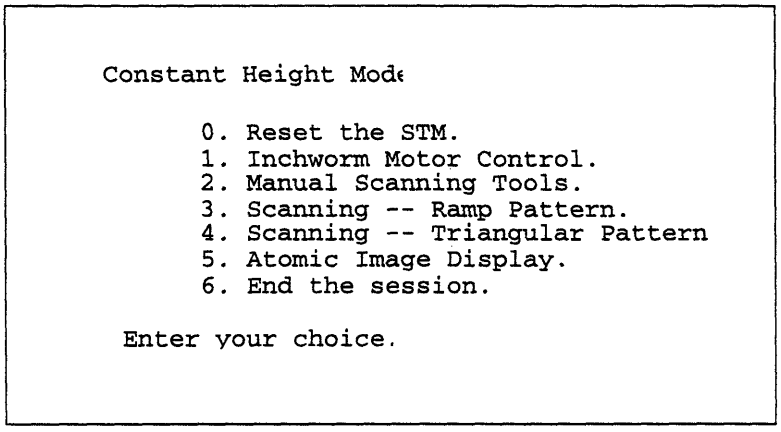


Figure 4.2: The Constant Height Mode

- *Reset the STM.* It resets the STM instrument by setting all the D/A channels equal to zero voltage except for the bias voltage channel. This command becomes useful when the D/A voltage abruptly changes to an unexpected value.

- *Inchworm Motor Control.* This function allows the user to control the motor. The speed, the direction, and the moving distance of the motor are the controllable parameters.

- *Manual Scanning Tools.* This function prints another menu for the manual scanning. The menu offers a step by step scanning. The user can perform the fine positioning, the initial height control, and the scanning of a sample separately.

- *Scanning - Ramp Pattern.* This module generates the ramp raster voltage during the scanning. The function scans the surface and retrieves the result to a specified array type variables. This should be used after the coarse positioning approach has been completed.

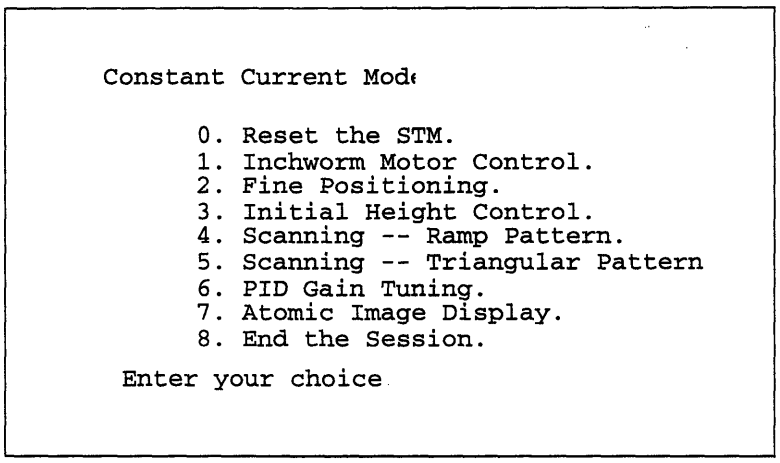


Figure 4.3: The Constant Current Mode

- *Scanning - Triangular Pattern.* This module is another version of the scanning program which generates the triangular raster voltage during the scanning process.

- *Atomic Image Display.* This function displays the scanned atom in grey scale on the screen. The atomic images shown in chpater 6 are the work done by this function.

Constant Current Mode

The menu of the constant current mode is shown in the figure 4.3. The mode enables the user to perform the following tasks. (Note that the explanations of the repeated items are omitted.)

- *Fine Positioning.* This function performs the fine positioning.

- *Initial Height Control.* This function initially controls the gap to a reference before the constant current scanning is performed. Such action is to prevent the

tip from taking a large step at very beginning of the constant current scanning.

- *Scanning – Ramp Pattern.* This function performs the constant current mode scanning with ramp pattern x-y raster input. The user can control the number of control loops, the scanning speed, the PID controller gains, and the reference voltage.

- *Scanning – Triangular Pattern.* This function performs the constant current mode scanning with a triangular pattern x-y raster voltage.

- *PID Gain Tuning.* This function allows the user to adjust the proportional, integral, derivative controller gain by displaying the step response of the closed loop system. The user can input the desired reference value, the proportional gain, integral gain, derivative gain. Two values, the data sampling rate and the control loop updating rate, are also inputted. The control loop updating rate is specified as multiples of the data sampling rate. This allows the user to view data between the PID loop samples. The figure 4.4 is one example of the step response plot it produces. In the plot, the inital voltage and the reference voltage are shown by the dotted line.

The data interpretation mode is self explanatory and thus further explanations are not given here. See appendix B.

4.2.1 Program Structure

The previously described functions related to the STM controls simply dump the assembled DSP code to the DSP board's program memory leaves the work of real-time controls and interactions with the instruments to the downloaded DSP code. Those functions take the same steps in order to accomplish the downloading task. The following summarizes the procedure it takes:

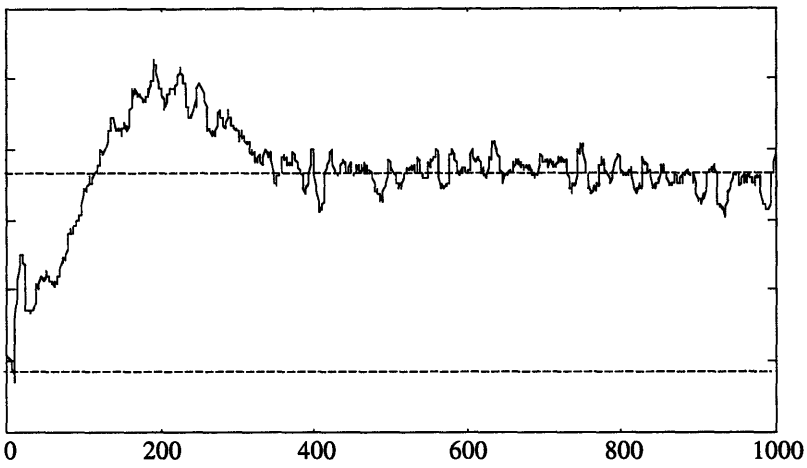


Figure 4.4: The Step Response Plot of PID Gain Tuning Function

1. Get user inputs.
2. Reset DSP.
3. Download the code.
4. Download the parameters.
5. Execute the program.
6. Check the status.
7. Upload the result and display.

The first part collects the user inputs. The one common input that the program requests is the sampling rate. The user inputs the sampling rate in Hz or kHz units. The program then translates the input value into the DSP compatible timer unit. In the next step, the program resets the DSP board to stop all the current operation

of DSP. This operation is to prevent the DSP from doing undesired operations when the new code is downloaded.

The downloading process comes next. The code and the user input parameter are downloaded into the memory location in the DSP board specified in the architecture file. The code should be compiled and linked with the architecture description file prior to this step.

Up to this point being complete, the program on the DSP board is now ready to be executed. The DSP starts executing the code when the run command is sent from PC. The run command is actually done by setting the reset flag of the control register to 1. Note that the flag has previously been pulled down to 0 by the reset command.

While the DSP board is running the code, the program checks the status of the current DSP operations through the status register. The status register is useful becuase it allows the host computer to check the DSP's condition without interrupting its current operation. The three conditions that are the most commonly checked through the status register. Those are,

1. Has DSP code started?

2. Is the sampling rate too fast?

3. has the DSP code finished?

Note that sequence of the status checking is important because all the status reports are sent through the same channel. When the digital signal processor reports the end of the required operation, the program uploads the results stored in the predetermined DSP memory location and presents the results to the user.

4.3 Application Software Development

The software is designed to provide a good basis for the development of advanced applications. The software offers two main features. First is the basic library subroutines that will be commonly used in the applications development. The library includes the basic STM control functions, the display routines, the DSP interface routines, and other program support routines. Second one is the proramming environment in which the developers can share the work they have done. The first section discusses this programming environment. The following two sections provides the description of the basic subroutines that should be useful in the software development.

4.3.1 Development Procedure

In using one's own written functions, the C language requires the prototype of the functions at the beginning of a program. For the library subroutines, one can just use the include statement to bring in the prototype. The prototypes of all the functions in the STM control software are listed in one main header file seperately from the main program file. If one have written the new functions, he should register the functions in this header file by writting the prototype in. The idea of function registration is also meant for a quick way to check the functions that others have written. One should provide a good descriptions when registering the new functions in the header file.

Next step is to include his sub menu in the provided main menu. Figure 4.5 is the main menu which will be shown on the screen. Each item represents the mode in which one's application is developed.

The last step is to add the content of one's application program to the main porgram. One should notify the compiler of its new program to be compiled as a

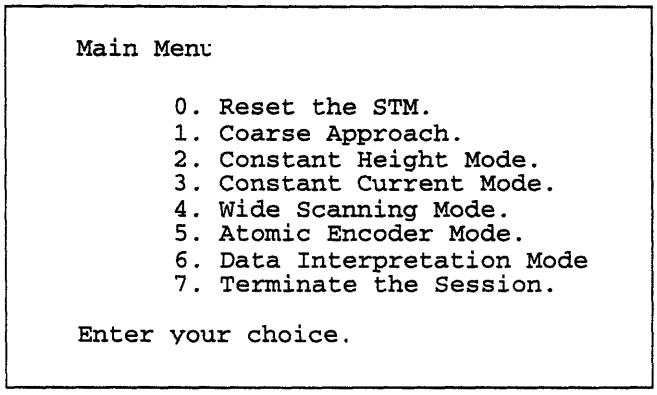


Figure 4.5: The Main Menu

part of the main program. Each commercially available C or C++ compilers have their own style of managing such projectwise programs. One should fully utilize those features.

The following is a summary of the procedure described above:

1. Register all the functions in the main header file.

2. Include the menu of the new module in the software's main menu.

3. Specify the new file name in the complier's option and compile.

4.3.2 Digital Signal Processor Interface

This section discusses the functions that the host PC uses to interface with the digital signal processing board. Only the design consideration of the software has given here. For the detail mechanism, one may refer to the user manual of this board. [20]

PC-DSP Handshaking

An effective communication method is critical in the real time operations of controlling scanning tunneling microscope, especially when the PC has to respond to the sudden malfunction of the controlled system. Such methods are machine dependent and the optimal way of communication has to be found by its software developer. The following is the discussion of the communication methods that can be implemented in our configuration.

It is the usual case that the PC sends the command to the DSP to perform a certain desired operation and the DSP sends the status of progress back to the PC. There are three ways of sending a command to the DSP from the PC that are possible with the current configuration. One way is to use the interrupt. The interrupt is triggered by writing to the bit 2 of the control register. This bit is connected to the IRQ0 pin of the ADSP 21020. When this pin is set low the provided interrupt routine in the DSP code is executed while the current operation of DSP is hold. Unfortunately, other IRQ pins are not connected and there are no means to trigger them from PC. Only one DSP operation can be done using this method.

Another way of sending a command to the DSP is to directly download the program which executes the desired operation. There exists a delay time between the DSP programs while the new program is being downloaded. However, this method gives many choices of what the DSP can do within one execution period. Most of the DSP code written in this thesis is designed in this way. When this method is used, the parameters that have to be shared between the DSP programs are transferred through the PC. The PC uploads all the parameters that include the last updated informations of the previous operation and downloads those parameters back with the new DSP code for next operation. This scheme is used, in the STM control software

case, to transfer the last position of the tip to the next module.

The other way of interacting with the DSP is to write a command to a specific memory location in the DSP while the DSP is in the loop of reading the same location. When a certain number is written to such location, the DSP can be dispatched to the corresponding part of the code.

The way of communicating to a PC from a DSP is well supported by the two registers located in the DSP board. The main registers are the status register and the timing register. 16 bit status register can be read by the PC without interrupting the DSP's operation. This feature prevents the DSP's operations from being interrupted while PC is in the loop of checking the its status. This is also hold for the 8 bit timing register.

One can think of the real time scanning scheme of using this feature of DSP board. Using the status register as the data transfering channel and the timing register as the status channel that reports the availability of the data, the high speed data transfer from DSP to PC can be achieved.[20] The DSP takes only 30 ns to hand over data to the PC, which enables the scanned data to be transferred at even a real-time video rate. However,the imaging rate will be limited by the time it takes for PC to read the data and display the image on the screen.

DSP Code Download

The download function reads the compiled program and writes the results to a DSP program memory. The subroutine reads only the .stk formatted code. The .stk file is produced by the software development tool, the PROM splitter. One should, after he compiled the original assembly file using the assembler and the linker to produce the .exe file, input this .exe file as a command argument to the PROM splitter with the proper options. [19]

DSP Memory Access

The functions to access a desired location in DSP memory are provided. The modules are provided for the different data types. Those are *int, long* and *float* type. The data should be divided into three 16 bit segments, high, middle, and low to be assigned to the corresponding registers in the DSP board. For both the fixed point format and the floating point format are compatible between the PC and the DSP and therefore further data format changes are not neccessary.

DSP Control

The two functions are provided in controlling the digital signal processor. One is for resetting the chip and the other is for initiating the chip. The chip reset can be done by toggling the bit 0 of the control register and vice versa for the chip initiation.

4.3.3 Grey Scale Display

The acquired atomic image can be represented in many ways. One can show the atoms graphically in a 3-dimensional graphics, the grey scale, and other 2-dimensional representations. The grey scale shows the atoms in 2-dimensional plane with the atom's height shown by the brightness of the point.

The grey scale is widely used because of its simpleness in algorithms. In figure 4.6, the scheme of implementing a grey scale into this software is shown. The 64 by 64 plane in the figure represents the scanned atom surface and the black dots show the data points. Each data point has the information of the atom's height which is then converted to a grey color according to its size. Note that the maximum number of colors that can be assigned to each data point is limited because the VGA monitor allows no more than 16 colors.

Because the 64 by 64 resolution didn't provide such good quality display, the

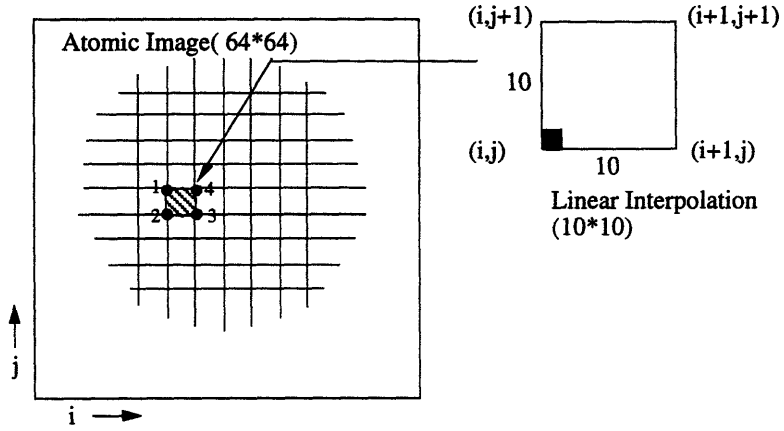


Figure 4.6: The Grey Scale Scheme

resolution was increased by the linear interpolation. The linear interpolation is done for the 4 data points from each of the unit square in the data plane.

Chapter 5

Low Level Software for Control and Identification

5.1 Introduction

The STM control software handles all of the machine's control functions such as the PID feedback, XY raster generations, coarse approach, and data acquisitions. Furthermore, the software provides the user interface and the display of the data.

As mentioned before, the architecture that are base on the DSP divides the task in which the DSP handles the instrument control functions and the PC handles the user interface and the display. This chapter discusses the control functions performed by DSP. (The expression, low level software, is used to describe the real-time control software implemented on the DSP board.) This software is written in assembly language supported by the ADSP21020 processor. The chapter is intended for giving only the design considerations. For the details of the software, one should refer to the program sources listed in appendix B and the ADSP21000 family user manual.

The first section provides a brief review of the software. Some of the general aspects of the software described in the section include the basic structure of the program, the timing considerations, and other relevant points to software design. The next four sections discuss each module of the STM control software. The modules described in these sections include coarse approach, PID feedback, PID gain tuning, and scanning. The underlying algorithms and the design considerations of each module are described in depth.

5.2 Software Overview

Timing Considerations

Most real-time control functions require consistent and predictable timing for proper operations. Likewise, the STM control software needs a known data sampling rate to predict and control the effect of thermal drift, control feedback characteristics, scanner's dynamics, and ambient noise.

The built-in timer in the DSP is used for consistent timing of the control functions. The minimum timer period that can be set by the user is 30 ns. Every time the timer expires, the DSP, for example in the case of constant current scanning, advances the x and y raster, reads the A/D, performs the PID feedback, stores the atom image data and waits for the next interrupt. The timer period should be determined by the instrunction's length or the A/D conversion speed to be as short as possible because the short timer period allows the faster scans and more number of data averaging for a slow scans. It is the usual case that A/D conversion time sets the lower boundary of the minimum possible timer period because most of the control code is less than 50 instructions.

Program's Structure

Figure 5.1 and figure 5.2 show two main parts of DSP code's structure in the STM control software. First, the DSP program reports its initiation of the program execution to the host computer. Then it initializes the DSP environment and the analog I/O environment. The initialization of the DSP includes the interrupt setups and the assignment of initial value for the variables and the registers. The analog I/O initialization includes the assignment of the number of channels, the reset of the analog board, and the first conversion of the A/D. The A/D conversion is done to provide

the data available in the FIFO before the first loop of the interrupt service routine reads the A/D value.

In the next step, configuration and initialization of the DSP timer is performed. Note that a multiple of the DSP's clock speed (33 M Hz) is used for the units instead of other time units such as the second and Hertz.[17] The last step of the main routine is to enable the timer and the interrupt and wait for the interrupt triggers. Every 3 μs the DSP timer triggers an interrupt which causes a jump to the interrupt service routine.

In figure 5.2, the interrupt service routine, during one timer period, is shown. Assuming that the A/D FIFO have the data available, the program reads the A/D FIFO and temporarily stores in a register. Likewise, it writes the new data to the D/A FIFO. Next the program checks the status of the analog board to ensure that the data it read from the FIFO is the one newly updated. Then, the program pulls the flag to start the next conversion. One should note that both the A/D and D/A are activated at the same time due to the architecture of the provided hardware.

The main loop calculations are done in the next step. Because the acquired A/D data is 14 bit wide, the data is sign-extended to 16 bit in order to perform the 16 bit fixed point calculations. (The DSP provides the instruction, *fext* which performs just that.) If floating point calculation is to be done on the A/D data, the data should be sign-extended to 16 bit first and then converted to floating point format using the *float* instruction. Also, because the actual reading of the A/D and the output of D/A are biased by a significant amount of voltage offsets, the software needs to compensate those offsets to get the true value.

At the end of the main loop computation, the final D/A data is converted from the 2's complement format to the binary offset format. This is because the D/A only supports the binary offset numbers. The numeric format is converted from a 2's

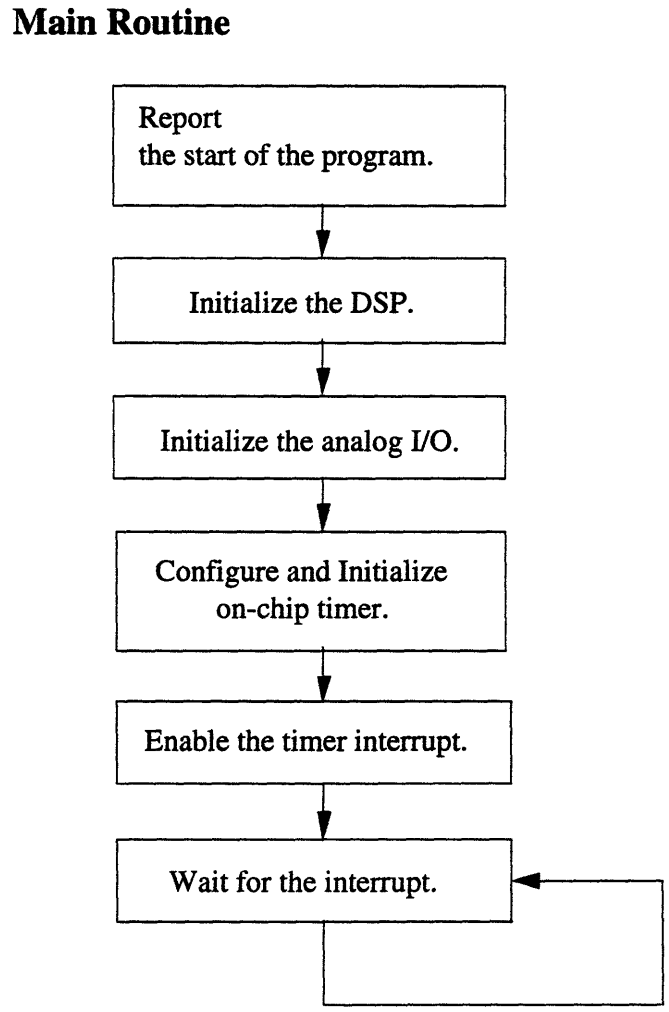


Figure 5.1: The Structure of Main Routine

complement to a binary offset by simply inverting the most significant bit of the 16 bit data.

When the main loop counter expires, the program disables the timer and the interrupt, empties the FIFO for the next program to be executed and reports to the host PC that it has finished the operation.

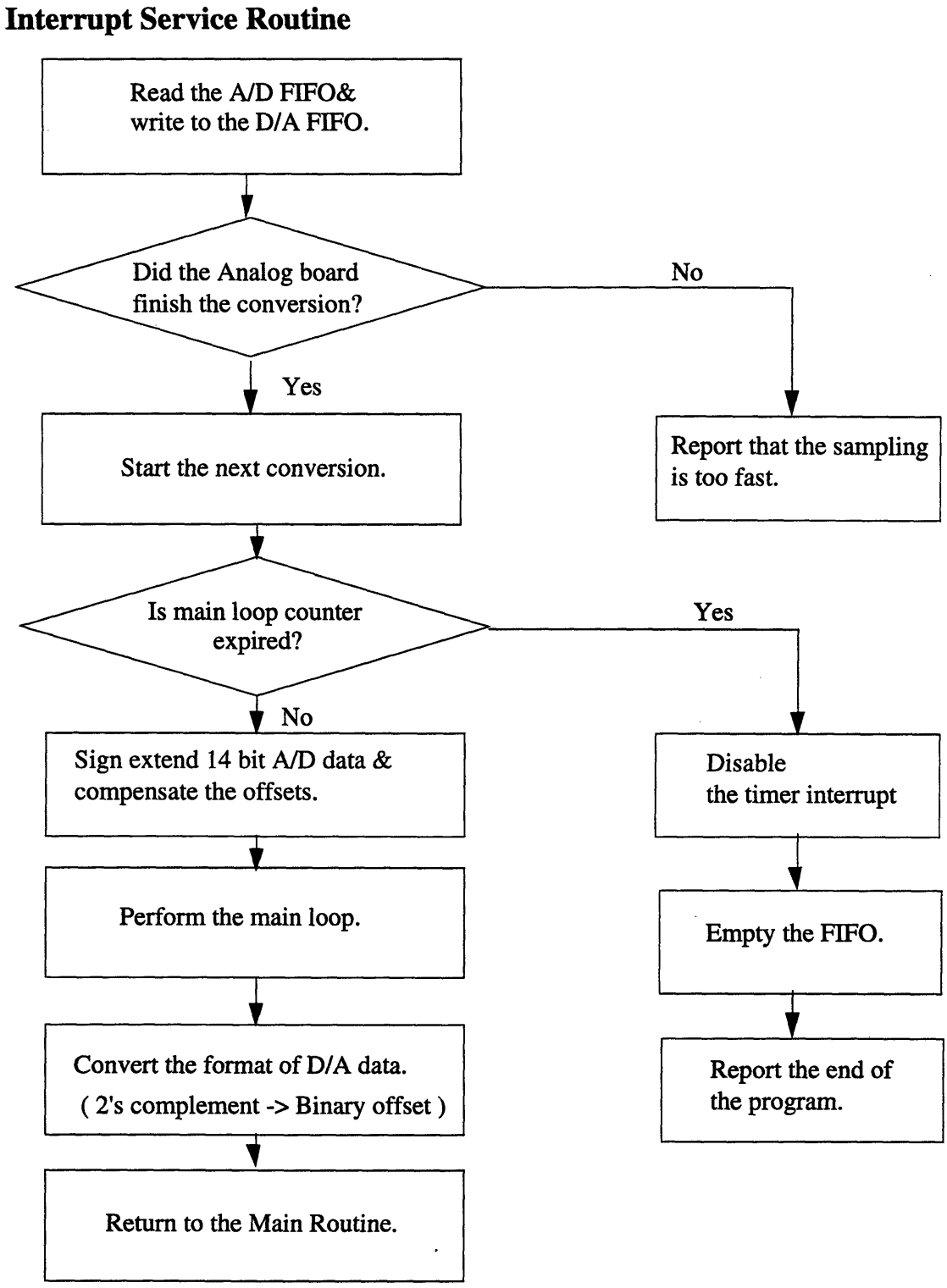


Figure 5.2: The Structure of Interrupt Service Routine

The Number of Channels

Each module may need a different number of channels. For example, the fine positioning module needs only one A/D channel for the measurement of the tunneling current and two D/A channels for the z-direction control of the piezo tube and the bias voltage. On the other hand, the scanning module needs additional two more D/A channels for the x and y direction control of the piezo tube. The DSP sets the number of channels by writing to 0x40000002 of the data memory.(The memory mapped port to the number of channel register in the analog board.)[20] Each module defines only the number of channels it needs because the redundant channels reduce the maximum possible sampling rate.

The order of writing the values to the D/A FIFO becomes important when more than one channel are used. The first data written to the FIFO goes to the first channel. When the flag 2 in ASTAT register of DSP is triggered, the D/A conversion for all the channels are performed at the same time. No channel can perform conversion at a different time. Also note that for each number of channels (from #1 to #4), certain channels are selected by default. For example, if the number of channels is set to one, the user can only use channel #1, not channel #2, channel #3, or channel #4. In this case, one should use channel #1 for most frequently used output.

5.3 Initial Approach

5.3.1 Coarse Positioning

In a coarse positioning, the DSP uses the inchworm motor to reduce the gap. To control the inchworm motor, the DSP writes a command to its 8 bit digital output port which is then connected to the digital port of the inchworm motor controller.

The controllable inchworm motor parameters are the direction of the motor's

motion(either upward or downward), the speed, and the travel distance. The speed and the travel distance of the motor is controlled through the clock signal input to the controller. The speed is determined by the ring number of this clock while the travel distance is determined by the number of the same clock signal. The inchworm motor moves 4 nm on the rising edge of this clock signal. The DSP simply writes 0 and 1 alternately to the digital output port to generate the 5 V amplitude clock signal.

During the coarse positioning, the software moves the motor at a average constant speed of 2 μm/s. The software samples the data at a slightly faster rate than the motor speed to respond immediately to a detected tunneling current. In the software point of view, this rate of data sampling is used as the fundamental timer clock and every multiples of samples, the motor is moved by one step. When the software detects the tunneling current, it jumps to the subroutine which stops the upward motion of the inchworm motor and move it again in the opposite direction by 20 steps with 100 kHz clock frequency. This reverse motion, as mentioned before, is to prevent the tip from crashing onto the sample.

5.3.2 Fine Positioning

In the fine positioning mode, the DSP uses both the inchworm motor and the piezoelectric tube. The DSP uses two D/A channels to send the bias voltage and the z-direction control voltage of the piezo tube. It also uses one A/D channel to measure the tunneling current.

The usual sampling rate is around 500 Hz. With that sampling rate, the motor moves upward at a speed of 1.5 nm/sec with the resolution of 0.1 $\AA$. (The speed and the resolution are calculated based on the z-direction sensitivity derived in section 2.3.1.) Figure 5.3 shows the command signals for the inchworm motor and the piezo

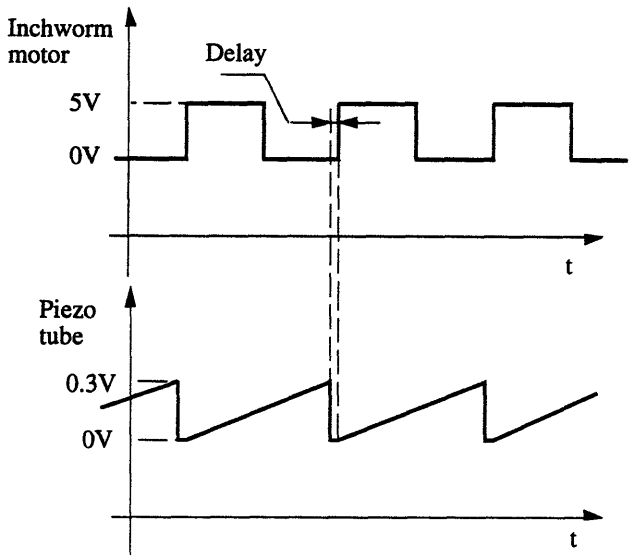


Figure 5.3: The Output Signal to the Inchworm Motor and the Piezoelectric Tube

tube that the DSP produces. The DSP produces the square wave signal to move the inchworm motor and a saw tooth shaped signal to control the piezo tube. The piezo tube is move by 4.8 nm while the inchworm motor moves one step. Note that the redundant 0.8 nm over the specified inchworm motor's resolution of 4 nm is a safety factor. Also, in the software, a delay was introduced between the inchworm motor's signal and the piezo tube's control signal. This is to avoid a crash as the inchworm motor moves upward with the tip at the lowest position.

5.4 PID Feedback

The PID feedback is used to control the tunneling current to a constant level that the user specifies. The computing power of the DSP enabled us to directly emulate the analog feedback and provides an easier means of changing feedback parameters at the same time. Also, the DSP's floating point architecture simplified the development of the control algorithm especially when heavy computations were involved. By using

this advantages of the DSP, we could plan wide choice of control algorithms and other features such as spectroscopy and digital modulation.

The S-domain transfer function of the proportional, integral, and derivative controller is shown below.

$$D(s) = K_p \left[1 + \frac{1}{T_I s} + T_D s\right] \tag{5.1}$$

Since we are using the digital controller and have to consider the effect of sample and hold operations, the Z-domain transfer function is derived as shown in the equation (5.2) below. The equation uses the backward Euler scheme in approximating derivatives and integrals.

$$D(z) = K_p \left[1 + \frac{Tz}{T_I(z-1)} + \frac{T_D(z-1)}{Tz}\right] \tag{5.2}$$

where T is the sampling rate, T_I is the integral time, T_D is the derivative time, and the K_p is the proportional gain. The difference equation of command output u(k) and the error signal e(k) is derived to be directly implemented in the software.

$$u(k) = A \times e(k) + B \times e(k-1) + S_I(k-1) \tag{5.3}$$

$$S_I(k) = C \times e(k) + S_I(k-1) \tag{5.4}$$

where,

$$A = K_p(1 + \frac{T}{T_I} + \frac{T_D}{T}) \tag{5.5}$$

$$B = -\frac{K_p T_D}{T} \tag{5.6}$$

$$C = \frac{K_p T}{T_I} \tag{5.7}$$

In the software, the A, B, and C coefficients are calculated and downloaded from the PC prior to the DSP code execution in order to reduce the computation time.

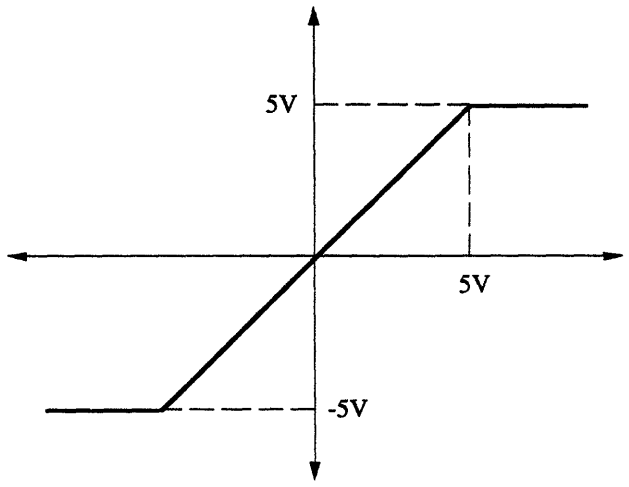


Figure 5.4: The Saturation

Figure 5.4 shows the software implemented saturation function. The x-axis represents the input voltage and the y-axis represents the output voltage. If a voltage of more than 5 V is written to D/A, a saturated voltage of 5 V is assigned instead. This operation is necessary in that the 16 bit D/A FIFO is written from the 32 MSB of the 40 bit register. If the data is, for example, 0x10000, then the D/A FIFO will be written with value of 0x0. This causes the D/A output voltage to roll over between positive 5 V and negative 5 V which can bring disaster to sensitive instruments.

Before computing the feedback parameters, the software converts the tunneling current to its log value. The software directly emulates the analog logarithmic amplifier previously employed in the STM instrument. See section 2.5 for the log conversion equation. Because of the execution time considerations, the look-up table scheme is used for the log conversion. The host PC computer calculates all the possible log value and downloads to the data memory(0x2000 - 0x3fff) prior to the PID feedback loop execution. During the feedback loop, the DSP generates the index to the look-up table residing in its memory. The following equation is used to generate the index.

$$Index = ADC Reading + 0x2000 \tag{5.8}$$

ADC Reading is the digital value to which A/D converts the input voltage. In this way, the address which corresponds to the log value can be uniquely determined from the input value. The software uses only 5 instructions which corresponds to 150 nano seconds to process the log conversion.

5.5 PID Gain Tuning

This module generates the step response curves for the purpose of tuning the PID controller gains. Such step response plot provided us a good visual indicator of closed loop response's rise time, noise level, settling time, overshoot, and other time domain characteristics as well. By adjusting the PID gains, we could shape the step response and find optimal values. This method provided a better means to adjust the PID gains than tuning PID gains by the inspection of the image quality.

Two types of sampling rates are inputted by the user before the program execution. The data sampling rate is the rate at which the DSP does the A/D conversions and the control loop sampling rate is the rate at which the DSP updates the control command output. This is to provide the user a means to view the data between the feedback samples.

Figures 5.5 to 5.8 are the step response plots generated by this module. Figure 5.5 shows the step response with the PID gains we usually used in the constant current mode scanning experiment. Figure 5.6 is the step response when the integral gain was increased. The plot shows the typical behavior of integral control, larger overshoot and more oscillations. Figure 5.7 is the result when the derivative gain was increased. The plot clearly shows one of the derivative controller's effect, the amplification of the noise. This frequency of the noise was measured at 450 Hz. Although its amplitude changes, the frequency was observed to be the same when different sampling rates and PID gains are used. The same phenomena happens when the sampling rate is

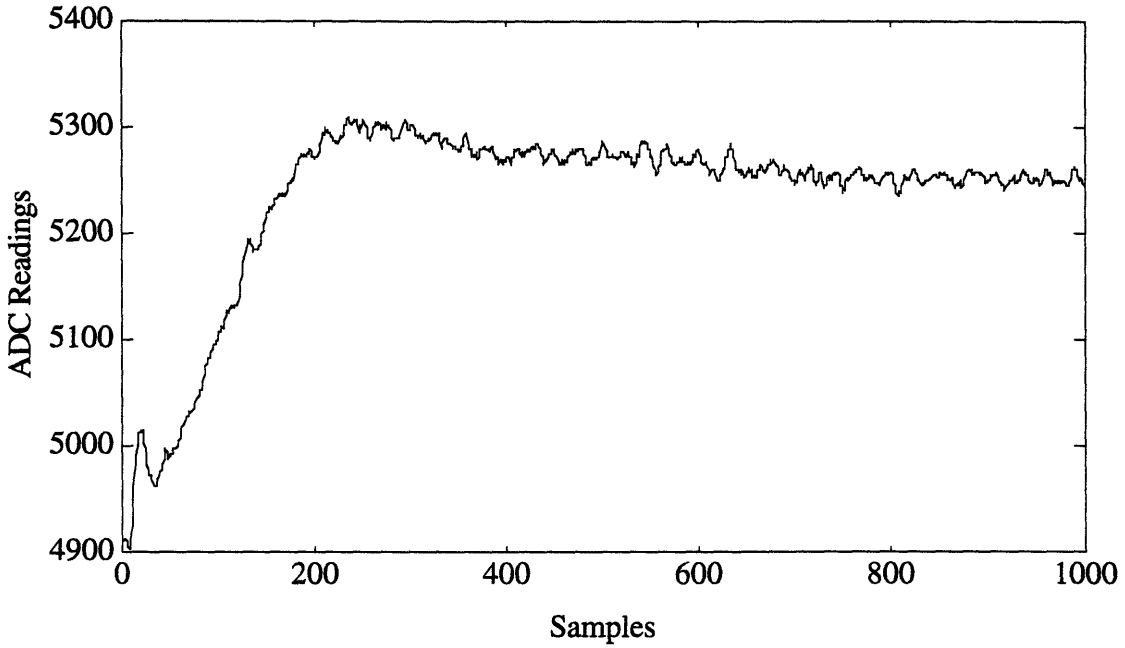


Figure 5.5: Fs=10 kHz, K=0.00065, Ti=0.065, Td=0.016 (unit) 100 $\frac{microsec}{sample}$

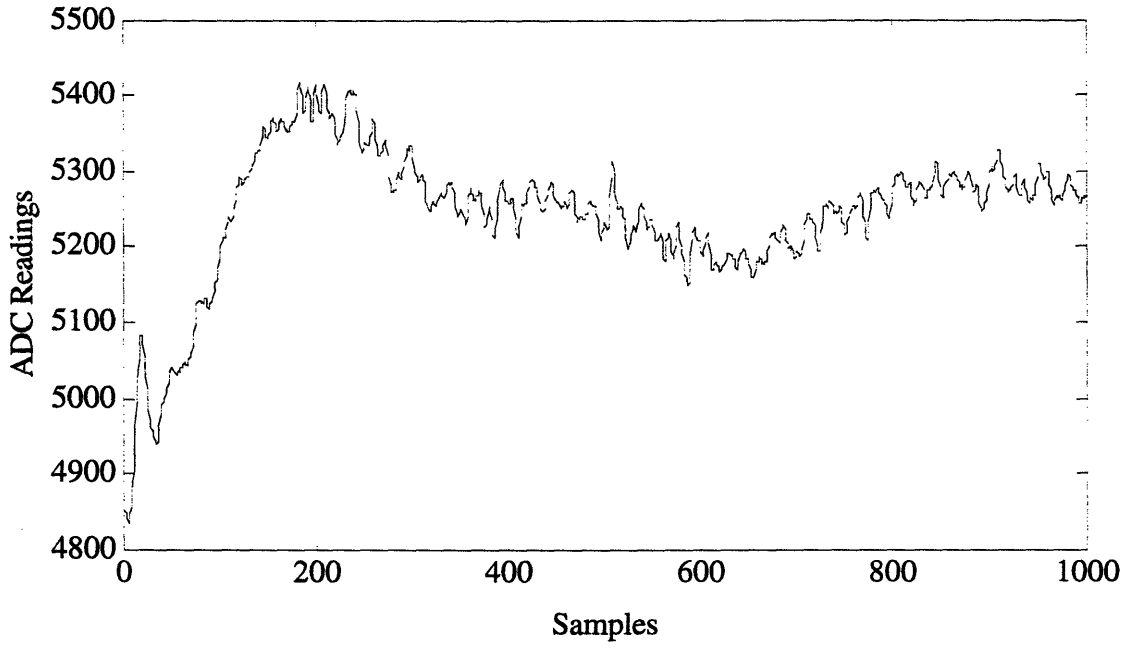


Figure 5.6: Fs=10 kHz, K=0.00065, Ti=0.03, Td=0.016 (unit) 100 $\frac{microsec}{sample}$

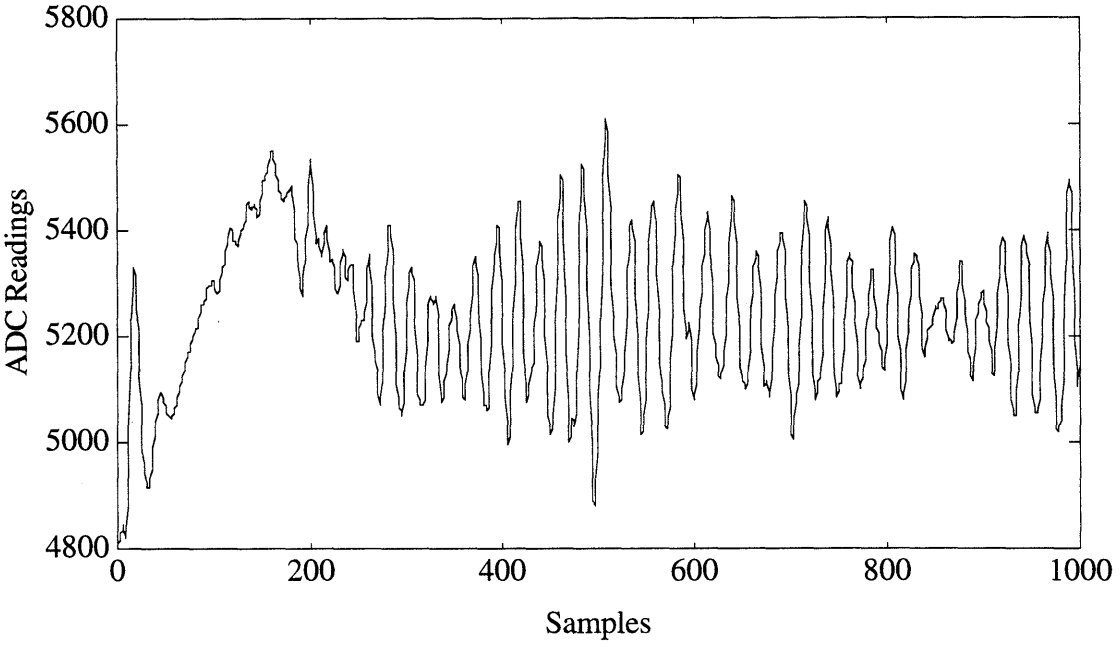


Figure 5.7: Fs=10 kHz, K=0.00065, Ti=0.065, Td=0.04 (unit) 100 $\frac{microsec}{sample}$

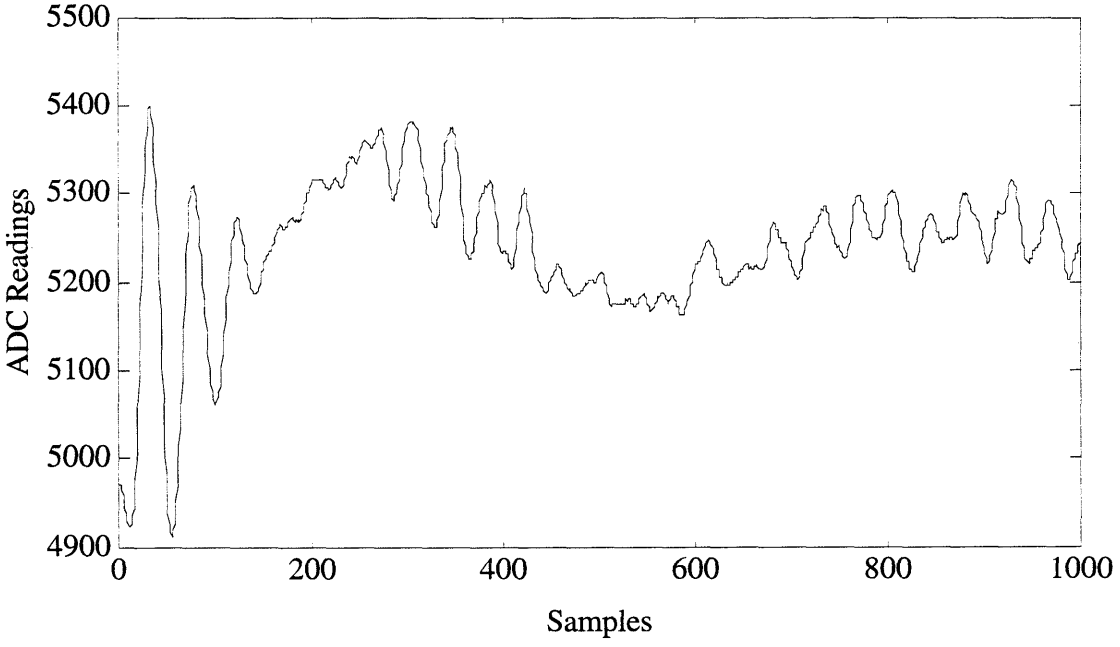


Figure 5.8: Fs=20 kHz, K=0.00065, Ti=0.065, Td=0.016 (unit) 50 $\frac{microsec}{sample}$

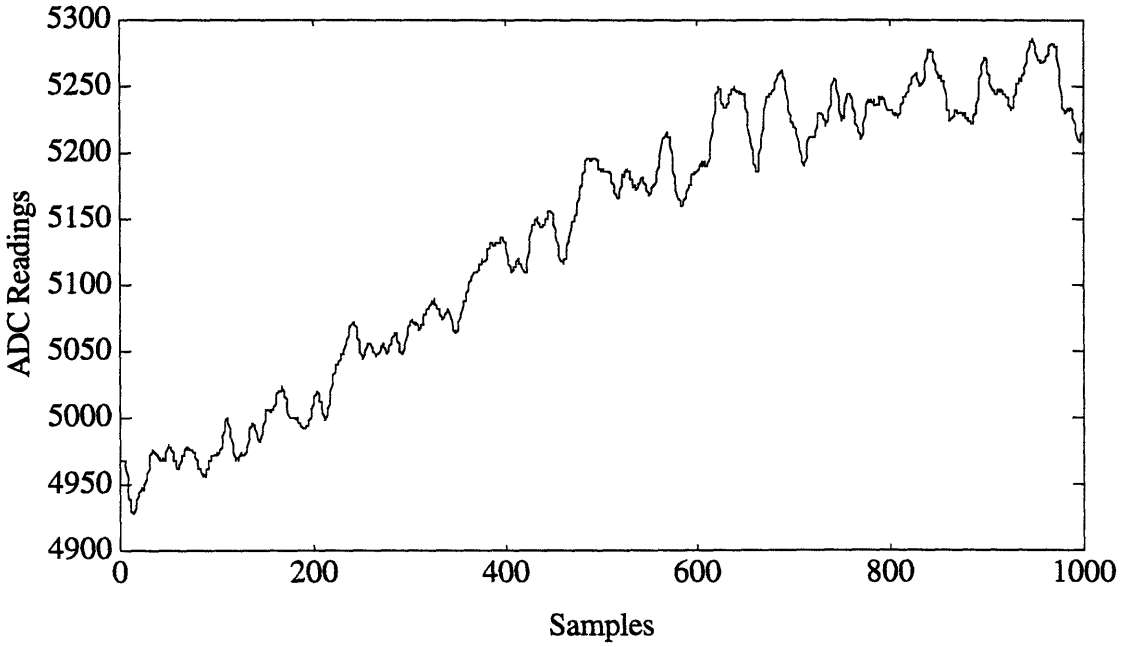


Figure 5.9: Fs=20 kHz, K=0.00065, Ti=0.065, Td=0.002 (unit) 50 $\frac{microsec}{sample}$

increased as shown in figure 5.8. This time, the sampling rate of 20 kHz is used. Notice that the amplitude of high frequency component of the response has increased to almost the step size. In order to reduce the noise effect, the derivative gain was decreased. The result is shown in figure 5.9. The plot shows the noise effect has been reduced but still remained as a significant error.

5.6 Software for Real Time Scanning

The scanning can be done in two modes, constant height mode and constant current mode. The constant height mode fixes the abosolute height of the tip during the scanning and uses the tunneling current for the atomic image data. The constant current mode on the other hand fixes the gap between the tip and the sample during the scanning and uses the z-direction voltage applied to the piezo tube for the atomic image data.

5.6.1 Constant Height Mode

The DSP uses four D/A channels to move the piezo tube in the x, y, z directions, and to apply a bias voltage between the tip and the sample material. It uses one A/D channel to measure the tunneling current.

The configuration of the hardware is carefully considered in this module. Since the A/D and the D/A do conversion at the same trigger, as the data is read and the control voltage is sent at the same sample, it is possible that tunneling current is measured while the tip is on the way to the next measurement point. To prevent this, the software implements an algorithm that alternates the A/D and D/A tasks in two sample periods.

The sampling rate for this module is usually around 6.4 kHz. This value is chosen to avoid the frequency region where the data signal is attenuated because of two filters that are used. As shown in figure 2.15, the filters attenuates the frequencies around 60 Hz and the region higher than 1 kHz. The 6.4 kHz sampling rate came from the consideration of the scanning speed. Here, the scanning speed of approximately 200 Hz(atoms/sec) is chosen which should be the reasonable value between 60 Hz and 1 kHz.

In the software, the scanning area can be changed using two macro variables. One defines the distance between the measurement points and the other defines the number of measurement points. In scanning experiments, we fequently used the 64× 64 measurement points and the 6 counts(≈ 2 $\mathring{A}$) between the measurement points as scanning parameters. Note that the number of the measurement points is limited by the amount of memory in DSP board. With each of 32,000 locations of data memory and program memory, a total of no more than 64,000 locations of on board random access memory can be used by DSP.

5.6.2 Constant Current Mode

Figure 5.10 shows the algorithm of the constant current mode module. The main routine is responsible only for the initialization of the variables and wait for the timer interrupt to be triggered. Every time an interrupt occurs, the program jumps to the interrupt service routine.

The interrupt routine performs the PID feedback and the x, y direction raster generation. The DSP updates the control command using PID feedback scheme and sums the z-direction control voltage. When the control loop counter expires, the DSP advances the tip in the x or y direction and records the average of the stored sum. When the scanning is completed, the DSP reports its end to the host computer and goes to an idling mode. Later, the PC uploads the averaged data and for process and further analysis.

The user inputs the PID gains, the sampling rate, the number of wait loops, and the number of control loops for the scanning parameters. The sampling rate, the number of control loop, and the PID gains are all determining factor of the scanning speed during this constant current mode. The faster sampling rate improves the performance of rejecting the disturbance and following the sample surface, but if the value is too high, the sensitivity of the noise will become larger as well. The number of control loop depends on the performance of the implemented PID controller. If the PID controller achieves a step response of the system which has the faster rise time and the settling time, the period that the tip has to stay at each data point will be reduced and the number of control loops can be set to a small value.

All the data taken throughout the PID feedback loop are summed and averaged before recorded. Such data averaging operation is one of the effort to improve the signal to noise ratio. (One can see in the step responses, such as the one shown in

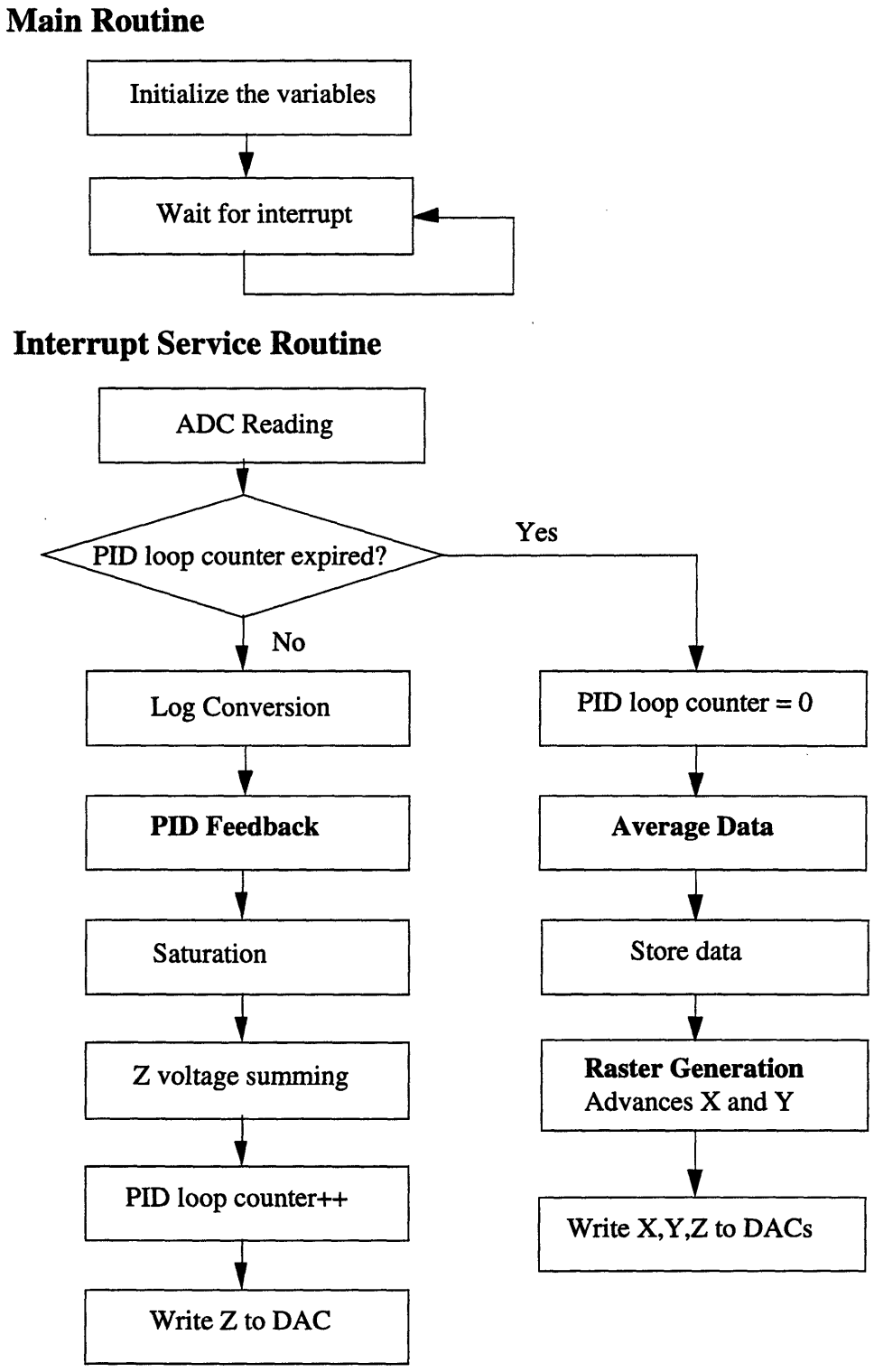


Figure 5.10: The Structure of Constant Current Scanning Routine

figure 5.8, that the significant errors are being caused by the high frequency noises.)

The wait loop is to introduce several sample delays between the XY advances and the PID feedback. Such delay gives the piezo tube some time to settle down at the new measurement location and therefore improves the performance of the PID controller and the accuracy of the averaged data.

Chapter 6
Experiments

6.1 Introduction

This chapter presents the atomic images obtained from the STM instrument described so far. We have scanned the highly oriented pyrolytic graphite (HOPG) in an air environment. The first section discusses the known structure of the sample graphite and its typical STM image. The following two sections describe the results of the two modes that have been conducted in this thesis work. Those are the constant height mode scanning and the constant current mode scanning.

6.2 The Structure of Highly Oriented Pyrolytic Graphite

Highly oriented pyrolytic graphite soon became our choice for the experiment sample because of its preparation convenience. Large, atomically flat surface could be easily obtained by cleaving with an adhesive tapes. Furthermore, the HOPG was relatively inert and little affected by contaminations or oxidizations from the surroundings and therefore it could be imaged in an air environment.[22,23] In our scanning experiments, we used the HOPG sample with the size of 6 mm $\times$ 6 mm $\times$ 1 mm provided by Advanced Ceramics Corporation

Figure 6.1 shows the hexagonal AB stacking sequence structure of the graphite, the most typical structure of the HOPG surface. The black dots in the diagram represents the carbon atoms of the graphite. The diagram shows that each carbon

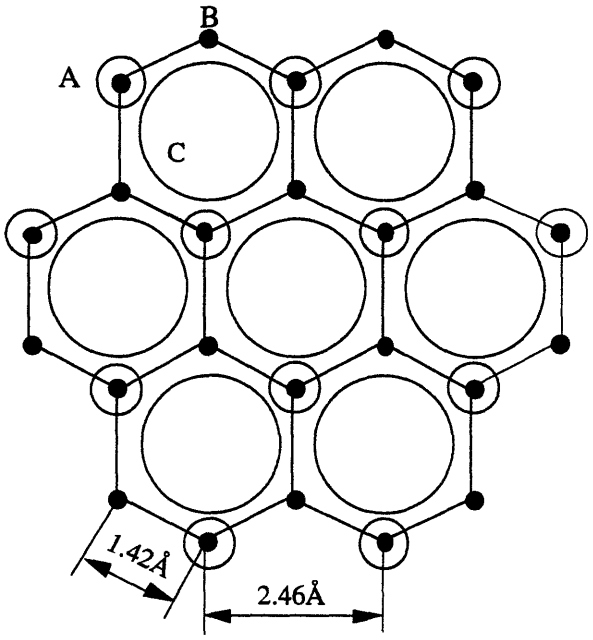


Figure 6.1: The Structure of a Highly Oriented Pyrolytic Graphite Surface

atom is located at either an A site or a B site. At site A the atoms in top layers coincide with those in the underneath layer, whereas at site B the atom does not have a corresponding atom in the underneath layer.[16,22] The distance between the neighboring A site atoms is 2.46 $\AA$. The distance beween an A site atom and the nearest B site atom is 1.42 $\AA$. At C site an atom exists in the underneath carbon layer, but the space directly above is just hollow.

According to the calculations of Selloni et al.[24] the atom at an A site is located slightly above the atoms at a site B by approximately 0.15 $\AA$. One can easily think that the image of the surface should have the brightest point at the A site. However, the real STM image of the atom shows the B site atom as the brightest region because of the fact that the local electronic density of states of the B site region greatly affects the images.[16] The typical STM image shows the brightest region in the B site, a slightly less bright region in the A site, and a dark region for the C site.

6.3 Constant Height Mode Scanning

The several experiments are conducted with different scanning parameters for the constant height scanning. The results are shown in figure 6.2 and figure 6.3. The first scan was conducted with the positive bias voltage 50 mV applied to the tip. The average value of the tunneling current generated at this voltage was 7 nA. The 64 by 64 data points were taken on the 2-D atom plane where the distance between the data points was $0.47\mathring{A}$. The scanning speed along the x-axis line was 600 Hz (atoms/sec).

Figure 6.2 shows the grey scale of the graphite image taken during the first scan experiment. The image shown in the figure corresponds to an area of 10 $\mathring{A}$ × 16 $\mathring{A}$. Each bright region represents a B site while the dark region represents a C site which is located at the center of a hexagon as shown before. The A site can be recognizable if seen carefully. It forms a triangular shape positioned opposite to the B site triangle. The C site should be at the center of this triangle.

The fact that the lower part of the image is brighter than the upper part shows that the control of the thermal drift has not been succesful.

The second experiment imaged an area of 8 $\mathring{A}$ × 6 $\mathring{A}$. The image was measured with an average tunneling current of 8 nA and a bias voltage of 45 mV. This time, the scanning speed was decreased to 200 Hz. The result is shown in figure 6.3. In the figure, the pattern of the graphite layer structure is clearly evident. However, the A site atoms forming a hexagonal pattern are barely recognizable because neighboring B site and C site regions are too bright to reveal such pattern.

Figure 6.4 shows the X-Z plane view of the atom image taken during the second experiment. From the figure, the amplitude of the atom signal was measured. The amplitude was about 2 $\mathring{A}$ which agreed well with other's results. [22]

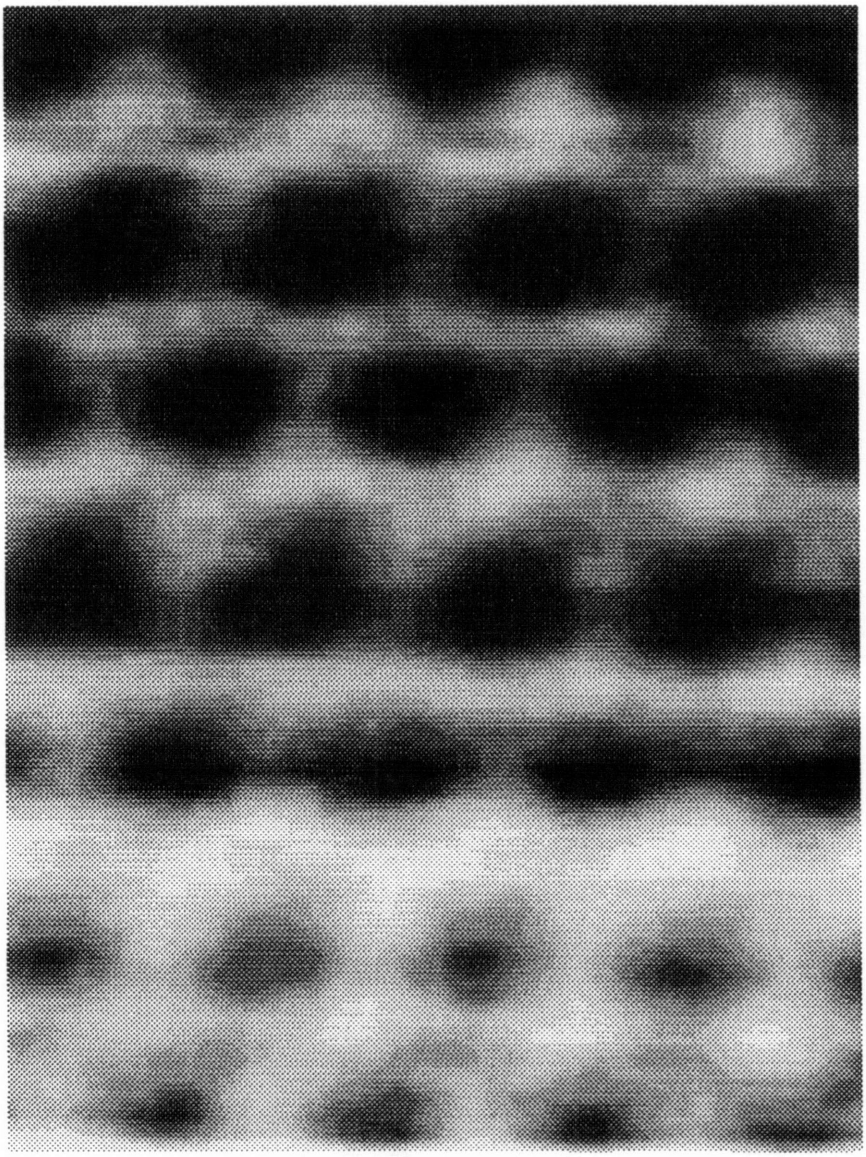


Figure 6.2: The Grey-scale Image of HOPG Surface (10Å x 16Å)

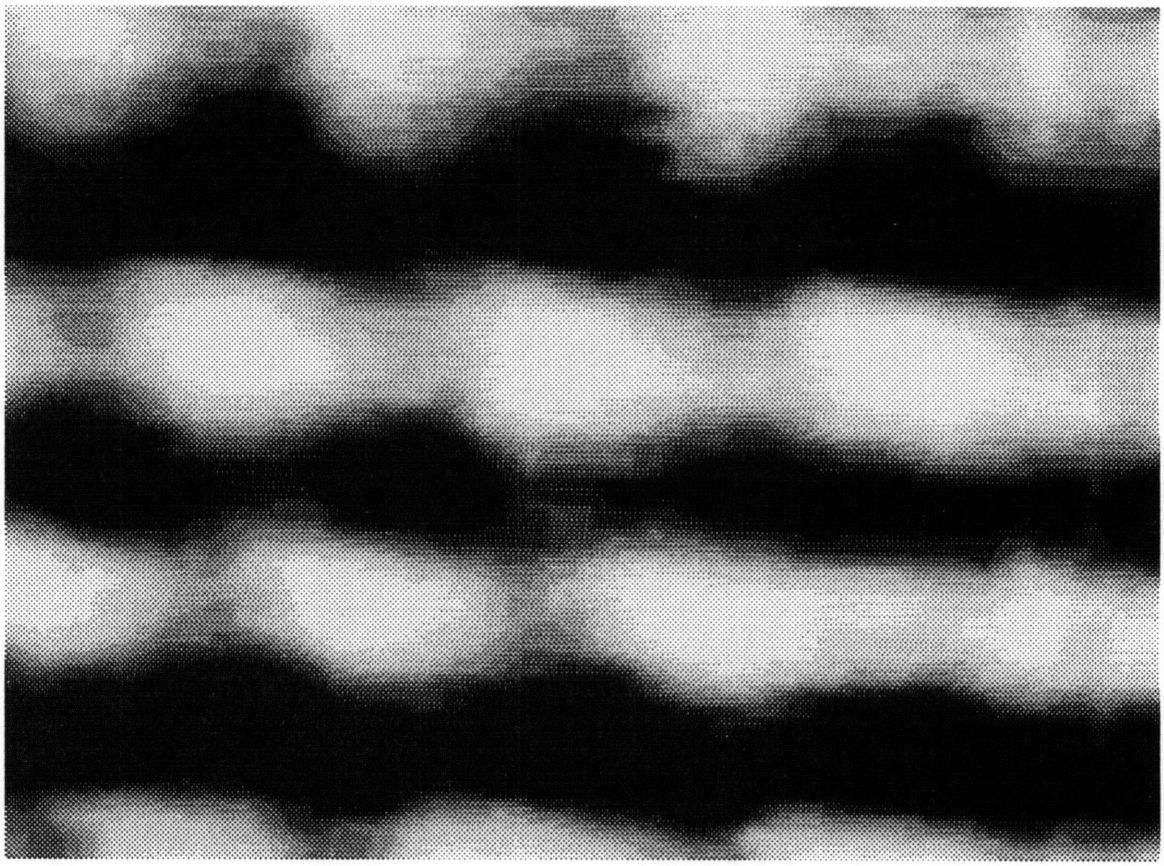


Figure 6.3: The Grey-scale Image of HOPG Surface (8Å x 6Å)

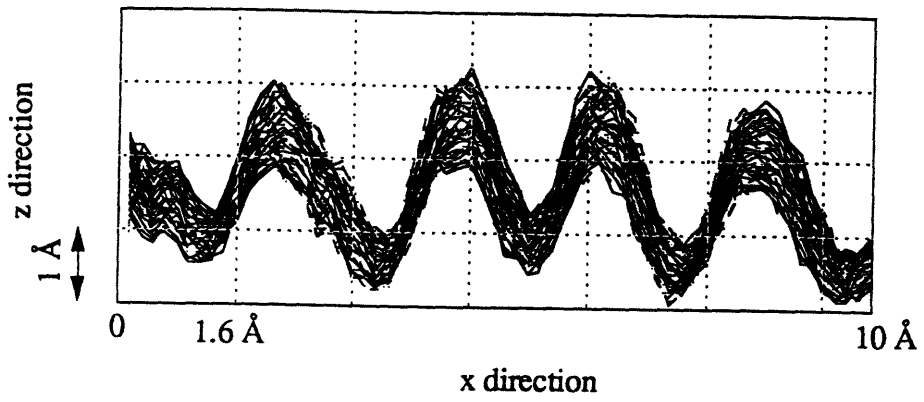


Figure 6.4: The X-Z Plane View of the Atom Surface

The grey scale image taken from another set of experiment is shown in figure 6.5. The data were taken with a tunneling current of 7 nA. The rest of the scanning parameters were the same as the first experiment. The figure shows part of the scanned image, an area of 16 $\mathring{A} \times 10 \mathring{A}$. Although it shows the image similar to an atomic structure evident in other experiment results, the atom structure shows rather unusual pattern. It provides a good insight of how the shape of the scanning surface is formed. The diagonally elongated atom shape pointing to the bottom suggest that the scanned atom surface is curved towards the bottom.

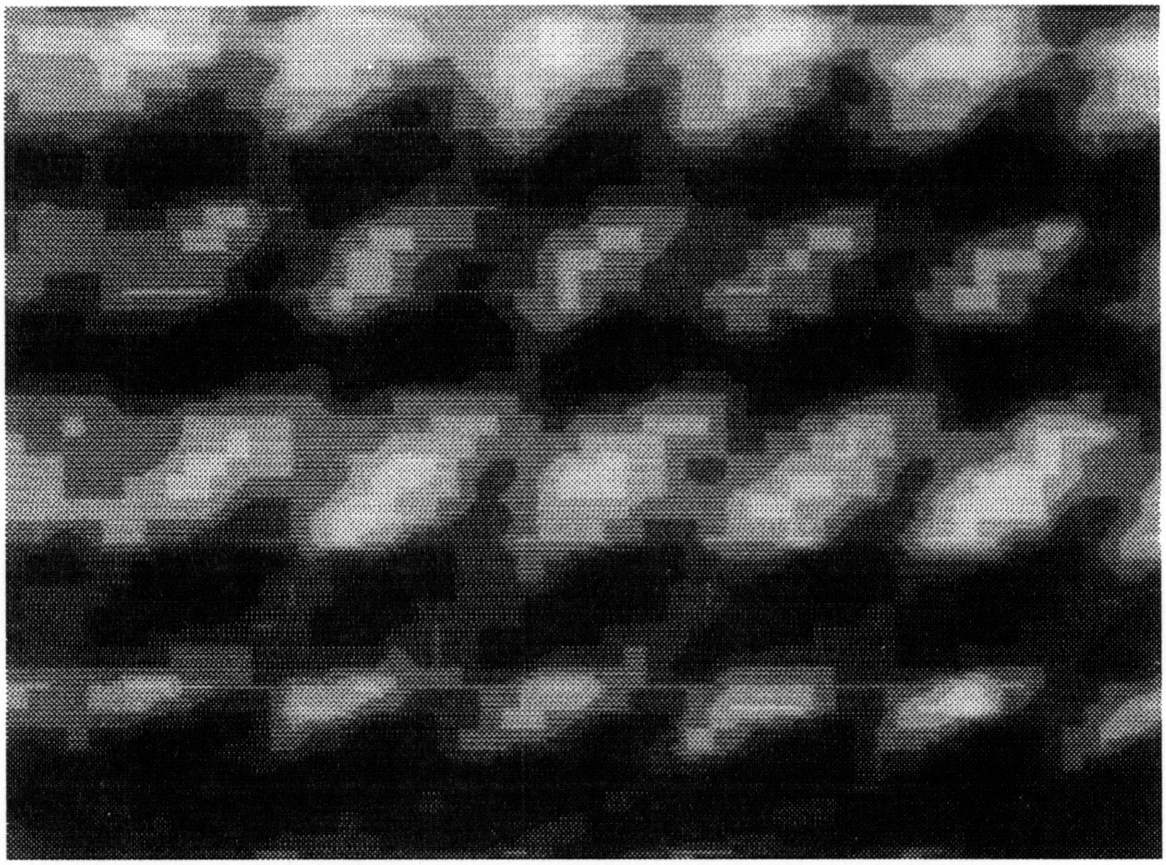


Figure 6.5: The Grey-scale Image of HOPG Surface (16Å x 10Å)

6.4 Constant Current Mode Scanning

The result of the constant current scanning experiment is shown in figure 6.6. This constant current imaged area of 9 $\mathring{A} \times 6.5\mathring{A}$, was measured with a tunneling current of 10 nA and a bias voltage of 45 mV.

The PID feedback control system used a sampling rate of 10 kHz. The P, I, D gains in the software were set to 0.00065, 0.065, 0.016 respectively. However, it has been discovered that because of environmental changes, the PID gains should be updated each time a new experiment is conducted. The figure shows the effect of the D gain and the fast sampling time clearly. Although the hexagonal pattern of the atom structure is evident in lower part of the figure, the image is much affected by the surrounding noises. The effort to increase the scanning speed to reduce the thermal drift resulted in amplifying the noise effect.

The minimum time between the data samples are set to be 5 milliseconds. At this rate, the tip should advance in the x and y directions at a speed of 12 Hz (30.5 $\frac{\mathring{A}}{sec}$).

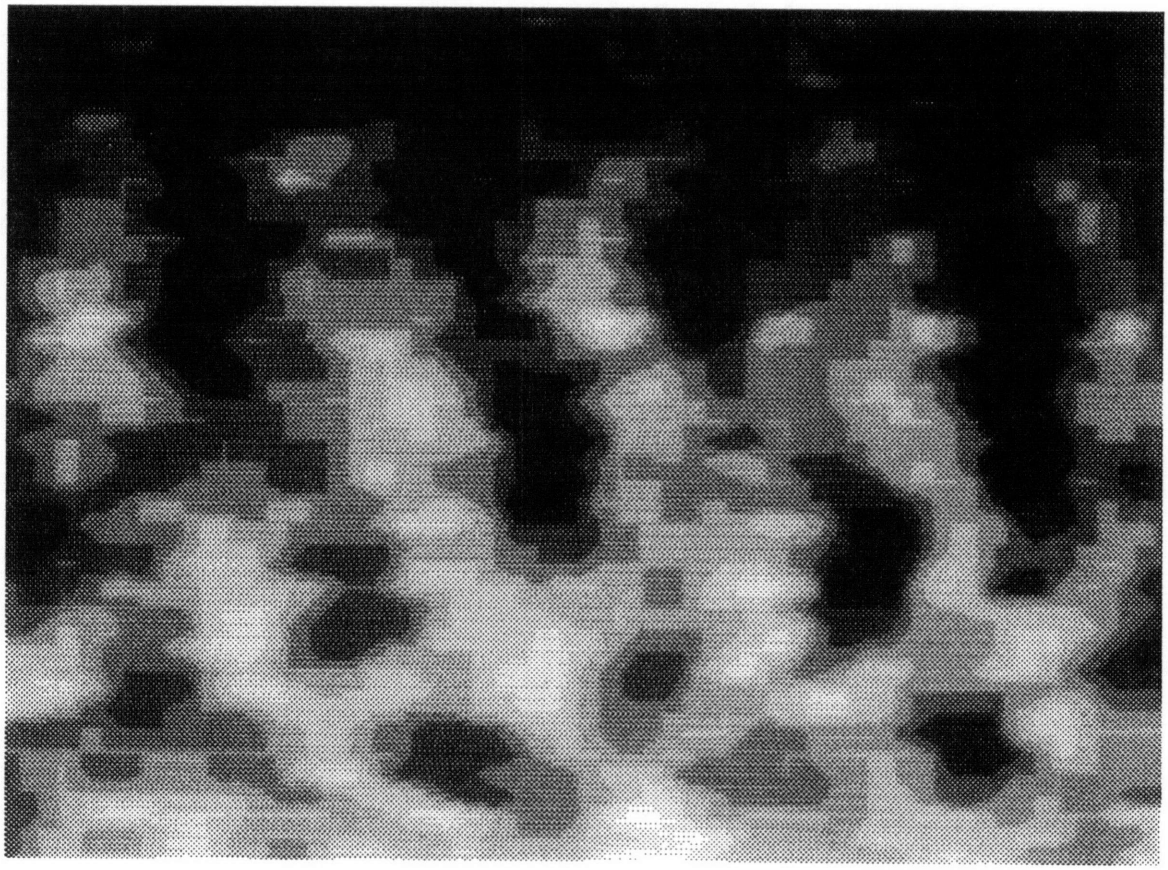


Figure 6.6: The Grey-scale Image of HOPG Surface (9Å x 6.5Å)

Chapter 7
Conclusion

The purpose of this thesis work was to provide a supporting environment of the application development for the scanning tunneling microscope, being underway in the laboratory of manufacturing and productivity. The task has been accomplished by implementing the digital signal processor based control system. The software is written for both the digital signal processor and the host personal computer. The DSP program is written in assembly language in order to minimize the program's execution time. The software for the basic STM control operations, the user interface, and the DSP interfaces are written for the high-level application developers. The steps neccessary for developing the application software are suggested.

All the softwares written for the STM control are proved to be satisfactory, but still need to be improved on several areas. First, the user interface should be improved. The current menu driven user interface structure is aimed for higher quality user interface routines such as Windows. Secondly, the software should support the wider range of the user's selections of the scanning parameters. The future software should provide the features such as a user control of number of measurement data points. Thirdly, the possibility of software compensation of hardware error should be explored more. The hardware errors such as the thermal drift, the electrical noise, and the piezo's hysteresis showed some regular patterns which may be cured by the software correction. Lastly the future applications software should take a full advantage of the digital signal processors's features.

The images taken from both the constant height mode and the constant current

mode experiment are presented. The result of constant current mode turned out to be not as good as the result of constant height mode. The main reason should be the improper adjustment of proportional, integral, and derivative gain value. The PID gains with the low sensitivity to the ambient noise should be found. It would be better to characterize the effect of an ambient noises when deriving the optimal gain values. If one could come up with the mathematical model of the noises, he can even perform the simulations using the provided block diagram model to see the noise sensitivity.

Also in the thesis, the modeling of the PID feedback loop was performed. The bond graph approach was adapted in deriving the 2nd order approximated model of the piezoelectric tube. The simulation based on the derived model showed that the outputs closely match the experimental results. The simulation result revealed the fact that the notch filter we adopted greatly affects the overall system response and thus determines the dynamics of controlled plants.

Again, the task of this thesis work to implant the new control system and to provide an application development environment have been successful and only remaining is to proceed with the development with the provided software and hardware tools.

Appendix A

Resources

The following companies were the supplier of the products we used in this project. The complete address is given for each company for further references.

- **Advanced Ceramic Technology**

  990F Enterprise St., Dept. TR

  Orange, CA 92667

  Tel) (714) 538-2524

  Fax) (714) 538-2589

- **Analog Devices, Inc.**

  DSP Division

  One Technology Way, P.O. Box 9106

  Norwood, Massachusetts 02062-9106

  Tel) (617) 461-3672

  Educational Support

  Tel) (617) 329-4700

  Fax) (617) 461-3010

- **Burleigh Instrument, Inc.**

  Burleigh Park P.O. Box E

  Fishers, NY 14453

  Tel) (716) 924-9355

  Fax) (716) 924-9072

- **JRG Signal Processing**

  Tel) (508) 655-9208

- **NPC Computer Corp.**

  1320 Centre Street

  Newton Centre, MA 02159

  Tel) (617) 965-8325

  Fax) (617) 965-3784

- **Omega Engineering, Inc.**

  P.O. Box 4047

  Stamford, CT 06907-0047

  Tel) (203) 359-1660

  Fax) (203) 359-7700

- **The Math Work, Inc.**

  24 Prime Park Way

  Natick, Massachusetts 01760

  Tel) (508) 653-1415

- **Microsoft Corp.**

  One Microsoft Way

  Redmond, WA 98052-6399

  Tel) (206) 454-2030

- **Stavely Sensor, Inc.**

  91 Prestige Park

  East Hartford, CT 06108

Tel) (203) 289-5428

Fax) (203) 289-3189

Appendix B

DSP Code

B.1 Architecture File

```
.SYSTEM          KYT;
.PROCESSOR =     ADSP21020;

.SEGMENT /ROM   /BEGIN=0x000000 /END=0x000007 /PM emu_svc;
.SEGMENT /ROM   /BEGIN=0x000008 /END=0x00000F /PM rst_svc;
.SEGMENT /ROM   /BEGIN=0x000018 /END=0x00001F /PM sovf_svc;
.SEGMENT /ROM   /BEGIN=0x000020 /END=0x000027 /PM tmzh_svc;
.SEGMENT /ROM   /BEGIN=0x000028 /END=0x00002F /PM irq3_svc;
.SEGMENT /ROM   /BEGIN=0x000030 /END=0x000037 /PM irq2_svc;
.SEGMENT /ROM   /BEGIN=0x000038 /END=0x00003F /PM irq1_svc;
.SEGMENT /ROM   /BEGIN=0x000040 /END=0x000047 /PM irq0_svc;
.SEGMENT /ROM   /BEGIN=0x000058 /END=0x00005F /PM cb7_svc;
.SEGMENT /ROM   /BEGIN=0x000060 /END=0x000067 /PM cb15_svc;
.SEGMENT /ROM   /BEGIN=0x000070 /END=0x000077 /PM tmzl_svc;
.SEGMENT /ROM   /BEGIN=0x000078 /END=0x00007F /PM fix_svc;
.SEGMENT /ROM   /BEGIN=0x000080 /END=0x000087 /PM flto_svc;
.SEGMENT /ROM   /BEGIN=0x000088 /END=0x00008F /PM fltu_svc;
.SEGMENT /ROM   /BEGIN=0x000090 /END=0x000097 /PM flti_svc;
.SEGMENT /ROM   /BEGIN=0x0000C0 /END=0x0000C7 /PM sft0_svc;
.SEGMENT /ROM   /BEGIN=0x0000C8 /END=0x0000CF /PM sft1_svc;
.SEGMENT /ROM   /BEGIN=0x0000D0 /END=0x0000D7 /PM sft2_svc;
.SEGMENT /ROM   /BEGIN=0x0000D8 /END=0x0000DF /PM sft3_svc;
.SEGMENT /ROM   /BEGIN=0x0000E0 /END=0x0000E7 /PM sft4_svc;
.SEGMENT /ROM   /BEGIN=0x0000E8 /END=0x0000EF /PM sft5_svc;
.SEGMENT /ROM   /BEGIN=0x0000F0 /END=0x0000F7 /PM sft6_svc;
.SEGMENT /ROM   /BEGIN=0x0000F8 /END=0x0000FF /PM sft7_svc;

.SEGMENT /RAM   /BEGIN=0x000100 /END=0x0007FF /PM pm_code;
.SEGMENT /RAM   /BEGIN=0x000800 /END=0x000FFF /PM pm_data;

.SEGMENT /PORT /BEGIN=0X800000 /END=0X800000 /PM pm_port1;
.SEGMENT /PORT /BEGIN=0X800001 /END=0X800001 /PM pm_port2;

.SEGMENT /RAM   /BEGIN=0x00000000 /END=0x0000000F /DM dm_userinput;
.SEGMENT /RAM   /BEGIN=0x00000010 /END=0x0000002f /DM dm_data;

.SEGMENT /PORT /BEGIN=0X20000000 /END=0X20000000 /DM dm_port1;
.SEGMENT /PORT /BEGIN=0X20000002 /END=0X20000002 /DM dm_port2;
.SEGMENT /PORT /BEGIN=0X40000000 /END=0X40000000 /DM dm_port3;
.SEGMENT /PORT /BEGIN=0X40000001 /END=0X40000001 /DM dm_port4;
.SEGMENT /PORT /BEGIN=0X40000002 /END=0X40000002 /DM dm_port5;

.BANK /PM0 /PGSIZE=256 /WTSTATES=0 /WTMODE=INTERNAL /BEGIN=0X000000;

.BANK /DM0 /PGSIZE=256 /WTSTATES=0 /WTMODE=INTERNAL /BEGIN=0X00000000;
.BANK /DM1 /PGSIZE=256 /WTSTATES=1 /WTMODE=INTERNAL /BEGIN=0X40000000;


.ENDSYS;
```

B.2 .ASM Files

```
/*****************************************************************

       init.asm--This program sets each D/A channels to preset value. The following
              values are assigned to each channels.

              D/A channel 1 - z-piezo       ------ 0 DAC
                  channel 2 - bias voltage  ------ 0x156d DAC( bias + offset )
                  channel 3 - x-piezo       ------ 0 DAC
                  channel 4 - y-piezo       ------ 0 DAC

              There exist huge offset voltages in each D/A channels. The offset
              voltage for DAC value 0 was measure at
                     D/A channel 1 -> -0.241 V - 0x626 DAC Value
                         channel 2 -> -0.237 V - 0x611 DAC Value
                         channel 3 -> -0.226 V - 0x5c9 DAC Value
                         channel 4 -> -0.248 V - 0x659 DAC Value.
              The resistor pack for 13 kHz was used when the offsets are measured.

                     written by Jungmok Bae          2/11/94
                     Final revision                  3/11/94
*****************************************************************/
#define BIAS    0x156d

.segment /pm    pm_port1;
.var   ADFifo;
.endseg;

.segment /pm    pm_port2;
.var   ANGBStatus;
.endseg;

.segment /dm    dm_port1;
.var   StatusReg;
.endseg;

.segment /dm    dm_port2;
.var    DigitalIO;
.endseg;

.segment /dm    dm_port3;
.var   DAFifo;
.endseg;

.segment /dm    dm_port4;
.var   ControlReg;
.endseg;

.segment /dm    dm_port5;
.var   ChanNum;
.endseg;

.SEGMENT/PM             rst_svc; { program starts at the reset vector }
       PMWAIT=0x00A1; {/pms0=0ws, /pms1=1ws}
       DMWAIT=0x9421; {/dms0=0ws, /dms1=0ws, /dms2=1ws, /dms3=0ws}

          jump start;
.ENDSEG;

.segment/pm             pm_code;
start:
lp:     r4=0x1;         { let pc know the scanning has started}
       dm(StatusReg)=r4;
```

```
        r9=0x8000;

        bit set MODE2 0x20000;  { flag 2 is an output }
        bit clr ASTAT 0x200000; { set flag2 to low }
        r4=0x0;
        dm(ControlReg)=r4;      {reset analog board}
        nop;nop;nop;
        r4=0x20;                { 0 a/d channels & 4 d/a channels }
        dm(ChanNum)=r4;       { number of channels }
        nop; nop; nop;
        r4=0x80;                { release ang reset & go mode=0 (go on flag2) }
        dm(ControlReg)=r4;      { control register }
        nop; nop; nop;

{ read the first a/d and write the initial d/a outputs }

wt1:    r10=pm(ANGBStatus);      {wait for the fifo to finish its work.}
        r8=0x3;
        r10=r10 AND r8;
        if ne jump wt1;

        r0=0;           /*previous z-vol. in 14bit from pidcon.asm*/
        r1=BIAS;

        r0=ASHIFT r0 by 2;
        r0=r0 XOR r9;       { and invert MSB:due to binary offset format}
        r1=r1 XOR r9;

        dm(DAFifo)=r0;      {send z-piezo sig-initial 0V}
        dm(DAFifo)=r1;      {apply bias voltage-initial 50mV}
        dm(DAFifo)=r0;      {apply x-piezo vol.}
        dm(DAFifo)=r0;      {apply y-piezo vol.}

        bit set ASTAT 0x200000;  { toggle flag2 to start first conv }
        nop; nop; nop; nop;
        bit clr ASTAT 0x200000;
        nop; nop; nop;

wt2:    r10=pm(ANGBStatus);        { wait for the first a/d to trash the }
        r8=0x3;                  { whatever sits in the fifo with new data.}
        r10=r10 AND r8;
        if ne jump wt2;

        r4=0x0;
        dm(StatusReg)=r4; { let pc know the scanning has finished}

adflush:r4=pm(ANGBStatus); {make sure last conversion is done}
        r8=0x3;
        r4=r4 AND r8;
        if ne jump adflush;
        r4=pm(ADFifo); {flush 1 a/d result from fifo}

        idle;
.ENDSEG;

/*************************************************************************

        moveiw.asm--This module controls the inchworm motor.

                user input 0 -  dm(0x0000)
                                sampling rate of a/d.
                user input 1 -  dm(0x0001)
                                Clock counter.
                user input 2 -  dm(0x0002)
```

```
                        flag : upward or downward
              ......This program is compiled with aenv1.ach file.

                                written by Jungmok Bae  2/11/94
*********************************************************************/

.segment /pm    pm_port1;
.var   ADFifo;
.endseg;

.segment /pm    pm_port2;
.var   ANGBStatus;
.endseg;

.segment /dm    dm_port1;
.var   StatusReg;
.endseg;

.segment /dm    dm_port2;
.var    DigitalIO;
.endseg;

.segment /dm    dm_port3;
.var   DAFifo;
.endseg;

.segment /dm    dm_port4;
.var   ControlReg;
.endseg;

.segment /dm    dm_port5;
.var   ChanNum;
.endseg;

.segment /dm    dm_data;
.var    LoopCount;
.endseg;

.segment /pm     rst_svc; { program starts at the reset vector }
       PMWAIT=0x00A1; {/pms0=0ws, /pms1=1ws}
       DMWAIT=0x9421; {/dms0=0ws, /dms1=0ws, /dms2=1ws, /dms3=0ws}

          jump start;
.endseg;

.segment/pm     pm_code;
start:          r4=0x0;
             dm(ControlReg)=r4;       {reset analog board}
             nop;nop;nop;
             r4=0x0;                 { 0 a/d channels & 0 d/a channels }
             dm(ChanNum)=r4;       { number of channels }
             nop; nop; nop;

             call init_timer(db);
               r4=0x1;          { let pc know the coarse approach has started}
               dm(StatusReg)=r4;
             r0=0;
             dm(LoopCount)=r0;

             bit set MODE2 0x20;      { enable timer }
             bit set MODE1 0x1000;    { enable interrupts }

       intwt:  idle;
```

```
                jump intwt;

Sample:           r0=dm(LoopCount);
                r1=dm(0x0001);
                r2=r1-r0;
                if eq jump endofprog;

                r1=dm(0x0002);
                r6=r1+1;          { the value of the clock : 1 }
                dm(DigitalIO)=r6; { send the clock through Digital I/O }
                lcntr=0xc8, do hold until lce;
        hold:    nop;

                r6=r1;          { the value of the clock : 0 }
                dm(DigitalIO)=r6;{ send the clock through Digital I/O }

                r0=dm(LoopCount);
                r0=r0+1;
                dm(LoopCount)=r0;

                rti;


init_timer:       bit clr MODE1 0x1000;    { disable all the previous interrupts }
                bit set irptl 0;         { clear pending interrupt }
                bit set IMASK 0x12;      { enable timer int.    }

                r10=dm(0x0000);  { sample period will be in dm(0x1011) }
                TPERIOD=r10;     { set up the timer }
                TCOUNT=r10;

                rts;


endofprog:        bit clr MODE2 0x20;      {disable timer}
                bit clr MODE1 0x1000;    {disable ints}

                r4=0x0;
                dm(StatusReg)=r4; { let pc know the coarse approach has finished}

                idle;
.ENDSEG;

.SEGMENT/PM  tmzh_svc;
jump Sample;
idle;
.ENDSEG;



/*****************************************************************

        coarse11.asm - This program executes the automatic coarse
                approach.

        Channel usage
                D/A channel 1 - z voltage          ---> r0 ( reserved reg.)
                    channel 2 - bias               ---> r1 (      "       )
                A/D channel 1 - tunneling current ---> r2 (      "       )
        User input
                user input 0 -  dm(0x0000)
                              sampling rate of a/d.
                user input 1 -  dm(0x0001)
                              ratio between a/d fs and IW's speed.
```

```
*****************************************************************/
#define BIAS    0xf5c          /* the bias voltage */
#define TCLMT   0xccc          /* the tunneling voltage limit */
#define BACKSTEP 0x14          /* the # of back up step when it detects TC */

.segment /pm    pm_port1;
.var   ADFifo;
.endseg;

.segment /pm    pm_port2;
.var   ANGBStatus;
.endseg;

.segment /dm    dm_port1;
.var   StatusReg;
.endseg;

.segment /dm    dm_port2;
.var    DigitalIO;
.endseg;

.segment /dm    dm_port3;
.var   DAFifo;
.endseg;

.segment /dm    dm_port4;
.var   ControlReg;
.endseg;

.segment /dm    dm_port5;
.var   ChanNum;
.endseg;

.segment /dm    dm_data;
.var    Counter;
.endseg;

.segment /pm     rst_svc; { program starts at the reset vector }
      PMWAIT=0x00A1; {/pms0=0ws, /pms1=1ws}
      DMWAIT=0x9421; {/dms0=0ws, /dms1=0ws, /dms2=1ws, /dms3=0ws}

         jump start;
.endseg;

.segment/pm     pm_code;
start:          r4=0x1;         { let pc know the coarse approach has started}
             dm(StatusReg)=r4;

             call init_analog(db);
               r9=0x8000;
               r4=0;
             call init_timer(db);
               dm(0x1010)=r4;
               r5=dm(0x0001);
             dm(Counter)=r5;

             bit set MODE2 0x20;     { enable timer }
             bit set MODE1 0x1000;   { enable interrupts }

      intwt:  idle;
             jump intwt;
```

```
Sample:          dm(DAFifo)=r0;       {send z-piezo sig-initial 0V}
                 r12=pm(ADFifo);      {read tunneling current}
                 dm(DAFifo)=r1;     {send bias voltage-initial 100mV}

                 bit set ASTAT 0x200000;  { toggle flag2 to start next conv }
                 r10=pm(ANGBStatus);
                 r8=0x3;
                 r10=r10 AND r8;
                 if ne jump tofast;
                 bit clr ASTAT 0x200000;
                 nop; nop; nop;

                 r2=fext r12 by 0:14(SE);   {sign extension of a/d result!!!}

                 r5=0x186;       {a/d offset vol. compensation}
                 r2=r2+r5;
                 r3=TCLMT;   {TC : 2V in 14 bit resolution.}
                 r3=r3-r2;   {checking the tunneling current if it goes above 4V}
                 if le jump backoff; {if TC is generated go to the end}

                 r5=dm(Counter);
                 r5=r5-1;
                 dm(Counter)=r5;
                 if eq call moveIW;

        go:      r0=0x0;
                 r1=BIAS;
                 r5=0x611;              {d/a offset compensation}
                 r1=r1+r5;
                 r0=r0 XOR r9;
                 r1=r1 XOR r9;

                 rti;

init_analog:     bit set MODE2 0x20000;  { flag 2 is an output }
                 bit clr ASTAT 0x200000; { set flag2 to low }
                 r4=0x0;
                 dm(ControlReg)=r4;      {reset analog board}
                 nop;nop;nop;
                 r4=0x11;                { 1 a/d channels & 2 d/a channels }
                 dm(ChanNum)=r4;      { number of channels }
                 nop; nop; nop;
                 r4=0x80;                { release ang reset & go mode=0 (go on flag2) }
                 dm(ControlReg)=r4;      { control register }
                 nop; nop; nop;

{ read the first a/d and write the initial d/a outputs }
        wt1:     r10=pm(ANGBStatus);        { wait for the first a/d to trash the }
                 r8=0x3;                  { whatever sits in the fifo with new data.}
                 r10=r10 AND r8;
                 if ne jump wt1;

                 r0=0x0;
                 r1=BIAS;
                 r5=0x611;               {d/a offset compensation}
                 r1=r1+r5;
                 r0=r0 XOR r9;
                 r1=r1 XOR r9;
                 dm(DAFifo)=r0;       {send z-piezo sig-initial 0V}
                 dm(DAFifo)=r1;       {apply bias voltage-initial 50mV}

                 bit set ASTAT 0x200000;  { toggle flag2 to start first conv }
                 nop; nop; nop; nop;
                 bit clr ASTAT 0x200000;
```

```
            nop; nop; nop;

      wt2:   r10=pm(ANGBStatus);        { wait for the first a/d to trash the }
             r8=0x3;                  { whatever sits in the fifo with new data.}
             r10=r10 AND r8;
             if ne jump wt2;

             rts;

init_timer:    bit clr MODE1 0x1000;   { disable all the previous interrupts }
             bit set irptl 0;        { clear pending interrupt }
             bit set IMASK 0x12;     { enable timer int.   }

             r10=dm(0x0000);  { sample period will be in dm(0x1011) }
             TPERIOD=r10;     { set up the timer }
             TCOUNT=r10;

             rts;

moveIW:        r6=0x1;         { the value of the clock : 1 }
             dm(DigitalIO)=r6; { send the clock through Digital I/O }
             lcntr=0xc8, do hold until lce;
      hold:   nop;

             r6=0x0;         { the value of the clock : 0 }
             dm(DigitalIO)=r6;{ send the clock through Digital I/O }

             r5=dm(0x0001);  { update the ratio }
             dm(Counter)=r5;
             rts;

tofast:        bit clr MODE2 0x20;     {disable timer}
             bit clr MODE1 0x1000;   {disable ints}

    adflush0:   r4=pm(ANGBStatus); {make sure the z-piezo goes to 0V pos.}
             r8=0x3;
             r4=r4 AND r8;
             if ne jump adflush0;
             r4=pm(ADFifo); {flush 1 a/d result from fifo}

             r4=0x23;           { send error to status }
             dm(StatusReg)=r4;

             jump endofprog;

backoff:       bit clr MODE2 0x20;     {disable timer}
             bit clr MODE1 0x1000;   {disable ints}

    adflush1:   r4=pm(ANGBStatus); {make sure the z-piezo goes to -5V pos.}
             r8=0x3;
             r4=r4 AND r8;
             if ne jump adflush1;
             r4=pm(ADFifo); {flush 1 a/d result from fifo}

             lcntr=BACKSTEP, do lp2 until lce;   {backing off about 40 steps (100kHz
spd)}
             r6=0x5;         { the value of the clock : 1 }
             dm(DigitalIO)=r6; { send the clock through Digital I/O }
             lcntr=0x3e8, do tick until lce;   { interval bet high & low}
      tick:   nop;

             r6=0x4;                        { the value of the clock : 0 }
             dm(DigitalIO)=r6;    {send the clock through Digital I/O }
             lcntr=0x3e8, do tock until lce;  { interval bet low & high}
```

```
      tock:   nop;
lp2:            nop;

              r4=0x0;
              dm(StatusReg)=r4; { let pc know the coarse approach has finished}

endofprog:      idle;
.ENDSEG;

.SEGMENT/PM  tmzh_svc;
jump Sample;
idle;
.ENDSEG;


/*********************************************************************

      fine1.asm--This program performs the automatic fine
            approach. The module generaets the clock to control the inchworm motor
            and also saw tooth voltage to control the piezo tube in z direction.
      Channel usage:
            D/A channel 1 - z-piezo
                channel 2 - bias voltage
            A/D channel 1 - tunneling current

      reserved register:
            r0 : z-piezo
            r1 : Bias
            r2 : Tunneling current
      reserved data memory:
            0x0001 : z-piezo for next step
            0x0000 : sampling time


                  written by Jungmok Bae          1/13/94
                  Final revision                  3/11/94
*********************************************************************/
#define BIAS    0x156d          /* bias voltage : 0.6V */

.segment /pm    pm_port1;
.var   ADFifo;
.endseg;

.segment /pm    pm_port2;
.var   ANGBStatus;
.endseg;

.segment /dm    dm_port1;
.var   StatusReg;
.endseg;

.segment /dm    dm_port2;
.var    DigitalIO;
.endseg;

.segment /dm    dm_port3;
.var   DAFifo;
.endseg;

.segment /dm    dm_port4;
.var   ControlReg;
.endseg;

.segment /dm    dm_port5;
```

```
.var   ChanNum;
.endseg;

.segment /dm    dm_data;
.var    Interval;
.endseg;

.SEGMENT/PM             rst_svc; { program starts at the reset vector }
       r4=0x0;
       dm(ControlReg)=r4;       {reset analog board}

          jump start;
.ENDSEG;

.segment/pm             pm_code;
start:  PMWAIT=0x00A1; {/pms0=0ws, /pms1=1ws}
       DMWAIT=0x9421; {/dms0=0ws, /dms1=0ws, /dms2=1ws, /dms3=0ws}

       r4=0x1;          { let pc know the coarse approach has started}
       dm(StatusReg)=r4;

       call init_analog(db);
         r9=0x8000;
         r2=0;
       call init_timer(db);
         r7=0;
         dm(Interval)=r7;
       dm(0x0001)=r7;

       bit set MODE2 0x20;      { enable timer }
       bit set MODE1 0x1000;    { enable interrupts }

intwt:  idle;
       jump intwt;

Sample: r0=dm(0x0001);
       r0=ASHIFT r0 by 2;
       r9=0x8000;
       r0=r0 XOR r9;

       r1=BIAS;
       r1=r1 XOR r9;

       dm(DAFifo)=r0;       {send z-piezo sig-initial 0V}
       r12=pm(ADFifo);        {read tunneling current-initial 0V}
       dm(DAFifo)=r1;       {send bias voltage-initial 50mV}

       bit set ASTAT 0x200000;  { toggle flag2 to start next conv }

       r10=pm(ANGBStatus);
       r8=0x3;
       r10=r10 AND r8;
       if ne jump error1;

       bit clr ASTAT 0x200000;
       nop; nop; nop;

       r2=fext r12 by 0:14(SE);   {sign extension of a/d result!!!}

       r3=0x186;        {offset vol. compensation}
       r2=r2+r3;

       r3=0x666;    {TC : 1V }
       r4=r3-r2;    {if the TC is above 1V}
```

```
        if le jump end1; {if TC is generated go to the end}

        r4=dm(0x0001);  {z-piezo voltage}
        r5=0x1eb;       /*0x7ae(1/2") maximum z-piezo vol : "1.2V"(safety factor 0.8nm)
*/
        r5=r4-r5;   {checking if z-piezo vol is over the range }
        if ge jump repeat1;

        r7=dm(Interval);
        r7=r7-1;
        if eq jump repeat1; { wait till the D/A does its job. }

        r7=dm(Interval);
        r13=2;
        r7=r7-r13;
        if eq jump repeat2; { if the z-piezo went back to 0 then  move }
                        {  the inchworm motor. }

        r0=r4+1;      { move the z-piezo by one step.}
        dm(0x0001)=r0;      { increase the counter. }

        r5=0xf6;
        r5=r4-r5;    { checking if z-piezo vol. is over half the range}
        if gt jump clk0;    { jump to the clock 0 }

        r6=0x1;          { the value of the clock : 1 }
        dm(DigitalIO)=r6; { send the clock through Digital I/O }
        rti;

clk0:   r6=0x0;          { the value of the clock : 0 }
        dm(DigitalIO)=r6;{ send the clock through Digital I/O }
        rti;

repeat1:r0=0;          { z-piezo to the original pt. }
        dm(0x0001)=r0;
        r7=dm(Interval);
        r7=r7+1;
        dm(Interval)=r7;{ the status is increased by one. }
        rti;

repeat2:r11=0x1;        { clk0 -> clk1 = move the inchworm motor }
        dm(DigitalIO)=r11;
        r7=0;
        dm(Interval)=r7;
        rti;

init_analog:    bit set MODE2 0x20000;  { flag 2 is an output }
                bit clr ASTAT 0x200000; { set flag2 to low }
                r4=0x0;
                dm(ControlReg)=r4;      {reset analog board}
                nop;nop;nop;
                r4=0x11;                { 1 a/d channels & 2 d/a channels }
                dm(ChanNum)=r4;      { number of channels }
                nop; nop; nop;
                r4=0x80;                { release ang reset & go mode=0 (go on flag2) }
                dm(ControlReg)=r4;      { control register }
                nop; nop; nop;

{ read the first a/d and write the initial d/a outputs }
        wt1:    r10=pm(ANGBStatus);      {wait for the fifo to finish its work.}
                r8=0x3;
                r10=r10 AND r8;
                if ne jump wt1;
```

```
            r0=0x0;
            r1=BIAS;

            r0=r0 XOR r9;
            r1=r1 XOR r9;
            dm(DAFifo)=r0;      {send z-piezo sig-initial 0V}
            dm(DAFifo)=r1;      {apply bias voltage-initial 50mV}

            bit set ASTAT 0x200000;  { toggle flag2 to start first conv }
            nop; nop; nop; nop;
            bit clr ASTAT 0x200000;
            nop; nop; nop;

      wt2:   r10=pm(ANGBStatus);        { wait for the first a/d to trash the }
            r8=0x3;                   { whatever sits in the fifo with new data.}
            r10=r10 AND r8;
            if ne jump wt2;

            rts;

init_timer:    bit clr MODE1 0x1000;   { disable all the previous interrupts }
            bit set irptl 0;        { clear pending interrupt }
            bit set IMASK 0x12;     { enable timer int.   }

            r10=dm(0x0000);  { sample period will be in dm(0x0000) }
            TPERIOD=r10;     { set up the timer }
            TCOUNT=r10;

            rts;

error1: r4=0x23;           { send sampling rate too fast error to status }
      dm(StatusReg)=r4;
      bit clr MODE2 0x20;
      bit clr MODE1 0x1000;
      jump endofprog;

end1:   bit clr MODE2 0x20;     {disable timer}
      bit clr MODE1 0x1000;   {disable ints}
      adflush:        r0=pm(ANGBStatus); {make sure last conversion is done}
                  r8=0x3;
                  r0=r0 AND r8;
                  if ne jump adflush;
                  r0=pm(ADFifo); {flush 1 a/d result from fifo}
      r4=0x0;
      dm(StatusReg)=r4; { let pc know the coarse approach has finished}
endofprog:      idle;
.ENDSEG;

.SEGMENT/PM  tmzh_svc;
jump Sample;
idle;
.ENDSEG;



/*****************************************************************

      pidcon.asm--This program sets initial gap distance by using the PID feedback
                  scheme.
      Channel usage:
            D/A channel 1 - z-piezo       ------ r0
                channel 2 - bias voltage  ------ r1
            A/D channel 1 - tunneling current -- r2
```

```
       The memory usage:
              0x0000 ----- user defined sampling period.
              0x0006 ----- user defined ref. vol.(floating pt.)
              0x0008 ----- user defined coef. - a(floating pt.)
              0x0009 ----- user defined coef. - b(floating pt.)
              0x000b ----- user defined coef. - c(floating pt.)
              0x0001 ----- z piezo control signal update.

       The reserved register:( Refer to chapter 5 for what a, b, and c mean. )
              f5 ---- coefficient a
              f6 ---- coefficient b
              f7 ---- coefficient c
              r9 ---- constant 0x8000
              r1 ---- bias voltage

                     written by Jungmok Bae          1/15/94
                     revised                         1/24/94
                     Final revision                  3/11/94
*****************************************************************/
#define BIAS    0x156d   /* bias voltage 4.5V */

.segment /pm    pm_port1;
.var   ADFifo;
.endseg;

.segment /pm    pm_port2;
.var   ANGBStatus;
.endseg;

.segment /dm    dm_port1;
.var   StatusReg;
.endseg;

.segment /dm    dm_port2;
.var    DigitalIO;
.endseg;

.segment /dm    dm_port3;
.var   DAFifo;
.endseg;

.segment /dm    dm_port4;
.var   ControlReg;
.endseg;

.segment /dm    dm_port5;
.var   ChanNum;
.endseg;

.segment /dm    dm_data;
.var    IntegSum, ConLoop, PreError;      /*sum for the integral gain.S(k-1)(floating
pt )*/
.endseg;

.SEGMENT/PM             rst_svc; { program starts at the reset vector }
      PMWAIT=0x00A1; {/pms0=0ws, /pms1=1ws}
      DMWAIT=0x9421; {/dms0=0ws, /dms1=0ws, /dms2=1ws, /dms3=0ws}

         jump start;
.ENDSEG;

.segment/pm             pm_code;

start:  r4=0x1;         { let pc know the scanning has started}
```

```
        dm(StatusReg)=r4;

        call init_analog(db);
          r2=0;
          r9=0x8000;
        call init_timer(db);
          dm(ConLoop)=r2;
          f3=0;
        dm(IntegSum)=f3;
        dm(PreError)=f3;
        r10=0;
        f5=dm(0x0008);          {read coeff.    a}
        f6=dm(0x0009);          {read coeff.    b}
        f7=dm(0x000b);          {read coeff.    c}

        bit set MODE2 0x20;     { enable timer }
        bit set MODE1 0x1000;   { enable interrupts }

intwt:  idle;
        jump intwt;               { Check if the control loop counter has }
                            { been expired. }

Sample: dm(DAFifo)=r0;       {send z-piezo sig-previous vol.}
        r2=pm(ADFifo);        {read tunneling current}
        dm(DAFifo)=r1;       {send bias voltage-initial 50mV}

        bit set ASTAT 0x200000;  { toggle flag2 to start next conv }
        r3=pm(ANGBStatus);   {checking the status}
        r4=0x3;
        r3=r4 AND r3;
        if ne jump error1;
        bit clr ASTAT 0x200000;
        nop; nop; nop;

        r3=dm(ConLoop);             {control loop counter}
        r4=0x7530;                   {maximum loop counter : 300000x7530}
        r4=r4-r3;
        if eq jump end1;          { the loop counter is expired }
        r3=r3+1;
        dm(ConLoop)=r3;             {increase the loop counter by one}

{floating point calculation}
        r2=fext r2 by 0:14(SE);        {sign extension of a/d result}

        r3=0x186;                      {offset compensation}
        r2=r2+r3;

        r3=0x2000;                      {log table lookup}
        r3=r2+r3;
        i0=r3;
        m0=0;
        r2=dm(m0,i0);                    {read the log table}

        f10=float r2;                {convert to floating point format}

        f11=dm(0x0006);          {user defined ref. vol.}
        f10=f10-f11;                  {error; tuneling cur. - refv. }
        f12=dm(PreError);             {previous error e(k-1) }
        f13=dm(IntegSum);             {sum for integral gain S(k-1)}

{a*e(k) + b*e(k-1) + S(k-1)}
        f9=f5*f10;
        f3=f6*f12;
        f8=f9+f3;
```

```
        f8=f8+f13;

{update the e(k-1) and S(k-1) }
        dm(PreError)=f10;              { e(k-1)=error }
        f3=f7*f10;
        f13=f13+f3;
        dm(IntegSum)=f13;              {S(k-1)=c*error+S(k-2)}

        r3=fix f8;                   { convert to fixed point value }

        r0=dm(0x0001);               {z piezo voltge}
        r0=r0+r3;                    {adjust the z piezo vol. }
        dm(0x0001)=r0;               {update the z piezo vol. }

        r4=0x1fff;                 { over 5V? }
        r3=r0-r4;                  { difference bet. zpiezo and ref.}
        if ge r0=r4;

        r4=PASS r4;
        if lt call VolLim;

        r0=ASHIFT r0 by 2;         {from 14 bit resolution to 16 bit}
        r9=0x8000;
        r0=r0 XOR r9;              {and invert MSB}

        rti;

VolLim: r4=0x4000;                  { Is PID out of control?}
        r4=r0 AND r4;
        if eq jump error1;

        r4=0x2000;                 { under -5V? }
        r3=0xe001;
        r4=r0 AND r4;
        if eq r0=r3;
        rts;

init_analog:    bit set MODE2 0x20000;  { flag 2 is an output }
                bit clr ASTAT 0x200000; { set flag2 to low }
                r4=0x0;
                dm(ControlReg)=r4;      {reset analog board}
                nop;nop;nop;
                r4=0x11;                { 1 a/d channels & 2 d/a channels }
                dm(ChanNum)=r4;       { number of channels }
                nop; nop; nop;
                r4=0x80;                { release ang reset & go mode=0 (go on flag2) }
                dm(ControlReg)=r4;      { control register }
                nop; nop; nop;

{ read the first a/d and write the initial d/a outputs }

        wt1:    r10=pm(ANGBStatus);      {wait for the fifo to finish its work.}
                r8=0x3;
                r10=r10 AND r8;
                if ne jump wt1;

                r0=dm(0x0001);      {dm(0x0001) carries 14 bit information}
                r0=ASHIFT r0 by 2;  { shift up by 2 bits:14 bit data from fine.asm }
                r1=BIAS;

                r9=0x8000;
                r0=r0 XOR r9;       { and invert MSB      }
                r1=r1 XOR r9;
                dm(DAFifo)=r0;      {send z-piezo sig-initial 0V}
```

```
            dm(DAFifo)=r1;      {apply bias voltage-initial 50mV}

            bit set ASTAT 0x200000;  { toggle flag2 to start first conv }
            nop; nop; nop; nop;
            bit clr ASTAT 0x200000;
            nop; nop; nop;

      wt2:   r10=pm(ANGBStatus);        { wait for the first a/d to trash the }
            r8=0x3;                   { whatever sits in the fifo with new data.}
            r10=r10 AND r8;
            if ne jump wt2;

            rts;

init_timer:    bit clr MODE1 0x1000;   { disable all the previous interrupts }
            bit set irptl 0;         { clear pending interrupt }
            bit set IMASK 0x12;      { enable timer int.   }

            r10=dm(0x0000);  { sample period will be in dm(0x0000) }
            TPERIOD=r10;     { set up the timer }
            TCOUNT=r10;

            rts;

error1: bit clr MODE2 0x20;     {disable timer}
      bit clr MODE1 0x1000;   {disable ints}
      r4=0x23;
      dm(StatusReg)=r4; { let pc know the error occured}
      adflush0:   r4=pm(ANGBStatus); {make sure last conversion is done}
            r8=0x3;
            r0=r0 AND r8;
            if ne jump adflush0;
            r0=pm(ADFifo); {flush 1 a/d result from fifo}
      jump endofprog;

end1:   bit clr MODE2 0x20;     {disable timer}
      bit clr MODE1 0x1000;   {disable ints}
      r4=0x0;
      dm(StatusReg)=r4;       { let pc know the pid control is finished}
      adflush:   r4=pm(ANGBStatus); {make sure last conversion is done}
            r8=0x3;
            r0=r0 AND r8;
            if ne jump adflush;
            r0=pm(ADFifo);          {flush 1 a/d result from fifo}
endofprog:   idle;
.ENDSEG;

.SEGMENT/PM  tmzh_svc;
jump Sample;
idle;
.ENDSEG;



/*****************************************************************

      scan1.asm - This module performs the constant height scanning of the given
                  sample surface. The scanning parameters are controllable setting
                  diffrent value of Macro variables. During this module the tip
                  follows the ramp path in 2-D sample plane.
      Channel usage:
            D/A channel 1 - z-piezo       ------ r0
                channel 2 - bias voltage  ------ r1
                channel 3 - x-piezo       ------ r2
```

```
                channel 4 - y-piezo       ------ r3
              A/D channel 1 - tunneling current -- r12 -> r4

       The memory usage:
              0x0000 ----- sampling period.
              0x100 ----- data will be stored here 64*64 4k bytes
       The reserved register:
              r0      z-piezo vol.
              r1      bias vol.
              r2      x-piezo vol.
              r3      y-piezo vol.
              i6      address for the data of the atomic image
              m5      modifier register

                    written by Jungmok Bae          1/7/94
                    Final revision                  3/11/94
*********************************************************************/
#define BIAS    0x156d
#define XSTEP   0x3        {controls the x direction step size}
#define YSTEP   0x3        {controls the y direction step size}
#define XRNG  0x3f         {controls the x direction number of data points}
#define YRNG  0x40         {controls the y direction number of data points}

.segment /pm    pm_port1;
.var   ADFifo;
.endseg;

.segment /pm    pm_port2;
.var   ANGBStatus;
.endseg;

.segment /dm    dm_port1;
.var   StatusReg;
.endseg;

.segment /dm    dm_port2;
.var    DigitalIO;
.endseg;

.segment /dm    dm_port3;
.var   DAFifo;
.endseg;

.segment /dm    dm_port4;
.var   ControlReg;
.endseg;

.segment /dm    dm_port5;
.var   ChanNum;
.endseg;

.segment /dm    dm_data;
.var    Flag, XPiezoVol, YPiezoVol, XCounter, YCounter;
.endseg;
/* xaxis counter, x-dir piezo vol, y-dir piezo vol, y-axis counter, flag */

.SEGMENT/PM             rst_svc; { program starts at the reset vector }
       PMWAIT=0x00A1; {/pms0=0ws, /pms1=1ws}
       DMWAIT=0x9421; {/dms0=0ws, /dms1=0ws, /dms2=1ws, /dms3=0ws}

          {    DMBANK1=0x20000000--status & timing }
          {    DMBANK2=0x40000000--analog board    }
          {    DMBANK3=0x80000000--not used        }
          jump start;
```

```
.ENDSEG;

.segment/pm             pm_code;
start:
       bit set MODE2 0x20000;  { flag 2 is an output }
       bit clr ASTAT 0x200000; { set flag2 to low }
       bit clr MODE1 0x1000;   { disable all the previous interrupts }
       bit set irptl 0;        { clear pending interrupt }
       bit set IMASK 0x12;     { enable timer int.   }

lp:     r4=0x1;          { let pc know the scanning has started}
       dm(StatusReg)=r4;

       call init_analog(db);
         i0=0x100;          {data goes to dm starting at 0x100}
         m0=1;
       call init_timer(db);
         r7=0;
         dm(XCounter)=r7;
       dm(XPiezoVol)=r7;
       dm(YPiezoVol)=r7;
       dm(YCounter)=r7;
       dm(Flag)=r7;
       dm(0x0002)=r7;

       bit set MODE2 0x20;     { enable timer }
       bit set MODE1 0x1000;   { enable interrupts }

intwt:  idle;
       jump intwt;               {Wait for the interrupt }

Sample: dm(DAFifo)=r0;      {send z-piezo sig-initial previous position }
       r12=pm(ADFifo);        {???read tunneling current-initial 0V}
       dm(DAFifo)=r1;      {send bias voltage-initial 50mV}
       dm(DAFifo)=r2;      {send x-piezo vol.-initial 0V}
       dm(DAFifo)=r3;      {send y-piezo vol.-initial 0V}

       bit set ASTAT 0x200000;  { toggle flag2 to start next conv }
       r10=pm(ANGBStatus);   {checking the status}
       r8=0x3;
       r10=r10 AND r8;
       if ne jump tofast;
       bit clr ASTAT 0x200000;
       nop; nop; nop;

       r13=dm(Flag);
       r14=1;
       r13=r13-r14;
       if eq jump dtoa;

atod:   r13=1;                   {set the flag to 0 }
       dm(Flag)=r13;
       r4=fext r12 by 0:14(SE);   {sign extension of a/d result!!!}
                                  {sign conversion}
       r5=0x186;
       r4=r4+r5;               {offset vol. compensation}

       dm(i0,m0)=r4;{ Store TC result to specific memory place. }
       jump lpend;

dtoa:   r13=0;          {set the flag to 1 }
       dm(Flag)=r13;
       r11=dm(YCounter);  { previous y-counter }
       r5=YRNG;        { Maximum 64 points!!! on y-axis }
```

```
        r5=r5-r11;        { Checking if y-dist.is over the range }
        if eq jump end;{ The scanning is done. }

        r11=dm(XCounter);  {previous x-counter}
        r5=XRNG;        { maximum 64 points!!! on x-axis }
        r5=r5-r11;   {checking if x-dist. is over the range }
        if eq jump movey;

movex:  r11=XSTEP;          {x-step:1.4 angstrom}
        r2=dm(XPiezoVol);  {present x-position}
        r2=r2+r11;        {increase the x-piezo by one step vol. }
        dm(XPiezoVol)=r2;  {store new x-piezo vol.}
        r9=0x8000;
        r2=r2 XOR r9;

        r11=dm(XCounter);
        r11=r11+1;         {increase the x-counter}
        dm(XCounter)=r11;

        jump lpend;

movey:  r2=0;            { x-piezo -> 0V }
        dm(XPiezoVol)=r2;  {set x position to zero again}
        dm(XCounter)=r2;  {set x counter to zero again}
        r9=0x8000;
        r2=r2 XOR r9;

        r11=YSTEP;          {y-step:1.4 angstrom}
        r3=dm(YPiezoVol);  {present y-position}
        r3=r3+r11;        {increase the y-dir by one step}
        dm(YPiezoVol)=r3;
        r9=0x8000;
        r3=r3 XOR r9;

        r11=dm(YCounter);  {previous y counter.}
        r11=r11+1;         {increase the y counter by one.}
        dm(YCounter)=r11;

        r1=BIAS;               { 100mV bias voltage--->50mV;done  }
        r9=0x8000;
        r1=r1 XOR r9;

lpend:  rti;

init_analog:    r4=0x0;
              dm(ControlReg)=r4;       {reset analog board}
              nop;nop;nop;
              r4=0x21;                 { 1 a/d channels & 4 d/a channels }
              dm(ChanNum)=r4;       { number of channels }
              nop; nop; nop;
              r4=0x80;                 { release ang reset & go mode=0 (go on flag2) }
              dm(ControlReg)=r4;       { control register }
              nop; nop; nop;

{ read the first a/d and write the initial d/a outputs }

        wt1:    r10=pm(ANGBStatus);      {wait for the fifo to finish its work.}
              r8=0x3;
              r10=r10 AND r8;
              if ne jump wt1;

              r0=dm(0x0001);           {previous z-vol. in 14bit from pidcon.asm}
              r0=ASHIFT r0 by 2;  { shift up by 2 bits because D/A is 16 bits }
              r1=BIAS;
```

```
              r2=0;                {x offset : 0}
              r3=0;                {y offset : 0}

              r9=0x8000;
              r0=r0 XOR r9;        { and invert MSB:due to binary offset format}
              r1=r1 XOR r9;
              r2=r2 XOR r9;
              r3=r3 XOR r9;

              dm(DAFifo)=r0;       {send z-piezo sig-initial 0V}
              dm(DAFifo)=r1;       {apply bias voltage-initial 50mV}
              dm(DAFifo)=r2;       {apply x-piezo vol.}
              dm(DAFifo)=r3;       {apply y-piezo vol.}

              bit set ASTAT 0x200000;  { toggle flag2 to start first conv }
              nop; nop; nop; nop;
              bit clr ASTAT 0x200000;
              nop; nop; nop;

       wt2:    r10=pm(ANGBStatus);        { wait for the first a/d to trash the }
              r8=0x3;                  { whatever sits in the fifo with new data.}
              r10=r10 AND r8;
              if ne jump wt2;

              rts;


init_timer:     r10=dm(0x0000);  { sample period will be in dm(0x0000) }
              TPERIOD=r10;     { set up the timer }
              TCOUNT=r10;

              rts;

tofast: r4=0x23;           { send error to status }
       dm(StatusReg)=r4;
       jump endofprog;

end:   bit clr MODE2 0x20;     {disable timer}
       bit clr MODE1 0x1000;   {disable ints}
{ pull up the z-piezo to 0V position }
wt3:    r10=pm(ANGBStatus);
       r8=0x3;
       r10=r10 AND r8;
       if ne jump wt3;

       r0=1;                    {set the tip position to (0,0,0)}
       r1=BIAS;                {16 bit-> 5V=32763, 50mV=328=0x148 }
       r9=0x8000;
       r0=r0 XOR r9;        { and invert MSB      }
       r1=r1 XOR r9;
       dm(DAFifo)=r0;       {send z-piezo sig-initial 0V}
       r12=pm(ADFifo);         {read tunneling current}
       dm(DAFifo)=r1;       {send bias voltage-initial 50mV}
       dm(DAFifo)=r0;       {send x-piezo vol.-initial 0V}
       dm(DAFifo)=r0;       {send y-piezo vol.-initial 0V}

       bit set ASTAT 0x200000;  { toggle flag2 to start next conv }
       nop; nop; nop; nop;
       bit clr ASTAT 0x200000;
       nop; nop; nop;

       r4=0x0;
       dm(StatusReg)=r4; { let pc know the scanning has finished}
       adflush:   r4=pm(ANGBStatus); {make sure last conversion is done}
```

```
                r8=0x3;
                r4=r4 AND r8;
                if ne jump adflush;
                r4=pm(ADFifo); {flush 1 a/d result from fifo}

endofprog:          idle;
.ENDSEG;

.SEGMENT/PM  tmzh_svc;
jump Sample;
idle;
.ENDSEG;



/*********************************************************************

      scan2.asm - This module performs the constant height mode scanning in
            triangular pattern raster voltage.

      Channel usage:
            D/A channel 1 - z-piezo        ------ r0
                channel 2 - bias voltage   ------ r1
                channel 3 - x-piezo        ------ r2
                channel 4 - y-piezo        ------ r3
            A/D channel 1 - tunneling current -- r12 -> r4

      The memory usage:
            0x0000 ----- sampling period.
            0x100 ----- data will be stored here 64*64 4k bytes
      The reserved register:
            r0      z-piezo vol.
            r1      bias vol.
            r2      x-piezo vol.
            r3      y-piezo vol.
            i6      address for the data of the atomic image
            m5      modifier register

      Note: The shape of the x raster is not ramp but triangular. Such scheme
      provides the continuous motion of the tip and may reduce the
      drift problem.
                                        March 1, 1994
                                        written by Jungmok Bae
*********************************************************************/
#define BIAS    0xf5c
#define XSTEP   0x2
#define YSTEP   0x2
#define XRNG  0x3f
#define YRNG  0x40

.segment /pm    pm_port1;
.var   ADFifo;
.endseg;

.segment /pm    pm_port2;
.var   ANGBStatus;
.endseg;

.segment /dm    dm_port1;
.var   StatusReg;
.endseg;

.segment /dm    dm_port2;
.var    DigitalIO;
```

```
.endseg;

.segment /dm    dm_port3;
.var   DAFifo;
.endseg;

.segment /dm    dm_port4;
.var   ControlReg;
.endseg;

.segment /dm    dm_port5;
.var   ChanNum;
.endseg;

.segment /dm    dm_data;
.var    Flag, XPiezoVol, YPiezoVol, XCounter, YCounter, XStep;
.endseg;
/* xaxis counter, x-dir piezo vol, y-dir piezo vol, y-axis counter, flag */

.SEGMENT/PM             rst_svc; { program starts at the reset vector }
      PMWAIT=0x00A1; {/pms0=0ws, /pms1=1ws}
      DMWAIT=0x9421; {/dms0=0ws, /dms1=0ws, /dms2=1ws, /dms3=0ws}

         jump start;
.ENDSEG;

.segment/pm             pm_code;
start:
      bit set MODE2 0x20000;  { flag 2 is an output }
      bit clr ASTAT 0x200000; { set flag2 to low }
      bit clr MODE1 0x1000;   { disable all the previous interrupts }
      bit set irptl 0;        { clear pending interrupt }
      bit set IMASK 0x12;     { enable timer int.   }

lp:     r4=0x1;         { let pc know the scanning has started}
      dm(StatusReg)=r4;

      call init_analog(db);
        i0=0x100;          {data goes to dm starting at 0x100}
        m0=1;
      call init_timer(db);
        r7=0;
        dm(XCounter)=r7;
      dm(XPiezoVol)=r7;
      dm(YPiezoVol)=r7;
      dm(YCounter)=r7;
      dm(Flag)=r7;
      r7=XSTEP;
      dm(XStep)=r7;

      bit set MODE2 0x20;     { enable timer }
      bit set MODE1 0x1000;   { enable interrupts }

intwt:  idle;
      jump intwt;               {Wait for the interrupt }

Sample: dm(DAFifo)=r0;      {send z-piezo sig-initial previous position }
      r12=pm(ADFifo);         {???read tunneling current-initial 0V}
      dm(DAFifo)=r1;       {send bias voltage-initial 50mV}
      dm(DAFifo)=r2;       {send x-piezo vol.-initial 0V}
      dm(DAFifo)=r3;       {send y-piezo vol.-initial 0V}

      bit set ASTAT 0x200000;  { toggle flag2 to start next conv }
      r10=pm(ANGBStatus);   {checking the status}
```

```
        r8=0x3;
        r10=r10 AND r8;
        if ne jump tofast;
        bit clr ASTAT 0x200000;
        nop; nop; nop;

        r13=dm(Flag);
        r14=1;
        r13=r13-r14;
        if eq jump dtoa;

atod:   r13=1;                    {set the flag to 0 }
        dm(Flag)=r13;
        r4=fext r12 by 0:14(SE);    {sign extension of a/d result!!!}
                                    {sign conversion}
        r5=0x186;
        r4=r4+r5;                {offset vol. compensation}

        dm(i0,m0)=r4;{ Store TC result to specific memory place. }
        jump lpend;

dtoa:   r13=0;           {set the flag to 1 }
        dm(Flag)=r13;
        r11=dm(YCounter);  { previous y-counter }
        r5=YRNG;         { Maximum 64 points!!! on y-axis }
        r5=r5-r11;        { Checking if y-dist.is over the range }
        if eq jump end;{ The scanning is done. }

        r11=dm(XCounter);  {previous x-counter}
        r5=XRNG;        { maximum 64 points!!! on x-axis }
        r5=r5-r11;    {checking if x-dist. is over the range }
        if eq jump movey;

movex:  r11=dm(XStep);          {x-step:1.4 angstrom}
        r2=dm(XPiezoVol);  {present x-position}
        r2=r2+r11;        {increase the x-piezo by one step vol. }
        dm(XPiezoVol)=r2;  {store new x-piezo vol.}
        r9=0x8000;
        r2=r2 XOR r9;

        r11=dm(XCounter);
        r11=r11+1;         {increase the x-counter}
        dm(XCounter)=r11;

        jump lpend;

movey:  r9=dm(XStep);
        r8=-r9;
        dm(XStep)=r8;
        r9=0x0;
        dm(XCounter)=r9;  {set x counter to zero again}

        r11=YSTEP;          {y-step:1.4 angstrom}
        r3=dm(YPiezoVol);  {present y-position}
        r3=r3+r11;        {increase the y-dir by one step}
        dm(YPiezoVol)=r3;
        r9=0x8000;
        r3=r3 XOR r9;

        r11=dm(YCounter);  {previous y counter.}
        r11=r11+1;         {increase the y counter by one.}
        dm(YCounter)=r11;

        r1=BIAS;               { 100mV bias voltage--->50mV;done  }
```

```
        r9=0x611;               {d/a offset compensation}
        r1=r1+r9;
        r9=0x8000;
        r1=r1 XOR r9;

lpend:  rti;

init_analog:    r4=0x0;
                dm(ControlReg)=r4;      {reset analog board}
                nop;nop;nop;
                r4=0x21;                { 1 a/d channels & 4 d/a channels }
                dm(ChanNum)=r4;       { number of channels }
                nop; nop; nop;
                r4=0x80;                { release ang reset & go mode=0 (go on flag2) }
                dm(ControlReg)=r4;      { control register }
                nop; nop; nop;

{ read the first a/d and write the initial d/a outputs }

        wt1:    r10=pm(ANGBStatus);      {wait for the fifo to finish its work.}
                r8=0x3;
                r10=r10 AND r8;
                if ne jump wt1;

                r0=dm(0x0001);          {previous z-vol. in 14bit from pidcon.asm}
                r0=ASHIFT r0 by 2;  { shift up by 2 bits because D/A is 16 bits }
                r1=BIAS;
                r2=0;               {x offset : 0}
                r3=0;               {y offset : 0}
                r9=0x611;
                r1=r1+r9;
                r9=0x8000;
                r0=r0 XOR r9;       { and invert MSB:due to binary offset format}
                r1=r1 XOR r9;
                r2=r2 XOR r9;
                r3=r3 XOR r9;

                dm(DAFifo)=r0;      {send z-piezo sig-initial 0V}
                dm(DAFifo)=r1;      {apply bias voltage-initial 50mV}
                dm(DAFifo)=r2;      {apply x-piezo vol.}
                dm(DAFifo)=r3;      {apply y-piezo vol.}

                bit set ASTAT 0x200000;  { toggle flag2 to start first conv }
                nop; nop; nop; nop;
                bit clr ASTAT 0x200000;
                nop; nop; nop;

        wt2:    r10=pm(ANGBStatus);        { wait for the first a/d to trash the }
                r8=0x3;                  { whatever sits in the fifo with new data.}
                r10=r10 AND r8;
                if ne jump wt2;

                rts;


init_timer:     r10=dm(0x0000);  { sample period will be in dm(0x0000) }
                TPERIOD=r10;     { set up the timer }
                TCOUNT=r10;

                rts;

tofast: r4=0x23;           { send error to status }
        dm(StatusReg)=r4;
        jump endofprog;
```

```
end:    bit clr MODE2 0x20;      {disable timer}
        bit clr MODE1 0x1000;    {disable ints}
{ pull up the z-piezo to 0V position }
wt3:     r10=pm(ANGBStatus);
        r8=0x3;
        r10=r10 AND r8;
        if ne jump wt3;

        r0=1;                     {set the tip position to (0,0,0)}
        r1=BIAS;                 {16 bit-> 5V=32763, 50mV=328=0x148 }
        r9=0x611;                 {d/a offset compensation}
        r1=r1+r9;
        r9=0x8000;
        r0=r0 XOR r9;        { and invert MSB      }
        r1=r1 XOR r9;
        dm(DAFifo)=r0;       {send z-piezo sig-initial 0V}
        r12=pm(ADFifo);         {read tunneling current}
        dm(DAFifo)=r1;       {send bias voltage-initial 50mV}
        dm(DAFifo)=r0;       {send x-piezo vol.-initial 0V}
        dm(DAFifo)=r0;       {send y-piezo vol.-initial 0V}

        bit set ASTAT 0x200000;  { toggle flag2 to start next conv }
        nop; nop; nop; nop;
        bit clr ASTAT 0x200000;
        nop; nop; nop;

        r4=0x0;
        dm(StatusReg)=r4; { let pc know the scanning has finished}
        adflush:   r4=pm(ANGBStatus); {make sure last conversion is done}
                   r8=0x3;
                   r4=r4 AND r8;
                   if ne jump adflush;
                   r4=pm(ADFifo); {flush 1 a/d result from fifo}

endofprog:          idle;
.ENDSEG;

.SEGMENT/PM  tmzh_svc;
jump Sample;
idle;
.ENDSEG;



/*********************************************************************

   ccscan1.asm  Constant current ramp raster mode.

        Channel Usage:
               D/A channel 1 - z-piezo        ------ r0
                   channel 2 - bias voltage   ------ r1
                   channel 3 - x-piezo        ------ r2
                   channel 4 - y-piezo        ------ r3
               A/D channel 1 - tunneling current -- r4

        The memory usage
               0x0000 ----- user defined sampling period.
               0x0006 ----- user defined ref. vol.(floating pt.)
               0x0008 ----- user defined coef. - a(floating pt.)
               0x0009 ----- user defined coef. - b(floating pt.)
               0x000b ----- user defined coef. - c(floating pt.)
               0x0001 ----- z piezo control signal update.
               0x100 -> 0x1100
```

```
                ---- data will be stored here 64*64 4k bytes.
            0x2000 -> 0x4000(program memory)
                ---- reserved for the log table.

       Programmer's Note: Try to use the on-board memory instead of the registers.
                This will prevent the confusion of the register's usage.
                The case is not applicable for the PC inputs.

                written by Jungmok Bae          12/16/93
                revised                         2/17/94
                revised again                   3/4/94
*****************************************************************/
#define         BIAS    0x156d          /* bias voltage 0.6V */
#define         XSTEP   0x2
#define         YSTEP   0x2
#define         XRNG    0x3f
#define         YRNG    0x40
#define         CLOOP   0x32            { sets the number of the PID feedback for
                                              each data sample points }

.segment /pm    pm_port1;
.var    ADFifo;
.endseg;

.segment /pm    pm_port2;
.var    ANGBStatus;
.endseg;

.segment /dm    dm_port1;
.var    StatusReg;
.endseg;

.segment /dm    dm_port2;
.var    DigitalIO;
.endseg;

.segment /dm    dm_port3;
.var    DAFifo;
.endseg;

.segment /dm    dm_port4;
.var    ControlReg;
.endseg;

.segment /dm    dm_port5;
.var    ChanNum;
.endseg;

.segment /dm    dm_data;
.var    IntegSum, ConLoop, PreError, DataAdr, XPiezoVol, YPiezoVol;
.var    XCounter, YCounter, XStep, WaitLoop, WaitFlag,ZPiezoSum;
.endseg;

.SEGMENT/PM             rst_svc; { program starts at the reset vector }
      PMWAIT=0x00A5; {/pms0=0ws, /pms1=1ws}
      DMWAIT=0x9425; {/dms0=0ws, /dms1=0ws, /dms2=1ws, /dms3=0ws}

         jump start;
.ENDSEG;

.segment/pm             pm_code;

start:  bit set MODE2 0x20000;  { flag 2 is an output }
      bit clr ASTAT 0x200000; { set flag2 to low }
```

```
        bit clr MODE1 0x1000;    { disable all the previous interrupts }
        bit set irptl 0;         { clear pending interrupt }
        bit set IMASK 0x12;      { enable timer int.    }

        r4=0x1;
        dm(StatusReg)=r4;

        call init_analog(db);
          r5=0;
          dm(ConLoop)=r5;
        call init_timer(db);
          f4=0;
          dm(IntegSum)=f4;
        dm(PreError)=f4;
        f5=dm(0x0008);
        f6=dm(0x0009);
        f7=dm(0x000b);
        r4=0;
        dm(XCounter)=r4;
        dm(XPiezoVol)=r4;
        dm(YPiezoVol)=r4;
        dm(YCounter)=r4;
        dm(WaitLoop)=r4;
        dm(ZPiezoSum)=r4;
        r4=XSTEP;
        dm(XStep)=r4;
        r4=0x1;
        dm(WaitFlag)=r4;
        m0=1;
        i0=0x100;

        bit set MODE2 0x20;
        bit set MODE1 0x1000;

intwt:  idle;
        jump intwt;

Sample: r5=0x8000;
        r0=dm(0x0001);
        r0=ASHIFT r0 by 2;
        r0=r0 XOR r5;

        r1=BIAS;
        r1=r1 XOR r5;

        r2=dm(XPiezoVol);
        r2=r2 XOR r5;

        r3=dm(YPiezoVol);
        r3=r3 XOR r5;

        dm(DAFifo)=r0;           { z-piezo vol.}
        r4=pm(ADFifo);           { tunneling current signal }
        dm(DAFifo)=r1;           { bias vol. }
        dm(DAFifo)=r2;           { x-piezo vol.}
        dm(DAFifo)=r3;           { y-piezo vol.}

        bit set ASTAT 0x200000;
        r10=pm(ANGBStatus);
        r8=0x3;
        r10=r10 AND r8;
        if ne jump error1;
        bit clr ASTAT 0x200000;
        nop; nop; nop;
```

```
        r10=dm(WaitFlag);
        r8=0x0;
        r8=r8-r10;
        if eq jump wait2;

        r10=dm(ConLoop);          {control loop counter}
        r8=dm(0x0002);                {maximum loop counter : user input}
        r8=r8-r10;
        if eq jump MoveXY;       {set the flag equals to zero}
        r10=r10+1;
        dm(ConLoop)=r10;          {increase the loop counter}

        r4=fext r4 by 0:14(SE);    {sign extension of a/d result!!!}
        r1=0x186;
        r4=r4+r1;
        r1=0x2000;                 {log conversion}
        r1=r4+r1;
        i8=r1;
        m8=0;
        r4=pm(m8,i8);

        f10=float r4;
        f11=dm(0x0006);          {user defined ref. vol.}
        f10=f10-f11;
        f12=dm(PreError);
        f13=dm(IntegSum);
{ a*e(k) + b*e(k-1) + S(k-1) }
        f5=dm(0x0008);
        f6=dm(0x0009);
        f9=f5*f10;
        f0=f6*f12;
        f8=f9+f0;
        f8=f8+f13;
{ update the e(k-1) and S(k-1) }
        f7=dm(0x000b);
        dm(PreError)=f10;
        f0=f7*f10;
        f13=f13+f0;
        dm(IntegSum)=f13;

        r1=fix f8;

        r0=dm(0x0001);
        r0=r0+r1;

        r4=0x1fff;               { over 5V? }
        r3=r0-r4;
        if ge r0=r4;

        r4=PASS r4;              { minus voltage?}
        if lt call limit;

        dm(0x0001)=r0;           {store minus voltage}

        r2=dm(ZPiezoSum);
        r2=r2+r0;
        dm(ZPiezoSum)=r2;

        rti;

limit:  r4=0x4000;               { Is PID out of control?}
        r4=r0 AND r4;
        if eq jump error1;
```

```
        r4=0x2000;              { under -5V? }
        r3=0xe001;
        r4=r0 AND r4;
        if eq r0=r3;
        rts;

MoveXY: r10=0;
        dm(ConLoop)=r10;            {set control loop counter to zero}
        dm(WaitFlag)=r10;
        f10=0;
        dm(PreError)=f10;
        dm(IntegSum)=f10;            {set integral gain sum to zero}

        r3=dm(ZPiezoSum);       { retrieve the zpiezovoltage sum}
        f4=float r3;            { data format conversion}
        f5=dm(0x0003);          { retrieve the recips of #control loop}
        f4=f5*f4;               { take the average}
        r3=fix f4;
        dm(i0,m0)=r3;           { Store TC result to specific memory place. }
        r3=0x0;
        dm(ZPiezoSum)=r3;

        r11=dm(YCounter);  { previous y-counter }
        r8=YRNG;         { Maximum 64 points!!! on y-axis }
        r8=r8-r11;        { Checking if y-dist.is over the range }
        if eq jump end;{ The scanning is done. }

        r11=dm(XCounter);  {previous x-counter}
        r8=XRNG;        { maximum 64 points!!! on x-axis }
        r8=r8-r11;   {checking if x-dist. is over the range }
        if eq jump movey;

movex:  r11=XSTEP;          {x-step:1.4 angstrom}
        r2=dm(XPiezoVol);  {present x-position}
        r2=r2+r11;        {increase the x-piezo by one step vol. }
        dm(XPiezoVol)=r2;  {store new x-piezo vol.}

        r11=dm(XCounter);
        r11=r11+1;         {increase the x-counter}
        dm(XCounter)=r11;

        jump lpend;

movey:  r2=0;            { x-piezo -> 0V }
        dm(XPiezoVol)=r2;  {set x position to zero again}
        dm(XCounter)=r2;  {set x counter to zero again}

        r11=YSTEP;          {y-step:1.4 angstrom}
        r3=dm(YPiezoVol);  {present y-position}
        r3=r3+r11;        {increase the y-dir by one step}
        dm(YPiezoVol)=r3;

        r11=dm(YCounter);  {previous y counter.}
        r11=r11+1;         {increase the y counter by one.}
        dm(YCounter)=r11;

lpend:  rti;

wait2:  r10=dm(WaitLoop);          {control wait loop counter}
        r10=r10+1;
        dm(WaitLoop)=r10;          {increase the wait loop counter}
        r8=0x5;                 {maximum wait loop counter : 100}
        r8=r8-r10;
```

```
        if eq jump waitend;       {set the flag equals to other than zero}
        rti;
waitend:r8=0x0;
        dm(WaitLoop)=r8;
        r8=0x1;
        dm(WaitFlag)=r8;
        rti;

init_analog:    r4=0x0;
                dm(ControlReg)=r4;       {reset analog board}
                nop;nop;nop;
                r4=0x21;                 { 1 a/d channels & 4 d/a channels }
                dm(ChanNum)=r4;       { number of channels }
                nop; nop; nop;
                r4=0x80;                 { release ang reset & go mode=0 (go on flag2) }
                dm(ControlReg)=r4;       { control register }
                nop; nop; nop;

{ read the first a/d and write the initial d/a outputs }

        wt1:    r10=pm(ANGBStatus);      {wait for the fifo to finish its work.}
                r8=0x3;
                r10=r10 AND r8;
                if ne jump wt1;

                r0=dm(0x0001);           {previous z-vol. in 14bit from pidcon.asm}
                r0=ASHIFT r0 by 2;  { shift up by 2 bits because D/A is 16 bits }
                r1=BIAS;
                r2=0;                    {x offset : 0}
                r3=0;                    {y offset : 0}
                r6=0x8000;
                r0=r0 XOR r6;            { and invert MSB:due to binary offset format}
                r1=r1 XOR r6;
                r2=r2 XOR r6;
                r3=r3 XOR r6;

                dm(DAFifo)=r0;           {send z-piezo sig-initial 0V}
                dm(DAFifo)=r1;           {apply bias voltage-initial 50mV}
                dm(DAFifo)=r2;           {apply x-piezo vol.}
                dm(DAFifo)=r3;           {apply y-piezo vol.}

                bit set ASTAT 0x200000;  { toggle flag2 to start first conv }
                nop; nop; nop; nop;
                bit clr ASTAT 0x200000;
                nop; nop; nop;

        wt2:    r10=pm(ANGBStatus);        { wait for the first a/d to trash the }
                r8=0x3;                  { whatever sits in the fifo with new data.}
                r10=r10 AND r8;
                if ne jump wt2;

                rts;


init_timer:     r10=dm(0x0000);  { sample period will be in dm(0x0000) }
                TPERIOD=r10;     { set up the timer }
                TCOUNT=r10;

                rts;

error1: r0=0x23;           { send error to status }
        dm(StatusReg)=r0;
        bit clr MODE2 0x20;
        bit clr MODE1 0x1000;
```

```
        jump endofprog;

end:    bit clr MODE2 0x20;      {disable timer}
        bit clr MODE1 0x1000;    {disable ints}
{ pull up the z-piezo to 0V position }
wt3:     r10=pm(ANGBStatus);
        r8=0x3;
        r10=r10 AND r8;
        if ne jump wt3;

        r0=1;                     {set the tip position to (0,0,0)}
        r1=BIAS;                 {16 bit-> 5V=32763, 50mV=328=0x148 }
        r9=0x8000;
        r0=r0 XOR r9;         { and invert MSB      }
        r1=r1 XOR r9;
        dm(DAFifo)=r0;        {send z-piezo sig-initial 0V}
        r12=pm(ADFifo);          {read tunneling current}
        dm(DAFifo)=r1;        {send bias voltage-initial 50mV}
        dm(DAFifo)=r0;        {send x-piezo vol.-initial 0V}
        dm(DAFifo)=r0;        {send y-piezo vol.-initial 0V}

        bit set ASTAT 0x200000;  { toggle flag2 to start next conv }
        nop; nop; nop; nop;
        bit clr ASTAT 0x200000;
        nop; nop; nop;

        r4=0x0;
        dm(StatusReg)=r4; { let pc know the scanning has finished}
        adflush:   r4=pm(ANGBStatus); {make sure last conversion is done}
                   r8=0x3;
                   r4=r4 AND r8;
                   if ne jump adflush;
                   r4=pm(ADFifo); {flush 1 a/d result from fifo}

endofprog:           idle;
.ENDSEG;

.SEGMENT/PM  tmzh_svc;
jump Sample;
idle;
.ENDSEG;



/*********************************************************************

   ccscan.asm   Constant current triangular raster mode.

        Channel usage:
               D/A channel 1 - z-piezo       ------ r0
                   channel 2 - bias voltage  ------ r1
                   channel 3 - x-piezo       ------ r2
                   channel 4 - y-piezo       ------ r3
               A/D channel 1 - tunneling current -- r4

        The memory usage:
               0x0000 ----- user defined sampling period.
               0x0006 ----- user defined ref. vol.(floating pt.)
               0x0008 ----- user defined coef. - a(floating pt.)
               0x0009 ----- user defined coef. - b(floating pt.)
               0x000b ----- user defined coef. - c(floating pt.)
               0x0001 ----- z piezo control signal update.
               0x0002 ----- the number of the control loop
               0x0003 ----- the reciprocal of above value
```

```
            0x100 -> 0x1100
                  ---- data will be stored here 64*64 4k bytes.
            0x2000 -> 0x4000(program memory)
                  ---- reserved for the log table.


                  written by Jungmok Bae          12/16/93
                  averaging added
                  revised for 1"piezotube         3/9/94
*****************************************************************/
#define         BIAS    0x156d          /* bias voltage 0.6V + offset vol */
#define         XSTEP   0x2
#define         YSTEP   0x2
#define         XRNG    0x3f
#define         YRNG    0x40
#define         CLOOP   0x32

.segment /pm    pm_port1;
.var    ADFifo;
.endseg;

.segment /pm    pm_port2;
.var    ANGBStatus;
.endseg;

.segment /dm    dm_port1;
.var    StatusReg;
.endseg;

.segment /dm    dm_port2;
.var    DigitalIO;
.endseg;

.segment /dm    dm_port3;
.var    DAFifo;
.endseg;

.segment /dm    dm_port4;
.var    ControlReg;
.endseg;

.segment /dm    dm_port5;
.var    ChanNum;
.endseg;

.segment /dm    dm_data;
.var    IntegSum, ConLoop, PreError, DataAdr, XPiezoVol, YPiezoVol;
.var    XCounter, YCounter, XStep, WaitFlag, WaitLoop, ZPiezoSum;
.endseg;

.SEGMENT/PM             rst_svc; { program starts at the reset vector }
      PMWAIT=0x00A5; {/pms0=0ws, /pms1=1ws}
      DMWAIT=0x9425; {/dms0=0ws, /dms1=0ws, /dms2=1ws, /dms3=0ws}

         jump start;
.ENDSEG;

.segment/pm             pm_code;

start:  bit set MODE2 0x20000;  { flag 2 is an output }
      bit clr ASTAT 0x200000; { set flag2 to low }
      bit clr MODE1 0x1000;   { disable all the previous interrupts }
      bit set irptl 0;        { clear pending interrupt }
      bit set IMASK 0x12;     { enable timer int.   }
```

```
        r4=0x1;
        dm(StatusReg)=r4;

        call init_analog(db);
          r5=0;
          dm(ConLoop)=r5;
        call init_timer(db);
          f4=0;
          dm(IntegSum)=f4;
        dm(PreError)=f4;
        f5=dm(0x0008);
        f6=dm(0x0009);
        f7=dm(0x000b);
        r4=0;
        dm(XCounter)=r4;
        dm(XPiezoVol)=r4;
        dm(YPiezoVol)=r4;
        dm(YCounter)=r4;
        dm(WaitLoop)=r4;
        dm(ZPiezoSum)=r4;
        r4=XSTEP;
        dm(XStep)=r4;
        r4=0x1;
        dm(WaitFlag)=r4;
        m0=1;
        i0=0x100;

        bit set MODE2 0x20;
        bit set MODE1 0x1000;

intwt:  idle;
        jump intwt;

Sample: r5=0x8000;
        r0=dm(0x0001);
        r0=ASHIFT r0 by 2;
        r0=r0 XOR r5;

        r1=BIAS;
        r1=r1 XOR r5;

        r2=dm(XPiezoVol);
        r2=r2 XOR r5;

        r3=dm(YPiezoVol);
        r3=r3 XOR r5;

        dm(DAFifo)=r0;          { z-piezo vol.}
        r4=pm(ADFifo);          { tunneling current signal }
        dm(DAFifo)=r1;          { bias vol. }
        dm(DAFifo)=r2;          { x-piezo vol.}
        dm(DAFifo)=r3;          { y-piezo vol.}

        bit set ASTAT 0x200000;
        r10=pm(ANGBStatus);
        r8=0x3;
        r10=r10 AND r8;
        if ne jump error1;
        bit clr ASTAT 0x200000;
        nop; nop; nop;

        r10=dm(WaitFlag);
        r8=0x0;                 /*never equal0x0 - debugging*/
```

```
        r8=r8-r10;
        if eq jump wait2;

        r10=dm(ConLoop);          {control loop counter}
        r8=dm(0x0002);           {user input maximum loop counter}/*dm(0x0002)*/
        r8=r8-r10;
        if eq jump MoveXY;       {set the flag equals to zero}
        r10=r10+1;
        dm(ConLoop)=r10;          {increase the loop counter}

        r4=fext r4 by 0:14(SE);    {sign extension of a/d result!!!}
        r1=0x186;
        r4=r4+r1;                {offset vol. compensation}
        r1=0x2000;                 {log conversion}
        r1=r4+r1;
        i8=r1;
        m8=0;
        r4=pm(m8,i8);

        f10=float r4;
        f11=dm(0x0006);          {user defined ref. vol.}
        f10=f10-f11;
        f12=dm(PreError);
        f13=dm(IntegSum);
{ a*e(k) + b*e(k-1) + S(k-1) }
        f5=dm(0x0008);
        f6=dm(0x0009);
        f9=f5*f10;
        f0=f6*f12;
        f8=f9+f0;
        f8=f8+f13;
{ update the e(k-1) and S(k-1) }
        f7=dm(0x000b);
        dm(PreError)=f10;
        f0=f7*f10;
        f13=f13+f0;
        dm(IntegSum)=f13;

        r1=fix f8;

        r0=dm(0x0001);
        r0=r0+r1;

        r4=0x1fff;               { over 5V? }
        r3=r0-r4;
        if ge r0=r4;

        r4=PASS r4;              { minus voltage?}
        if lt call limit;

        dm(0x0001)=r0;           {store the final value}

        r2=dm(ZPiezoSum);
        r2=r2+r0;
        dm(ZPiezoSum)=r2;

        rti;

limit:  r4=0x4000;               { Is PID out of control?}
        r4=r0 AND r4;
        if eq jump error1;

        r4=0x2000;               { under -5V? }
        r3=0xe001;
```

```
        r4=r0 AND r4;
        if eq r0=r3;
        rts;

MoveXY: r10=0;
        dm(ConLoop)=r10;            {set control loop counter to zero}
        dm(WaitFlag)=r10;
        f10=0;
        dm(PreError)=f10;
        dm(IntegSum)=f10;            {set integral gain sum to zero}

        r3=dm(ZPiezoSum);       { retrieve the zpiezovoltage sum}
        f4=float r3;            { data format conversion}
        f5=dm(0x0003);          { retrieve the recips of #control
loop}/*debuggingdm(0x0003)*/
        f4=f5*f4;               { take the average}
        r3=fix f4;
        dm(i0,m0)=r3;
        r3=0;
        dm(ZPiezoSum)=r3;       {set the sum zero for next averaging}

        r11=dm(YCounter);  { previous y-counter }
        r8=YRNG;         { Maximum 64 points!!! on y-axis }
        r8=r8-r11;        { Checking if y-dist.is over the range }
        if eq jump end;{ The scanning is done. }

        r11=dm(XCounter);  {previous x-counter}
        r8=XRNG;        { maximum 64 points!!! on x-axis }
        r8=r8-r11;   {checking if x-dist. is over the range }
        if eq jump movey;

movex:  r11=dm(XStep);          {x-step:1.4 angstrom}
        r2=dm(XPiezoVol);  {present x-position}
        r2=r2+r11;        {increase the x-piezo by one step vol. }
        dm(XPiezoVol)=r2;  {store new x-piezo vol.}

        r11=dm(XCounter);
        r11=r11+1;         {increase the x-counter}
        dm(XCounter)=r11;

        jump lpend;

movey:  r2=dm(XStep);
        r3=-r2;
        dm(XStep)=r3;
        r2=0;
        dm(XCounter)=r2;  {set x counter to zero again}

        r11=YSTEP;          {y-step:1.4 angstrom}
        r3=dm(YPiezoVol);  {present y-position}
        r3=r3+r11;        {increase the y-dir by one step}
        dm(YPiezoVol)=r3;

        r11=dm(YCounter);  {previous y counter.}
        r11=r11+1;         {increase the y counter by one.}
        dm(YCounter)=r11;

lpend:  rti;

wait2:  r10=dm(WaitLoop);          {control loop counter}
        r10=r10+1;
        dm(WaitLoop)=r10;         {increase the loop counter}
        r8=0x2;                 {maximum loop counter : 100}
        r8=r8-r10;
```

```
       if eq jump waitend;       {set the flag equals to other than zero}
       rti;
waitend:r8=0x0;
       dm(WaitLoop)=r8;
       r8=0x1;
       dm(WaitFlag)=r8;
       rti;

init_analog:    r4=0x0;
              dm(ControlReg)=r4;       {reset analog board}
              nop;nop;nop;
              r4=0x21;                 { 1 a/d channels & 4 d/a channels }
              dm(ChanNum)=r4;       { number of channels }
              nop; nop; nop;
              r4=0x80;                 { release ang reset & go mode=0 (go on flag2) }
              dm(ControlReg)=r4;       { control register }
              nop; nop; nop;

{ read the first a/d and write the initial d/a outputs }

       wt1:    r10=pm(ANGBStatus);       {wait for the fifo to finish its work.}
              r8=0x3;
              r10=r10 AND r8;
              if ne jump wt1;

              r0=dm(0x0001);           {previous z-vol. in 14bit from pidcon.asm}
              r0=ASHIFT r0 by 2;  { shift up by 2 bits because D/A is 16 bits }
              r1=BIAS;
              r2=0;                {x offset : 0}
              r3=0;                {y offset : 0}
              r6=0x8000;
              r0=r0 XOR r6;        { and invert MSB:due to binary offset format}
              r1=r1 XOR r6;
              r2=r2 XOR r6;
              r3=r3 XOR r6;

              dm(DAFifo)=r0;       {send z-piezo sig-initial 0V}
              dm(DAFifo)=r1;       {apply bias voltage-initial 50mV}
              dm(DAFifo)=r2;       {apply x-piezo vol.}
              dm(DAFifo)=r3;       {apply y-piezo vol.}

              bit set ASTAT 0x200000;  { toggle flag2 to start first conv }
              nop; nop; nop; nop;
              bit clr ASTAT 0x200000;
              nop; nop; nop;

       wt2:    r10=pm(ANGBStatus);        { wait for the first a/d to trash the }
              r8=0x3;                  { whatever sits in the fifo with new data.}
              r10=r10 AND r8;
              if ne jump wt2;

              rts;


init_timer:     r10=dm(0x0000);  { sample period will be in dm(0x0000) }
              TPERIOD=r10;     { set up the timer }
              TCOUNT=r10;

              rts;

error1: r0=0x23;           { send error to status }
       dm(StatusReg)=r0;
       bit clr MODE2 0x20;
       bit clr MODE1 0x1000;
```

```
        jump endofprog;

end:     bit clr MODE2 0x20;      {disable timer}
        bit clr MODE1 0x1000;    {disable ints}
{ pull up the z-piezo to 0V position }
wt3:     r10=pm(ANGBStatus);
        r8=0x3;
        r10=r10 AND r8;
        if ne jump wt3;

        r0=1;                     {set the tip position to (0,0,0)}
        r1=BIAS;                 {16 bit-> 5V=32763, 50mV=328=0x148 }
        r9=0x8000;
        r0=r0 XOR r9;        { and invert MSB      }
        r1=r1 XOR r9;
        dm(DAFifo)=r0;       {send z-piezo sig-initial 0V}
        r12=pm(ADFifo);         {read tunneling current}
        dm(DAFifo)=r1;       {send bias voltage-initial 50mV}
        dm(DAFifo)=r0;       {send x-piezo vol.-initial 0V}
        dm(DAFifo)=r0;       {send y-piezo vol.-initial 0V}

        bit set ASTAT 0x200000;  { toggle flag2 to start next conv }
        nop; nop; nop; nop;
        bit clr ASTAT 0x200000;
        nop; nop; nop;

        r4=0x0;
        dm(StatusReg)=r4; { let pc know the scanning has finished}
        adflush:   r4=pm(ANGBStatus); {make sure last conversion is done}
                  r8=0x3;
                  r4=r4 AND r8;
                  if ne jump adflush;
                  r4=pm(ADFifo); {flush 1 a/d result from fifo}

endofprog:          idle;
.ENDSEG;

.SEGMENT/PM  tmzh_svc;
jump Sample;
idle;
.ENDSEG;



/*********************************************************************

        pidtune.asm--This program is for tuning the PID gain.
               The user inputs two sampling frequencies. The one is
               for the pid control and the other is for the sampling
               step response. The sampling rate for the step response is
               usually set higher. The DSP changes the reference voltage,
               measures the tunneling current response and transfers the data
               to PC which displays it in nice graphic form.
        Channel usages:
               D/A channel 1 - z-piezo        ------ r0
                   channel 2 - bias voltage   ------ r1
               A/D channel 1 - tunneling current -- r2

        The memory usage
               0x0000 ----- user defined sampling period.
               0x0006 ----- user defined ref. vol.(floating pt.)
               0x0008 ----- user defined coef. - a(floating pt.)
               0x0009 ----- user defined coef. - b(floating pt.)
               0x000b ----- user defined coef. - c(floating pt.)
```

```
            0x0001 ----- z piezo control signal update.
            0x0002 ----- multiple of sampling rate.
            0x800 -> 0x3fff -- the data goes here.
      The reserved register
            f5 ---- a
            f6 ---- b
            f7 ---- c
            r15 --- 0x8000
            r1 ---- bias

                  written by Jungmok Bae          3/4/94

*********************************************************************/
#define BIAS    0x156d   /* bias voltage 4.5V */

.segment /pm    pm_port1;
.var   ADFifo;
.endseg;

.segment /pm    pm_port2;
.var   ANGBStatus;
.endseg;

.segment /dm    dm_port1;
.var   StatusReg;
.endseg;

.segment /dm    dm_port2;
.var    DigitalIO;
.endseg;

.segment /dm    dm_port3;
.var   DAFifo;
.endseg;

.segment /dm    dm_port4;
.var   ControlReg;
.endseg;

.segment /dm    dm_port5;
.var   ChanNum;
.endseg;

.segment /dm    dm_data;
.var    IntegSum, ConLoop, PreError,TNC,Counter;      /*sum for the integral
gain.S(k-1)(floating pt )*/
.endseg;

.SEGMENT/PM             rst_svc; { program starts at the reset vector }
      PMWAIT=0x00A1; {/pms0=0ws, /pms1=1ws}
      DMWAIT=0x9421; {/dms0=0ws, /dms1=0ws, /dms2=1ws, /dms3=0ws}

         jump start;
.ENDSEG;

.segment/pm             pm_code;

start:  r4=0x1;         { let pc know the scanning has started}
      dm(StatusReg)=r4;

      call init_analog(db);
        r2=0;
        r9=0x8000;
      call init_timer(db);
```

```
        dm(ConLoop)=r2;
        f3=0;
      dm(IntegSum)=f3;
      dm(PreError)=f3;
      r10=0;
      dm(Counter)=r10;
      f5=dm(0x0008);          {read coeff.    a}
      f6=dm(0x0009);          {read coeff.    b}
      f7=dm(0x000b);          {read coeff.    c}
      i8=0x800;
      m8=1;

      bit set MODE2 0x20;     { enable timer }
      bit set MODE1 0x1000;   { enable interrupts }

intwt:  idle;
      jump intwt;               { Check if the control loop counter has }
                          { been expired. }

Sample: dm(DAFifo)=r0;       {send z-piezo sig-previous vol.}
      r2=pm(ADFifo);         {read tunneling current}
      dm(DAFifo)=r1;       {send bias voltage-initial 50mV}

      bit set ASTAT 0x200000;  { toggle flag2 to start next conv }
      r3=pm(ANGBStatus);   {checking the status}
      r4=0x3;
      r3=r4 AND r3;
      if ne jump error;
      bit clr ASTAT 0x200000;
      nop; nop; nop;

      r2=fext r2 by 0:14(SE);        {sign extension of a/d result!!!}
                               {sign conversion}
      r3=0x186;
      r2=r2+r3;                      {offset vol. compensation}
      r3=0x2000;
      r3=r2+r3;                         { convert to log }
      i0=r3;
      m0=0;
      r2=dm(m0,i0);                   {read the log table}

      pm(i8,m8)=r2;                  {store the data in the program memory}
      dm(TNC)=r2;                    {store the data for pid control}

      r3=dm(Counter);
      r3=r3+1;
      dm(Counter)=r3;
      r4=dm(0x0002);
      r4=r3-r4;
      if eq jump PIDloop;

      rti;

PIDloop:r3=dm(ConLoop);             {control loop counter}
      r4=0x1000;                    {maximum loop counter : 0x1000}
      r4=r4-r3;
      if eq jump end1;          { the loop counter is expired }
      r3=r3+1;
      dm(ConLoop)=r3;             {increase the loop counter by one}

{floating point calculation}
      r2=dm(TNC);
      f10=float r2;               {convert to floating point format}
      f11=dm(0x0006);           {user defined ref. vol.}
```

```
        f10=f10-f11;                    {error; tuneling cur. - refv. }
        f12=dm(PreError);               {previous error e(k-1) }
        f13=dm(IntegSum);               {sum for integral gain S(k-1)}
{a*e(k) + b*e(k-1) + S(k-1)}
        f9=f5*f10;
        f3=f6*f12;
        f8=f9+f3;
        f8=f8+f13;
{update the e(k-1) and S(k-1) }
        dm(PreError)=f10;               { e(k-1)=error }
        f3=f7*f10;
        f13=f13+f3;
        dm(IntegSum)=f13;               {S(k-1)=c*error+S(k-2)}

        r3=fix f8;                  { convert to fixed point value }

        r0=dm(0x0001);              {z piezo voltge}
        r0=r0+r3;                   {adjust the z piezo vol. }
        dm(0x0001)=r0;              {update the z piezo vol. }

        r0=ASHIFT r0 by 2;          {from 14 bits to 16 bits!!! }
        r9=0x8000;
        r0=r0 XOR r9;       { and invert MSB      }

        r3=0x0;
        dm(Counter)=r3;

        rti;

init_analog:    bit set MODE2 0x20000;  { flag 2 is an output }
              bit clr ASTAT 0x200000; { set flag2 to low }
              r4=0x0;
              dm(ControlReg)=r4;        {reset analog board}
              nop;nop;nop;
              r4=0x11;                  { 1 a/d channels & 2 d/a channels }
              dm(ChanNum)=r4;       { number of channels }
              nop; nop; nop;
              r4=0x80;                  { release ang reset & go mode=0 (go on flag2) }
              dm(ControlReg)=r4;        { control register }
              nop; nop; nop;

{ read the first a/d and write the initial d/a outputs }

        wt1:    r10=pm(ANGBStatus);       {wait for the fifo to finish its work.}
              r8=0x3;
              r10=r10 AND r8;
              if ne jump wt1;

              r0=dm(0x0001);        {dm(0x0001) carries 14 bit information}
              r0=ASHIFT r0 by 2;    { shift up by 2 bits:14 bit data from fine.asm }
              r1=BIAS;
              r0=r0 XOR r9;         { and invert MSB      }
              r1=r1 XOR r9;
              dm(DAFifo)=r0;        {send z-piezo sig-initial 0V}
              dm(DAFifo)=r1;        {apply bias voltage-initial 50mV}

              bit set ASTAT 0x200000;  { toggle flag2 to start first conv }
              nop; nop; nop; nop;
              bit clr ASTAT 0x200000;
              nop; nop; nop;

        wt2:    r10=pm(ANGBStatus);         { wait for the first a/d to trash the }
              r8=0x3;                   { whatever sits in the fifo with new data.}
              r10=r10 AND r8;
```

```
                if ne jump wt2;

                rts;

init_timer:     bit clr MODE1 0x1000;   { disable all the previous interrupts }
                bit set irptl 0;        { clear pending interrupt }
                bit set IMASK 0x12;     { enable timer int.   }

                r10=dm(0x0000);  { sample period will be in dm(0x0000) }
                TPERIOD=r10;     { set up the timer }
                TCOUNT=r10;

                rts;

error:  r4=0x23;            { send error to status }
        dm(StatusReg)=r4;
        jump endofprog;
end1:    bit clr MODE2 0x20;      {disable timer}
        bit clr MODE1 0x1000;    {disable ints}
        r4=0x0;
        dm(StatusReg)=r4; { let pc know the scanning has finished}
        adflush:   r4=pm(ANGBStatus); {make sure last conversion is done}
                r8=0x3;
                r0=r0 AND r8;
                if ne jump adflush;
                r0=pm(ADFifo); {flush 1 a/d result from fifo}
endofprog:    idle;
.ENDSEG;

.SEGMENT/PM  tmzh_svc;
jump Sample;
idle;
.ENDSEG;
```

Appendix C

PC Code

C.1 Header File

```
/*****************************************************************************

The Scanning Tunneling Microscope Project

Laboratory for Manufacturing and Productivity
The Department of Mechanical Engineering
Massachusetts Institute of Technology

       Header File
       control.h

Description  :

This has the prototypes of the subroutines and the declarations of constant variables
for the address of the DSP board interface registers and the structures for the two
data types used in the DSP interface.

Written by Jungmok Mitchell Bae
4/17/94

*****************************************************************************/

const unsigned int CNTL =    0x300 + 0x04;
const unsigned int PMAD =    0x300 + 0x10;
const unsigned int DMAD =    0x300 + 0x12;
const unsigned int TMG  =    0x300 + 0x12;
const unsigned int STAT =    0x300 + 0x10;
const unsigned int DMDL =    0x300 + 0x1E;
const unsigned int DMDM =    0x300 + 0x1C;
const unsigned int DMDH =    0x300 + 0x1A;
const unsigned int PMDL =    0x300 + 0x18;
const unsigned int PMDM =    0x300 + 0x16;
const unsigned int PMDH =    0x300 + 0x14;
const unsigned int PMWR =    0x300 + 0x02;
const unsigned int DMWR =    0x300 + 0x00;
const unsigned int PMRD =    0x300 + 0x02;
const unsigned int DMRD =    0x300 + 0x00;

/* data type for 48-bit DSP program memory inst. */
struct HEX48 {
        unsigned int USW;
        unsigned int MSW;
        unsigned int LSW;
        };

/* data type for 40-bit unformatted values for DM or PM*/
struct HEX40 {
        unsigned int USW;
        unsigned int MSW;
        unsigned char LSB;
        };
union f_l {
      unsigned long l;
      float f; };
void run( void );
void reset( void );
void stat( void );
```

```
int statreg( void );
void timing( void );
int loadmem_i( unsigned int start_adr, unsigned int num_words,
      unsigned int *datarray, unsigned int bank );
int loadmem_l( unsigned int start_adr, unsigned int num_words,
      unsigned long *datarray, unsigned int bank );
int loadmem_pm( unsigned int start_adr, unsigned int num_words,
      struct HEX48 *datarray, unsigned int bank );
int loadmem_dm( unsigned int start_adr, unsigned int num_words,
      struct HEX40 *datarray, unsigned int bank );
int loadmem_f( unsigned int start_adr, unsigned int num_words,
      float *datarray, unsigned int bank );

int readmem_i( unsigned int start_adr, unsigned int num_words,
      unsigned int *datarray, unsigned int bank );
int readmem_l( unsigned int start_adr, unsigned int num_words,
      unsigned long *datarray, unsigned int bank );
int readmem_dm( unsigned int start_adr, unsigned int num_words,
      struct HEX40 *datarray, unsigned int bank );
int readmem_pm( unsigned int start_adr, unsigned int num_words,
      struct HEX48 *datarray, unsigned int bank );
int readmem_f( unsigned int start_adr, unsigned int num_words,
      float *datarray, unsigned int bank );

void down( char *file_name );

/*****************************************************************************

The Scanning Tunneling Microscope Project

Laboratory for Manufacturing and Productivity
The Department of Mechanical Engineering
Massachusetts Institute of Technology

      Main Program
      constm3.c

Description:

This program drives the scanning tunneling microscope. It communicate with the
digital signal processing board using the DSP interface subroutines such as
download function, run function, reset function, memory read and write function.
This program is divided into smaller modules. A user can control each step of
scanning process using these modules. The program also provides the user
interface functions and the display functions of the result.

The program uses the following header files.

      control.h
      constm.h

Written by Jungmok Mitchell Bae         4/17/94

*****************************************************************************/
#include "control.h"
#include "constm.h"

/*****************************************************************************
*
*       Define Macros
*
*       RGB( r, g, b )   : Color settings. (Refer to graph.h)
*       TN_DAT( i, j )   : 2-D array representation of 1-D array.
*       MAX              : Maximum number of data in each row.
```

```
*       WAIT            : Time delay routine.
*/
#define RGB( r, g, b ) (0x3F3F3FL & ((long)(b) << 16 | (g) << 8 | (r) ))
#define TRUE 1
#define FALSE 0
#define TN_DAT( i, j ) tn_dat[ (64*(i)+(j)) ]   /* You have to change this part */
#define MAX     100     /* in order to vary the resolution*/
#define WAIT    for(i=0;i<1000;i++)
```

C.2 Main Function

```
/*****************************************************************************
*
*       Declaration of Global Variable
*
*       dat[MAX][MAX]   : Tunneling current data are stored here!
*       lpos            : The last position of the tip when the program is
*                               finished.
*/
unsigned        int dat[MAX][MAX];
int             resol=64, range;
unsigned long   lpos=0;             /* The last pos. is stored here. */
float           PreRef=0;

main()
{
       while(mainmenu());
}

/*****************************************************************************
*
*       int mainmenu( void )
*
*       Description:    Print main menu, wait for the user input, and execute
*                       the selection
*
*       Return:         0 = user terminates the session.
*                       None = otherwise.
*
*       Note :          The menu for a new mode should be added here!!!
*/
int mainmenu( void )
{
       int             flag=1, n, ans;

       while( flag != 0 ){
              printf( "Main Menu\n");
              printf( "       0. Reset the STM\n" );
              printf( "       1. Coarse Approach\n" );
              printf( "       2. Constant Height Mode\n" );
              printf( "       3. Constant Current Mode\n" );
              printf( "       4. Wide Scanning Mode\n" );
              printf( "       5. Atomic Encoder Mode\n" );
              printf( "       6. New Mode???\n" );
              printf( "       7. Data Interpretation Mode\n" );
              printf( "       8. Termination of the session\n" );
              n = get_int( "your choice", 0, 8 );
              switch( n ){
                     case 0:
                            parameter_set();
                            break;
                     case 1:
                            coarse_approach1_atomsurf();
                            break;
                     case 2:
```

```
                        constant_height_mode();
                        break;
                case 3:
                        constant_current_mode();
                        break;
                case 4:
                        wide_scanning_mode();
                        break;
                case 5:
                        atomic_encoder_mode();
                        break;
                case 6:
                        break;
                case 7:
                        data_interpret_mode();
                        break;
                case 8:
                        ans=waitYN("Termination of the session -
sure?(Y/N/<Enter>)",'N');
                        if(ans==1)flag = 0;
                        break;
                default:
                        printf( "Wrong input.\n");
                        break;
            }
        }
        return 0;
}
```

C.3 The STM Control Support Subroutines

```
/*************************************************************************
*
*       void parameter_set( void )
*
*       Description:    Reset the scanning tunneling microscope.
*                       Set all d/a voltages to preset value.
*                       Refer to 'init.asm'.
*
*       Return:         None
*
*/
void parameter_set( void )
{
        reset();
        down("init.stk");
        run();
        reset();
}

/*************************************************************************
*
*       void move_inchworm_motor( void )
*
*       Description:    Move the inchworm motor. The module controls the speed,
*                       the direction, and the moving range of inchworm motor.
*                       Refer to 'moveiw.asm'.
*
*       Return:         None
*
*/
void move_inchworm_motor( void )
{
```

```
        int             flag1;
        unsigned long   loop, forb, tper;
        float           tsamp;

/*      user input1 - sampling rate     */
        printf("The IW speed selection\n");
        printf("        0. fast\n");
        printf("        1. medium\n");
        printf("        2. slow\n");
        flag1 = get_int( "the mode of the speed",0,2);
        switch( flag1){
                case 0:
                        tsamp = 1.0;
                        break;
                case 1:
                        tsamp = 0.5;
                        break;
                default:
                        tsamp = 0.2;
                        break;
        }
        tper = ( unsigned long )(33300.0 / tsamp);

/*      user input2 - the direction    */
        forb = get_long( "Direction -> upward(0) or downward(1).", 0, 1 );
        if( forb )
                forb = 0x4;
        else
                forb = 0x0;
/*      user input3 - the maximum clock cycle     */
        loop = get_long( "the maximum clock cycle", 0, 100000 );

        reset();

        down( "moveiw.stk" );

        loadmem_l( 0x00000000, 1, &tper, 1 );
        loadmem_l( 0x00000002, 1, &forb, 1 );
        loadmem_l( 0x00000001, 1, &loop, 1 );

        printf( "Now starts ..... if you want to stop hit any key now\n" );
        run();
        getch();
        reset();
}
/*****************************************************************************
*
*       void coarse_approach1_atomsurf( void )
*
*       Description:    This program is for the coarse approach.
*                       The speed has been set to a default value, 0.5kHz.
*                       The program asks for the number of ADC reading in
*                       multiple of 0.5kHz.
*                       Refer to 'coarse11.asm'.
*
*       Return:         None
*
*/
void coarse_approach1_atomsurf( void )
{
        unsigned long     tper, rto;
        float             tsamp;

/*      user input1 - sampling rate */
```

```
        tsamp = get_float( "sampling period in multiple of 0.5kHz", 0, 10 );
        rto=(unsigned long)tsamp/0.5;
        tper = ( unsigned long )(33300.0 / tsamp);

        reset();
        down( "coarse11.stk" );
        loadmem_1( 0x00000000, 1, &tper, 1 );/* send sampling period to DSP */
        loadmem_1( 0x00000001, 1, &rto, 1 ); /* send ratio of a/d & IW's speed*/

        run();
        while( statreg() != 1 );

        if( statreg() == 0x23 ){
              printf( "Input sampling rate is too fast.\n" );
              exit(1);
        }
        printf( "The coarse approach1 is now working!\n" );
        printf( "Emergency stop:hit any key now!\n" );
        getch();
        reset();
        if(statreg() != 0 ) printf( "The program was in progress...\n\n" );

}

/*************************************************************************
*
*       void fine_approach_atomsurf( void )
*
*       Description:    This is program is for the next approaching mode,
*                       fine approach.
*                       Refer to 'fine1.asm'.
*
*       Return:         None
*
*/
void fine_approach_atomsurf( void )
{
       unsigned long     tper, lastpos;
       float             tsamp;

/*      user input1 - sampling rate     */
       tsamp = get_float( "sampling period in kHz", 0, 10 );
       tper = ( unsigned long )(33300.0 / tsamp);

       reset();
       down( "fine1.stk" );
       loadmem_1( 0x0000, 1, &tper, 1 );/* send sampling period to DSP */

       run();
       while( statreg() != 1 );
       if( statreg() == 0x23 ){
              printf( "Input sampling rate is too fast.\n" );
              exit(1);
       }
       printf( "The fine approach is now working.\n" );
       printf( "Emergency stop:hit any key now!\n" );
       getch();
       reset();
       readmem_1( 0x0001, 1, &lpos, 1 ); /* update the last position.*/

       if(statreg() != 0 )
              printf( "The program was in progress...\n\n" );
       else
              printf("O.K.!   it's done.\n" );
```

```
}

/*****************************************************************************
*
*       void pidcontrol_atomsurf( void )
*
*       Description:    PID control of the tunneling current to user input
*                                                       reference voltage.
*                       Refer to 'pidcon.asm'.
*
*       Return:         None
*
*/
void pidcontrol_atomsurf( void )
{
       unsigned long   tper, tempo;
       float           gaini, gainp, gaind, refv, fsamp, tsamp, a, b, c;

/* user input1 - sampling rate */
       fsamp = get_float( "sampling period in Hz", 0, 100000 );
       tper = ( unsigned long )(33300000.0 / fsamp);
/* user input2 - reference voltage */
       printf("Enter the reference voltage ");
       scanf("%e", &refv);
       refv=refv*8191.0/5.0;
       /*calculation to 16bit resolution 10V range value...*/
/* user input3 - proportional gain */
       printf("Enter proportional gain K ");
       scanf("%e", &gainp );
/* user input4 - integral gain */
       printf( "Enter integral gain Ti");
       scanf("%e", &gaini );
/* user input5 - derivative gain */
       printf( "Enter derivative gain Td" );
       scanf("%e", &gaind );
/* calculate the coefficients a, b, c */
       tsamp=1.0/fsamp;          /* T */
       a=gainp*(1.0+tsamp/gaini+gaind/tsamp);
       b=(-1.0)*gainp*gaind/tsamp;
       c=gainp*tsamp/gaini;

       reset();
       down( "pidcon.stk" );  /* pidcon1.asm is the version for
                                 saturation+integral windup*/
       loadmem_l( 0x0000, 1, &tper, 1 );
       loadmem_f( 0x0006, 1, &refv, 1 );
       loadmem_f( 0x0008, 1, &a, 1 );
       loadmem_f( 0x0009, 1, &b, 1 );
       loadmem_f( 0x000b, 1, &c, 1 );
       logtable_download();

/* pcinput1 - last position of the tip */
       loadmem_l( 0x0001, 1, &lpos, 1 );       /*load the previous position*/

       run();
       printf( "The pidcontrol is now working.\n" );
       printf( "Emergency stop:hit any key now!\n" );
       while(kbhit()==0){
              if(statreg() == 0x23 ){
                     printf("PID vol is out of limit\n" );
                     exit(1);
              }
       }
       readmem_l( 0x0001, 1, &lpos, 1 );
```

```
        reset();
        if(statreg() != 0 )
                printf( "Was in progress...\n\n" );
        else
                printf( "O.K.! it's done. \n\n" );
}

/*****************************************************************************
*
*       void store_atomsurf( void )
*
*       Description:    Store the atomic image in user specified file.
*                       The module uses data stored in the global variable
*                                                       , dat[][].
*
*       Return:         None
*
*/
void store_atomsurf( void )
{
        char    *fn;
        FILE    *fp;
        int     i, j;

        fn = get_string( "file name" );
        fp = fopen( fn, "w" );
        for( i = 0; i < 64; i++ ){
                for( j = 0; j < 64; j++ )
                        fprintf( fp, "%7d ", dat[i][j] );
                fprintf( fp, "\n" );
        }
        free( fn );
        fclose( fp );
}

/*****************************************************************************
*
*       void retrieve_atomsurf( void )
*
*       Description:    Retrieve the atomic image data from the user input
*                       file. The data goes to the globally defined dat[][].
*
*       Return:         None
*
*/
void retrieve_atomsurf( void )
{
        char    *fn;
        FILE    *fp;
        int     i, j, size;

        fn = get_string( "file name" );
        size = get_int( "size of the data", 0, 100 );
        fp = fopen( fn, "r" );
        for( i = 0; i < size; i++ ){
                for( j = 0; j< size; j++ )
                        fscanf( fp, "%7d ", &dat[i][j] );
                fscanf( fp, "\n" );
        }
        free( fn );
        fclose( fp );
}

/*****************************************************************************
```

```
*
*       void logtable_download( void )
*
*       Description:    This function downloads the log table to the DSP data memory
*                        location from 0x0010 to 0x2000. The input voltage ranges
*                        from 10mV - 5V.
*
*       Return:         None
*
*/
void logtable_download( void )
{
       int     i;
       long     *logval;
       double  temp;

       logval = (long*)malloc( 9000*sizeof( long ) );
       if( logval==NULL ){
             printf( "Allocation is not succesful.\n" );
             exit( 1 );
       }
       for( i=0; i<10; i++ )
             logval[i]=(long)(2.76*(1.0-log10(10.0*5.0/8191.0))*1638.0);
       for( i=10; i<9000; i++ ){
             logval[i]=(long)(2.76*(1.0-log10((double)i*5.0/8191.0))*1638.0);
       }
       loadmem_l( 0x2000, 8192, logval, 1 );
       free( logval );
}

/*************************************************************************
*
*       void convert_to_log( void )
*
*       Description:    Perfom the Log conversion of the scanned data.
*                       The module uses the log conversion equation described in
*                                                    section 5 in chapter 2.
*       Return:         None
*
*/
void convert_to_log( void )
{
       int i, j;

/* convert the result into log */
       for(i=0;i<64;i++){
             for( j=0; j<64; j++ ){
               if( dat[i][j] <10 ){
                    printf("The T.C. during scan is out of range.%d, %d\n", i,j );
                    printf("%d\n", dat[i][j] );
                    exit( 1 );
               }
               dat[i][j]=(int)(2.76*(1.0-log10( (double)dat[i][j]/8191.0*5.0
))*1638.0);
             }
       }
}
```

C.4The Atomic Image Display Subroutines

```
float   elm[11][11];
/*************************************************************************
*
*       void display_atomsurf_f( void )
```

```
*
*       Description:    Display the atomic image in grey scale.
*                       The moudule uses the floating point data which has been
*                                                       digitally filtered prior to
this operation.
*
*       Return:         None
*
*/
void display_atomsurf_f( void )
{
       float   *tn_dat;
       float   max, min, step;
       int     rowst, colst, i, j, k, l;
       long    grey[16], col;
       char    *fn;
       FILE    *fp;
       double  col_ind, winsize;

/* dynamic memory allocation */
       tn_dat = (float*)malloc( resol*resol*sizeof( float ) );
       if(tn_dat==NULL){
              printf( "Allocation is not succesful.\n" );
              exit(1);
       }

/* retrieve the data */
       fn = get_string( "file name" );
       fp = fopen( fn, "r" );
       for( i = 0; i < resol; i++ ){
              for( j = 0; j< resol; j++ )
                     fscanf( fp, "%e ", &TN_DAT(i,j) );
              fscanf( fp, "\n" );
       }
       free( fn );
       fclose( fp );

/* graphic initialization */
       _setvideomode( _VRES16COLOR );
       _setviewport( 330, 90, 629, 389 );
/*_settextwindow*/
       winsize=(double)((resol-1)*10);
       _setwindow( TRUE, 0.0, 0.0, winsize, winsize );

       for( k=0; k<16; k++ )
              grey[k] = RGB( 4*k, 4*k, 4*k );
       _remapallpalette( grey );

/* find the maximum of the data */
       for( i=0; i<resol; i++ ){
              for( j=0; j<resol; j++ ){
                     if( TN_DAT(i,j) > max )
                            max=TN_DAT(i,j);
              }
       }
/* find the minimum of the data */
       for( i=0; i<resol; i++ ){
              for( j=0; j<resol; j++ ){
                     if( TN_DAT(i,j) < min )
                            min=TN_DAT(i,j);
              }
       }
/* find the step size */
       step=(max-min)/16.0;
```

```
/* set the color for each point 64*64 */
      for( i=0; i<resol; i++ ){
            for( j=0; j<resol; j++ ){
                  modf((double)((TN_DAT(i,j)-min)/step), &col_ind );
                  /*TN_DAT(i,j)=(float)(16.0-col_ind);*/
                  TN_DAT(i,j)=(float)(col_ind);
            }
      }

/* draw the graph piece by piece */
      for( i=0; i<(resol-1); i++ ){
            for( j=0; j<(resol-1); j++ ){
                  linear_interpo( TN_DAT(i,j), TN_DAT(i,j+1), TN_DAT(i+1,j+1),
TN_DAT(i+1,j) );
                  rowst=10*i;
                  colst=10*j;
                  for(k=0;k<=10;k++){
                        for(l=0;l<=10;l++){
                              _setcolor( (int)elm[k][l] );
                              _setpixel_w( (double)(colst+l), (double)(rowst+k)
);
                        }
                  }
            }
      }

      free(tn_dat);
}

/*************************************************************************
*
*       void display_atomsurf( void )
*
*       Description:    Display the atomic image data in grey scale.
*                       The module uses the fixed point data stored in the array
dat[][].
*
*       Return:         None
*
*/
void display_atomsurf( void )
{
      float   *tn_dat;
      float   max, min, step;
      int     rowst, colst, i, j, k, l, temp, ans;
      long    grey[16], col;
      double  col_ind, winsize;

/* ask user if the data to be log converted */
      ans=waitYN("Log convert? (Y/N/<Enter>)",'Y');
      if(ans=1) convert_to_log();

/* dynamic memory allocation */
      tn_dat = (float*)malloc( resol*resol*sizeof( float ) );
      if(tn_dat==NULL){
            printf( "Allocation is not succesful.\n" );
            exit(1);
      }

/* retrieve the data */
      for( i = 0; i < resol; i++ ){
            for( j = 0; j< resol; j++ ){
                  TN_DAT(i,j)=(float)dat[i][j];
```

```
                }
        }
/* graphic initialization */
        _setvideomode( _VRES16COLOR );
        _setviewport( 330, 90, 629, 389 );
/*_settextwindow*/
        winsize=(double)((resol-1)*10);
        _setwindow( TRUE, 0.0, 0.0, winsize, winsize );

        for( k=0; k<16; k++ )
                grey[k] = RGB( 4*k, 4*k, 4*k );
        _remapallpalette( grey );

        max=TN_DAT(0,0);
        min=TN_DAT(0,0);
/* find the maximum of the data */
        for( i=0; i<resol; i++ ){
                for( j=0; j<resol; j++ ){
                        if( TN_DAT(i,j) > max )
                                max=TN_DAT(i,j);
                }
        }
/* find the minimum of the data */
        for( i=0; i<resol; i++ ){
                for( j=0; j<resol; j++ ){
                        if( TN_DAT(i,j) < min )
                                min=TN_DAT(i,j);
                }
        }
/* find the step size */
        step=(max-min)/16.0;

/* set the color for each point 64*64 */
        for( i=0; i<resol; i++ ){
                for( j=0; j<resol; j++ ){
                        modf((double)((TN_DAT(i,j)-min)/step), &col_ind );
                        /*TN_DAT(i,j)=(float)(16.0-col_ind);*/
                        TN_DAT(i,j)=(float)(col_ind);

                }
        }

/* draw the graph piece by piece */
        for( i=0; i<(resol-1); i++ ){
                for( j=0; j<(resol-1); j++ ){
                        linear_interpo( TN_DAT(i,j), TN_DAT(i,j+1), TN_DAT(i+1,j+1),
TN_DAT(i+1,j) );
                        rowst=10*i;
                        colst=10*j;
                        for(k=0;k<10;k++){
                                for(l=0;l<10;l++){
                                        _setcolor( (int)elm[k][l] );
                                        _setpixel_w( (double)(colst+l), (double)(rowst+k)
);
                                }
                        }
                }
        }

        free(tn_dat);
}

/*************************************************************************
*
```

```
*       void linear_interpo( float arg1, float arg2, float arg3, float arg4 )
*
*       Description:    This is the sub-function of display_atomsurf and
*                                                       display_atomsurf_f. The
function performs the linear
*                                                       interpolation of given 4
points. The resolution
*                       is presetted 10 by 10.
*
*       Input:          The four points which form the smallest square in
*                       the 2-D atom plane.
*
*       Return:         None
*
*/
void linear_interpo( float cn1, float cn2, float cn3, float cn4 )
{
       int     i, j;

/* boundary value calculation */
       for( i=0; i<=10; i++ )
              elm[0][i]=(cn2-cn1)/10.0*(float)(i)+cn1;
       for( i=0; i<=10; i++ )
              elm[10][i]=(cn3-cn4)/10.0*(float)(i)+cn4;
/* inside boundary value calculation */
       for( j=0; j<=10; j++ ){
              for( i=1; i<10; i++ )
                     elm[i][j]=(elm[10][j]-elm[0][j])/10.0*(float)(i)+elm[0][j];
       }
}
```

C.5 Constant Height Mode Module

```
/*************************************************************************
*
*       void constant_height_mode( void )
*
*       Description:    Print menu, wait for the user input, and execute
*                       the selection
*
*       Return:         No return value.
*
*/
void constant_height_mode( void )
{
       int             flag, n, ans;

       while( flag != 0 ){
              printf( "Constant Height Mode\n");
              printf( "       0. Reset the STM\n" );
              printf( "       1. Inchworm Motor Control\n" );
              printf( "       2. Manual Scanning Tools\n" );
              printf( "       3. Scanning -- Ramp Pattern\n" );
              printf( "       4. Scanning -- Triangular Pattern\n" );
              printf( "       5. Atomic Image Display\n" );
              printf( "       6. End the session\n" );
              n = get_int( "your choice", 0, 6 );
              switch( n ){
                     case 0:
                            parameter_set();
                            break;
                     case 1:
                            move_inchworm_motor();
                            break;
```

```
                        case 2:
                                manual_scanning_mode();
                                break;
                        case 3:
                                ramp_auto_scan_atomsurf();
                                break;
                        case 4:
                                tri_auto_scan_atomsurf();
                                break;
                        case 5:
                                display_atomsurf();
                                getch();
                                _clearscreen( _GCLEARSCREEN );
                                _setvideomode( _DEFAULTMODE );
                                break;
                        case 6:
                                ans=waitYN("Termination of the session -
sure?(Y/N/<Enter>)",'N');
                                if(ans=1)flag = 0;
                                break;
                        default:
                                printf( "Wrong input.\n");
                                break;
                }
        }

}

/*************************************************************************
*
*       void manual_scanning_mode( void )
*
*       Description:    Print sub menu for the manual scanning mode, wait for the
user
*                       input, and execute the selection
*
*       Return:         No return value.
*
*/
void manual_scanning_mode( void )
{
        int             flag, n, ans;

        while( flag != 0 ){
                printf( "Manual Scanning Mode\n");
                printf( "       0. Reset the STM\n" );
                printf( "       1. Inchworm Motor Control\n" );
                printf( "       2. Fine Positioning\n" );
                printf( "       3. Initial Height PID Control\n" );
                printf( "       4. Scanning -- Ramp Pattern\n" );
                printf( "       5. Scanning -- Triangular Pattern\n" );
                printf( "       6. Atomic Image Display\n" );
                printf( "       7. End the session\n" );
                n = get_int( "your choice", 0, 7 );
                switch( n ){
                        case 0:
                                parameter_set();
                                break;
                        case 1:
                                move_inchworm_motor();
                                break;
                        case 2:
                                fine_approach_atomsurf();
                                break;
```

```
                        case 3:
                                pidcontrol_atomsurf();
                                break;
                        case 4:
                                ramp_scan_atomsurf();
                                break;
                        case 5:
                                tri_scan_atomsurf();
                                break;
                        case 6:
                                display_atomsurf();
                                getch();
                                _clearscreen( _GCLEARSCREEN );
                                _setvideomode( _DEFAULTMODE );
                                break;
                        case 7:
                                ans=waitYN("Termination of the session -
sure?(Y/N/<Enter>)",'N');
                                if(ans=1)flag = 0;
                                break;
                        default:
                                printf( "Wrong input.\n");
                                break;
                }
        }

}

/*************************************************************************
*
*       void ramp_auto_scan_atomsurf( void )
*
*       Description:    This program is for fine positioning, pid controlling,
*                       and constant height scanning in ramp pattern. The module
*                                                       automatically sequencing
those three steps of scanning process.
*
*       Return:         None
*
*/
void ramp_auto_scan_atomsurf( void )
{
        unsigned long     tper, lastpos;
        float   gaini, gainp, gaind, refv, fsamp, tsamp, a, b, c;
        int             adr, i, j, flag1=0,ans;
        unsigned int    *datpt;

/**********************         Fine Positioning        ******************/
/* user input1 - sampling rate */

        tsamp = 0.05;
        tper = ( unsigned long )(33300.0 / tsamp);

        reset();
        down( "fine1.stk" );
        loadmem_l( 0x0000, 1, &tper, 1 );/* send sampling period to DSP */

        run();

        printf( "The fine approach is now working.\n" );

        while(statreg()!=0);
        reset();
        readmem_l( 0x0001, 1, &lpos, 1 ); /* update the last position.*/
```

```
/**********************          PID Control      ***************************/
/* user input1 - sampling rate */
       fsamp = 4000.0;
       tper = ( unsigned long )(33300000.0 / fsamp);
/* user input2 - reference voltage */
       refv=3.0;
       refv=refv*8191.0/5.0;
/* user input3 - proportional gain */
       gainp=0.00065;
/* user input4 - integral gain */
       gaini=0.065;
/* user input5 - derivative gain */
       gaind=0.016;
/* calculate the coefficients a, b, c */
       tsamp=1.0/fsamp;          /* T */
       a=gainp*(1.0+tsamp/gaini+gaind/tsamp);
       b=(-1.0)*gainp*gaind/tsamp;
       c=gainp*tsamp/gaini;

       reset();
       down( "pidcon.stk" );
       loadmem_l( 0x0000, 1, &tper, 1 );
       loadmem_f( 0x0006, 1, &refv, 1 );
       loadmem_f( 0x0008, 1, &a, 1 );
       loadmem_f( 0x0009, 1, &b, 1 );
       loadmem_f( 0x000b, 1, &c, 1 );
       logtable_download();
       loadmem_l( 0x0001, 1, &lpos, 1 ); /*load the previous position*/

       run();

       printf( "The pidcontrol is now working.\n" );
       while(statreg()!=0){
              if(statreg()==0x23){
                     printf("PID vol is out of limit\n");
                     exit(1);
              }
       }
       reset();
       readmem_l( 0x0001, 1, &lpos, 1 );

/**************************      Scanning        ***************************/
/* user input1 - sampling rate */
       tsamp = 6.4;
       tper = ( unsigned long )(33300.0 / tsamp);

       reset();
       down( "scan1.stk" );

       loadmem_l( 0x0000, 1, &tper, 1 );
/* pcinput1 - last position of the tip */
       loadmem_l( 0x0001, 1, &lpos, 1 );
       run();
       printf( "scanning is now working.\n" );
       while(statreg()!=0);
       reset();
/* collect the data from dsp */
       adr = 0x100;
       for( i = 0; i < 64; i++ ){
              for( j=0; j<64; j++ ){
                     readmem_i( adr, 1, &(dat[i][j]), 1 );
                     adr++;
              }
```

```
        }

        /* converting to log  and display */
        for( i=0; i<64; i++ )
              printf( "dat %d = %d\n", i, dat[i][i] );
        /* data save */
        ans=waitYN("Save? (Y/N/<Enter>)",'Y');
        if(ans=1) store_atomsurf();
}

/*****************************************************************************
*
*       void tri_auto_scan_atomsurf( void )
*
*       Description:    This program is for fine positioning, pid controlling,
*                       and constant height scanning in triangular pattern.
*
*       Return:         None
*
*/
void tri_auto_scan_atomsurf( void )
{
        unsigned long     tper, lastpos;
        float   gaini, gainp, gaind, refv, fsamp, tsamp, a, b, c;
        int             adr, i, j, ans;
        unsigned int    *datpt;

/**********************         Fine Positioning        ******************/
/* user input1 - sampling rate */
        tsamp = 0.2;
        tper = ( unsigned long )(33300.0 / tsamp);

        reset();
        down( "fine1.stk" );
        loadmem_l( 0x0000, 1, &tper, 1 );/* send sampling period to DSP */

        run();

        printf( "The fine approach is now working.\n" );
        while( statreg()!=0);
              WAIT;
        reset();
        readmem_l( 0x0001, 1, &lpos, 1 ); /* update the last position.*/

/**********************         PID Control     ***************************/
/* user input1 - sampling rate */
        fsamp = 4000.0;
        tper = ( unsigned long )(33300000.0 / fsamp);
/* user input2 - reference voltage */
        refv=3.0;
        refv=refv*8191.0/5.0;
/* user input3 - proportional gain */
        gainp=0.00065;
/* user input4 - integral gain */
        gaini=0.065;
/* user input5 - derivative gain */
        gaind=0.016;
/* calculate the coefficients a, b, c */
        tsamp=1.0/fsamp;          /* T */
        a=gainp*(1.0+tsamp/gaini+gaind/tsamp);
        b=(-1.0)*gainp*gaind/tsamp;
        c=gainp*tsamp/gaini;

        reset();
```

```
        down( "pidcon.stk" );
        loadmem_l( 0x0000, 1, &tper, 1 );
        loadmem_f( 0x0006, 1, &refv, 1 );
        loadmem_f( 0x0008, 1, &a, 1 );
        loadmem_f( 0x0009, 1, &b, 1 );
        loadmem_f( 0x000b, 1, &c, 1 );
        logtable_download();
        loadmem_l( 0x0001, 1, &lpos, 1 ); /*load the previous position*/

        run();

        printf( "The pidcontrol is now working.\n" );
        while(statreg() != 0 ){
                if(statreg()==0x23){
                        printf("PID vol is out of limit\n");
                        exit(1);
                }
        }
        reset();
        readmem_l( 0x0001, 1, &lpos, 1 );

/*************************      Scanning        ****************************/
/* user input1 - sampling rate */
        tsamp = 6.4;
        tper = ( unsigned long )(33300.0 / tsamp);

        reset();
        down( "scan2.stk" );

        loadmem_l( 0x0000, 1, &tper, 1 );
/* pcinput1 - last position of the tip */
        loadmem_l( 0x0001, 1, &lpos, 1 );
        run();
        printf( "scanning is now working.\n" );
        while(statreg() != 0 )
                WAIT;
        reset();
/* collect the data from dsp */
        adr = 0x100;
        for( i = 0; i < 64; i++ ){
                for( j=0; j<64; j++ ){
                        readmem_i( adr, 1, &(dat[i][j]), 1 );
                        adr++;
                }
                i++;
                for( j=63; j>=0; j-- ){
                        readmem_i(adr, 1, &(dat[i][j]), 1 );
                        adr++;
                }
        }

        for( i=0; i<64; i++ )
                printf( "dat %d = %d\n", i, dat[i][i] );
        /* data save */
        ans=waitYN("Save? (Y/N/<Enter>)",'Y');
        if(ans=1) store_atomsurf();
}

/*****************************************************************************
*
*       void ramp_scan_atomsurf( void )
*
*       Description:    The function performs the constant height scanning in ramp
pattern
```

```
*                                                       It can be only used after
the initial height has been set.
*
*       Return:         None
*
*/
void ramp_scan_atomsurf(void)
{
      unsigned long   tper, tempo;
      float           tsamp;
      int             adr, i, j, ans;
      unsigned int    *datpt;

/* user input1 - sampling rate */
      tsamp = get_float( "sampling period in kHz", 0, 50 );
      tper = ( unsigned long )(33300.0 / tsamp);

      reset();
      down( "scan1.stk" );

      loadmem_1( 0x0000, 1, &tper, 1 );
/* pcinput1 - last position of the tip */
      loadmem_1( 0x0001, 1, &lpos, 1 );
      run();
      printf( "scanning is now working.\n" );
      printf( "Emergency stop:hit any key now!\n" );
      getch();
/* collect the data from dsp */
      adr = 0x100;
      for( i = 0; i < 64; i++ ){
            for( j=0; j<64; j++ ){
                  readmem_i( adr, 1, &(dat[i][j]), 1 );
                  adr++;
            }
      }
      for( i=0; i<64; i++ )
            printf( "dat %d = %d\n", i, dat[i][i] );
      reset();
      if(statreg() != 0 ){
            printf( "The program was in progress...\n\n" );
      }
      else{
            /* data save */
            ans=waitYN("Save? (Y/N/<Enter>)",'Y');
            if(ans=1) store_atomsurf();
      }
}

/*****************************************************************************
*
*       void tri_scan_atomsurf( void )
*
*       Description:    The function performs the constant height scanning in
triangular pattern.
*
*       Return:         None
*
*/
void tri_scan_atomsurf(void)
{
      unsigned long   tper, tempo;
      float           tsamp;
      int             adr, i, j, ans;
      unsigned int    *datpt;
```

```
/* user input1 - sampling rate */
       tsamp = get_float( "sampling period in kHz", 0, 50 );
       tper = ( unsigned long )(33300.0 / tsamp);

       reset();
       down( "scan2.stk" );

       loadmem_l( 0x0000, 1, &tper, 1 );
/* pcinput1 - last position of the tip */
       loadmem_l( 0x0001, 1, &lpos, 1 );
       run();
       printf( "scanning is now working.\n" );
       printf( "Emergency stop:hit any key now!\n" );
       getch();
/* collect the data from dsp */
       adr = 0x100;
       for( i = 0; i < 64; i++ ){
              for( j=0; j<64; j++ ){
                     readmem_i( adr, 1, &(dat[i][j]), 1 );
                     adr++;
              }
              i++;
              for( j=63; j>=0; j-- ){
                     readmem_i(adr, 1, &(dat[i][j]), 1 );
                     adr++;
              }
       }
       for( i=0; i<64; i++ )
              printf( "dat %d = %d\n", i, dat[i][i] );
       reset();
       if(statreg() != 0 ){
              printf( "The program was in progress...\n\n" );
       }
       else{
              /* data save */
              ans=waitYN("Save? (Y/N/<Enter>)",'Y');
              if(ans=1) store_atomsurf();
       }
}
```

C.6 Constant Current Mode Module

```
/*****************************************************************************
*
*       void constant_current_mode( void )
*
*       Description:    Print the menu for the constant current mode, takes user
*                       input value, and execute the selection.
*
*       Return:         None
*
*/
void constant_current_mode( void )
{
       int flag, n;

       while( flag != 0 ){
              printf( "Constant Current Mode\n");
              printf( "       0. Reset the STM. \n" );
              printf( "       1. Inchworm Motor Control. \n" );
              printf( "       2. Fine Positioning. \n" );
              printf( "       3. Initial Height Control. \n" );
```

```
            printf( "       4. Scanning -- Ramp Pattern. \n" );
            printf( "       5. Scanning -- Triangular Pattern.\n");
            printf( "       6. Atomic Image Display.  \n" );
            printf( "       7. PID Gain Tuning. \n" );
            printf( "       8. End the Session. \n" );
            n = get_int( "your choice", 0, 8 );

            switch( n ){
                  case 0:
                        parameter_set();
                        break;
                  case 1:
                        move_inchworm_motor();
                        break;
                  case 2:
                        fine_approach_atomsurf();
                        break;
                  case 3:
                        cc_pidcontrol_atomsurf();
                        break;
                  case 4:
                        cc_ramp_scan_atomsurf();
                        break;
                  case 5:
                        cc_tri_scan_atomsurf();
                        break;
                  case 6:
                        display_atomsurf();
                        getch();
                        _clearscreen( _GCLEARSCREEN );
                        _setvideomode( _DEFAULTMODE );
                        break;
                  case 7:
                        manual_pid_tuning();
                        break;
                  case 8:
                        flag = 0;
                        break;

                  default:
                        printf( "Wrong input.\n");
                        break;
            }
      }
}

/*************************************************************************
*
*       void cc_pidcontrol_atomsurf( void )
*
*       Description:    The fn performs the initial height setting using PID scheme.
*                       The preset reference voltage is 3V.
*
*       Return:         None
*
*/
void cc_pidcontrol_atomsurf( void )
{
      unsigned long   tper, tempo;
      float           gaini, gainp, gaind, refv, fsamp, tsamp, a, b, c;

/* user input1 - sampling rate */
      /*fsamp = get_float( "sampling period in Hz", 0, 100000 );*/
      fsamp=4000.0;
```

```
        tper = ( unsigned long )(33300000.0 / fsamp);
/* user input2 - reference voltage */
        /*printf("Enter the reference voltage ");
        scanf("%e", &refv);*/
        refv=3.0;
        refv=refv*8191.0/5.0;
        /*calculation to 16bit resolution 10V range value...*/
/* user input3 - proportional gain */
        /*printf("Enter proportional gain K ");
        scanf("%e", &gainp );*/
        gainp=0.00065;
/* user input4 - integral gain */
        /*printf( "Enter integral gain Ti");
        scanf("%e", &gaini );*/
        gaini=0.065;
/* user input5 - derivative gain */
        /*printf( "Enter derivative gain Td" );
        scanf("%e", &gaind );*/
        gaind=0.016;
/* calculate the coefficients a, b, c */
        tsamp=1.0/fsamp;          /* T */
        a=gainp*(1.0+tsamp/gaini+gaind/tsamp);
        b=(-1.0)*gainp*gaind/tsamp;
        c=gainp*tsamp/gaini;

        reset();
        down( "pidcon.stk" );

        loadmem_l( 0x0000, 1, &tper, 1 );
        loadmem_f( 0x0006, 1, &refv, 1 );
        loadmem_f( 0x0008, 1, &a, 1 );
        loadmem_f( 0x0009, 1, &b, 1 );
        loadmem_f( 0x000b, 1, &c, 1 );
        logtable_download();

/* pcinput1 - last position of the tip */
        loadmem_l( 0x0001, 1, &lpos, 1 );       /*load the previous position*/

        run();
        printf( "The picontrol is now working.\n" );
        printf( "Emergency stop:hit any key now!\n" );
        while(kbhit()==0){
                if(statreg() == 0x23 ){
                        printf("PID vol is out of limit\n" );
                        exit(1);
                }
        }
        readmem_l( 0x0001, 1, &lpos, 1 );
        reset();
        if(statreg() != 0 )
                printf( "Was in progress...\n\n" );
        else
                printf( "O.K.! it's done. \n\n" );
}

/*****************************************************************************
*
*       void cc_ramp_scan_atomsurf( void )
*
*       Description:    The function performs the constant current mode scanning in
ramp
*                       pattern. The module assumes that all the prior steps are
done.
```

```
*                                                    Refer section 7 in chapter
2.
*
*       Return:         None
*
*/
void cc_ramp_scan_atomsurf( void )
{
      unsigned long   tper, tempo,ConNum;
      float           gaini, gainp, gaind, refv, fsamp, tsamp, a, b, c, RecConNum;
      int             adr, i, j;

/* user input1 - sampling rate */
      fsamp = get_float( "sampling period in kHz", 0, 500 );
      tper = ( unsigned long )(33300.0 / fsamp);
/* user input2 - reference voltage */
      printf("Enter the reference voltage ");
      scanf("%e", &refv);
      refv=refv*8191.0/5.0;
      /*calculation to 16bit resolution 10V range value...*/
/* user input3 - proportional gain */
      printf("Enter proportional gain K ");
      scanf("%e", &gainp );
/* user input4 - integral gain */
      printf( "Enter integral gain Ti");
      scanf("%e", &gaini );
/* user input5 - derivative gain */
      printf( "Enter derivative gain Td" );
      scanf("%e", &gaind );
/* user input6 - #conloop */
      ConNum=get_int("the number of control loop",0,100);

/* calculate the coefficients a, b, c */
      tsamp=1.0/(fsamp*1000.0);          /* T */
      a=gainp*(1.0+tsamp/gaini+gaind/tsamp);
      b=(-1.0)*gainp*gaind/tsamp;
      c=gainp*tsamp/gaini;
/* Calculate the reciprocal */
      RecConNum=1.0/(float)ConNum;

      reset();
      down( "ccscan1.stk" );

      loadmem_l( 0x0000, 1, &tper, 1 );
      loadmem_f( 0x0006, 1, &refv, 1 );
      loadmem_f( 0x0008, 1, &a, 1 );
      loadmem_f( 0x0009, 1, &b, 1 );
      loadmem_f( 0x000b, 1, &c, 1 );
      loadmem_l( 0x0002, 1, &ConNum, 1 );
      loadmem_f( 0x0003, 1, &RecConNum, 1 );
      logtable_download();

/* pcinput1 - last position of the tip */
      loadmem_l( 0x0001, 1, &lpos, 1 );       /*load the previous position*/

      run();
      printf( "The constant current mode is now working.\n" );
      printf( "Emergency stop:hit any key now!\n" );
      while(statreg()!=0){
            if(statreg() == 0x23 ){
                  printf("PID vol is out of limit\n" );
                  exit(1);
            }
      }
```

```
        reset();

/* collect the data from dsp */
        adr = 0x100;
        for( i = 0; i < 64; i++ ){
                for( j=0; j<64; j++ ){
                        readmem_i( adr, 1, &(dat[i][j]), 1 );
                        adr++;
                }
        }
        for( i=0; i<64; i++ )
                printf( "dat %d = %d\n", i, dat[i][i] );
        loadmem_l( 0x0001, 1, &lpos, 1 );       /*load the previous position*/
        printf( "The program is finished\n\n" );
}

/*****************************************************************************
*
*       void cc_tri_scan_atomsurf( void )
*
*       Description:    The function performs the constant current mode scanning in
triangular
*                       pattern.
*
*       Return:         None
*
*/
void cc_tri_scan_atomsurf( void )
{
        unsigned long   tper, tempo, ConNum;
        float           gaini, gainp, gaind, refv, fsamp, tsamp, a, b, c,RecConNum;
        int             adr, i, j;

/* user input1 - sampling rate */
        fsamp = get_float( "sampling period in kHz", 0, 500 );
        tper = ( unsigned long )(33300.0 / fsamp);
/* user input2 - reference voltage */
        printf("Enter the reference voltage ");
        scanf("%e", &refv);
        refv=refv*8191.0/5.0;
        /*calculation to 16bit resolution 10V range value...*/
/* user input3 - proportional gain */
        printf("Enter proportional gain K ");
        scanf("%e", &gainp );
/* user input4 - integral gain */
        printf( "Enter integral gain Ti");
        scanf("%e", &gaini );
/* user input5 - derivative gain */
        printf( "Enter derivative gain Td" );
        scanf("%e", &gaind );
/* user input6 - #conloop */
        ConNum=get_int("the number of control loop",0,100);

/* calculate the coefficients a, b, c */
        tsamp=1.0/(fsamp*1000.0);           /* T */
        a=gainp*(1.0+tsamp/gaini+gaind/tsamp);
        b=(-1.0)*gainp*gaind/tsamp;
        c=gainp*tsamp/gaini;
/* Calculate the reciprocal */
        RecConNum=1.0/(float)ConNum;
        reset();
        down( "ccscan.stk" );

        loadmem_l( 0x0000, 1, &tper, 1 );
```

```
        loadmem_f( 0x0006, 1, &refv, 1 );
        loadmem_f( 0x0008, 1, &a, 1 );
        loadmem_f( 0x0009, 1, &b, 1 );
        loadmem_f( 0x000b, 1, &c, 1 );
        loadmem_l( 0x0002, 1, &ConNum, 1 );
        loadmem_f( 0x0003, 1, &RecConNum, 1 );
        logtable_download();

/* pcinput1 - last position of the tip */
        loadmem_l( 0x0001, 1, &lpos, 1 );       /*load the previous position*/

        run();
        printf( "The constant current mode is now working.\n" );
        printf( "Emergency stop:hit any key now!\n" );
        while(statreg()!=0){
                if(statreg() == 0x23 ){
                        printf("PID vol is out of limit\n" );
                        exit(1);
                }
        }
        reset();

/* collect the data from dsp */
        adr = 0x100;
        for( i = 0; i < 64; i++ ){
                for( j=0; j<64; j++ ){
                        readmem_i( adr, 1, &(dat[i][j]), 1 );
                        adr++;
                }
                i++;
                for( j=63; j>=0; j-- ){
                        readmem_i(adr, 1, &(dat[i][j]), 1 );
                        adr++;
                }
        }
        for( i=0; i<64; i++ )
                printf( "dat %d = %d\n", i, dat[i][i] );
        loadmem_l( 0x0001, 1, &lpos, 1 );       /*load the previous position*/
        printf( "The program is done\n\n" );
}

/*************************************************************************
*
*       void manual_pid_tuning( void )
*
*       Description:    The function performs the pid gain tuning. It displays the
step response
*                       of the PID feedback C-L system. The user can tune P, I, and D
gain
*                                                       manually using this module.
Refer to 'pidtune.asm'.
*
*       Return:         None
*
*/
void manual_pid_tuning( void )
{
        unsigned long   tper, tper1, mult, *RespData;
        float           gaini, gainp, gaind, refv, fsamp1, fsamp2, tsamp, a, b, c;
        float           gaini1, gainp1, gaind1, refv1, fsamp3,tsamp1,a1,b1,c1;
        int             adr,sord;

/* user input1 - sampling rate for the pid control */
        fsamp1 = get_float( "sampling period for pid control in Hz", 0, 100000 );
```

```
/* user input2 - sampling rate for the response curve */
      fsamp2 = get_float( "sampling period for response curve in Hz", 0, 100000 );
      tper = ( unsigned long )(33300000.0 / fsamp2);
      mult=(unsigned long)(fsamp2/fsamp1);
/* user input3 - reference voltage */
      printf("Enter the reference voltage ");
      scanf("%e", &refv);
      refv=refv*8191.0/5.0;
      /*calculation to 16bit resolution 10V range value...*/
/* user input4 - proportional gain */
      printf("Enter proportional gain K ");
      scanf("%e", &gainp );
/* user input5 - integral gain */
      printf( "Enter integral gain Ti");
      scanf("%e", &gaini );
/* user input6 - derivative gain */
      printf( "Enter derivative gain Td" );
      scanf("%e", &gaind );
/* calculate the coefficients a, b, c */
      tsamp=1.0/fsamp1;          /* T */
      a=gainp*(1.0+tsamp/gaini+gaind/tsamp);
      b=(-1.0)*gainp*gaind/tsamp;
      c=gainp*tsamp/gaini;

/*********************          PID Control     ****************************/
/* user input1 - sampling rate */
      fsamp3 = 4000.0;
      tper1 = ( unsigned long )(33300000.0 / fsamp3);
/* user input2 - reference voltage */
      refv1=3.0;
      refv1=refv1*8191.0/5.0;
/* user input3 - proportional gain */
      gainp1=0.00065;
/* user input4 - integral gain */
      gaini1=0.065;
/* user input5 - derivative gain */
      gaind1=0.016;
/* calculate the coefficients a, b, c */
      tsamp1=1.0/fsamp3;          /* T */
      a1=gainp1*(1.0+tsamp1/gaini1+gaind1/tsamp1);
      b1=(-1.0)*gainp1*gaind1/tsamp1;
      c1=gainp1*tsamp1/gaini1;

      reset();
      down( "pidcon.stk" );
      loadmem_l( 0x0000, 1, &tper1, 1 );
      loadmem_f( 0x0006, 1, &refv1, 1 );
      loadmem_f( 0x0008, 1, &a1, 1 );
      loadmem_f( 0x0009, 1, &b1, 1 );
      loadmem_f( 0x000b, 1, &c1, 1 );
      logtable_download();
      loadmem_l( 0x0001, 1, &lpos, 1 ); /*load the previous position*/

      run();

      printf( "The pidcontrol is now working.\n" );
      while(statreg()!=0);
      reset();
      readmem_l( 0x0001, 1, &lpos, 1 );
      PreRef=refv1;
/******************************** PID Tuning ******************************/
      reset();
      down( "pidtune.stk" );
```

```
        loadmem_l( 0x0000, 1, &tper, 1 );
        loadmem_l( 0x0002, 1, &mult, 1 );
        loadmem_f( 0x0006, 1, &refv, 1 );
        loadmem_f( 0x0008, 1, &a, 1 );
        loadmem_f( 0x0009, 1, &b, 1 );
        loadmem_f( 0x000b, 1, &c, 1 );
        logtable_download();

/* pcinput1 - last position of the tip */
        loadmem_l( 0x0001, 1, &lpos, 1 );       /*load the previous position*/

        run();
        printf( "The pidtuning is now working.\n" );
        printf( "Emergency stop:hit any key now!\n" );
        getch();
        reset();
        readmem_l( 0x0001, 1, &lpos, 1 );

/* dynamic memory allocation */
        RespData = (unsigned long*)malloc( 1001*sizeof( unsigned long ) );
        if(RespData==NULL){
                printf( "Allocation is not succesful.\n" );
                exit(1);
        }
/* data retrieval */
        adr=0x800;
        readmem_l( adr, 1001, RespData, 0 );
/* graphic output */
        graph_init();
        plot( RespData, 1000, (int)fsamp2, refv, PreRef );
        getch();
        _clearscreen( _GCLEARSCREEN );
        _setvideomode( _DEFAULTMODE );
        sord=get_int("save(0) or discard(1)", 0,1);
        if(!sord)
                store_stepresp(RespData);
        PreRef=refv;
        free(RespData);
}

int halfx, halfy, xwidth, yheight, cols, rows;
struct videoconfig screen;

/*****************************************************************************
*
*       void graph_init( void )
*
*       Description:    This module is sub-function of plot(). It sets the graphic
*                       environment according to the type of a screen adapter.
*
*       Return:         None
*
*/
void graph_init( void )
{
        _getvideoconfig( &screen );
        switch( screen.adapter ){
                case _CGA:
                case _OCGA:
                        _setvideomode( _MRES4COLOR );
                        break;
                case _EGA:
                case _OEGA:
                        _setvideomode( _ERESCOLOR );
```

```
                        break;
                case _VGA:
                case _OVGA:
                        _setvideomode( _VRES16COLOR );
                        break;
                default:
                        printf( "This program requires a CGA, EGA, \
                        or VGA graphic card.\n" );
                        exit( 1 );
        }
        _getvideoconfig( &screen );
        _clearscreen( _GCLEARSCREEN );
        xwidth = screen.numxpixels;
        yheight = screen.numypixels;
        halfx = xwidth/2;
        halfy = yheight/2;
        cols = screen.numtextcols;
        rows = screen.numtextrows;

        /*set view port and the coordinates */
        _setviewport( 70, 70, 570, 410 );
        _setwindow( TRUE, 0.0, -10.0, 200.0, 160.0 );

}

/*************************************************************************
*
*       void plot( unsigned long arg1, int arg2, int arg3, float arg4, float arg5 )
*
*       Description:    Displays the plot the data stored in input array.
*       Input:          1. Data to be plotted.
*                       2. Maximum number of data to be plotted.
*                       3. The sampling rate.
*                       4. Reference voltage.
*                       5. Previous reference voltage.
*
*       Return:         None
*
*/
void plot( unsigned long *yval, int xrange, int tsamp, float ref, float prev )
{

        int     i;
        unsigned long min, max;
        double  x, y, temp1, temp2, xstep, ystep, xin, pos;
        char    txt[20];

        max=yval[0];
        min=yval[0];
/* find the maximum of the data */
        for( i=0; i<1000; i++ ){
                        if( yval[i] > max )
                                max=yval[i];
        }
/* find the minimum of the data */
        for( i=0; i<1000; i++ ){
                        if( yval[i] < min )
                                min=yval[i];
        }

        xstep = 200.0 / (double)xrange;
        ystep = 150.0 / (double)(max-min);

/* Boundary & Coordinate */
```

```
        _setcolor( 1 );
        _rectangle_w( _GBORDER, 0.0, -10.0, 200.0, 160.0 );
        for( i=1; i<20; i++ ){
                pos = 10.0*(double)i;
                _moveto_w( pos, -10.0 );
                _lineto_w( pos, -8.0 );
        }
        for( i=0; i<16; i++ ){
                pos = 10.0*(int)i;
                _moveto_w( 0.0, pos );
                _lineto_w( 2.0, pos );
        }

/* title for x axis */
        _settextposition( 16, 40 );
        _settextcolor( 4 );
        sprintf( txt, "%3.1f", (float)(xrange/(tsamp*2.0)) );
        /*_outtext( txt );*/
        _settextposition( 16, 75 );
        _settextcolor( 4 );
        sprintf( txt, "%3.1f", (float)(xrange/tsamp) );
        /*_outtext( txt ); */
        _settextposition( 27, 70 );
        _settextcolor( 7 );
        _outtext( "time" );
/* title for y axis */
        _settextposition( 4, 3 );
        _outtext( "Magnitude" );
/* main title */
        _settextposition( 4, 32 );
        _outtext( "The step response");
/* draw the ref voltage & the previous voltage */
        _setcolor( 2 );
        _setlinestyle( 0x00ff );
        _moveto_w( 0.0, ((double)ref-(double)min)*ystep );
        _lineto_w( 200.0, ((double)ref-(double)min)*ystep );
        _moveto_w( 0.0, ((double)prev-(double)min)*ystep );
        _lineto_w( 200.0, ((double)prev-(double)min)*ystep );
/* real data graph */
        _setcolor( 5 );
        _setlinestyle(0xffff);
        for( i=0; i<1000; i++ ){
                x=xstep*(double)i;
                _moveto_w( x, ((double)(*yval)-(double)min)*ystep );
                yval++;
                x=xstep*(double)(i+1);
                _lineto_w( x, ((double)(*yval)-(double)min)*ystep );
        }
}

/*****************************************************************************
*
*       void store_stepresp( unsigned long *arg1 )
*
*       Description:    Store the step response data in the user input file.
*
*       Input:          1. The data to be stored.
*
*       Return:         None
*
*/
void store_stepresp( unsigned long *RespData )
{
```

```
        char    *fn;
        FILE    *fp;
        int     i, j;

        fn = get_string( "file name" );
        fp = fopen( fn, "w" );
        for( i = 0; i < 1000; i++ ){
                        fprintf( fp, "%7d ",RespData[i]  );
        }
        free( fn );
        fclose( fp );
}
```

C.7 Data Interpretation Mode

```
/*****************************************************************************
*
*       void data_interpret_mode( void )
*
*       Description:    Prints the menu for the data interpretation mode, takes
*                       user input, and execute the selection.
*
*       Return:         None
*
*/
void data_interpret_mode( void )
{
        int             flag, n, ans;

        while( flag != 0 ){
                printf( "Data Interpretation Mode\n");
                printf( "       0. Data Retrieve\n" );
                printf( "       1. Data Save\n" );
                printf( "       2. Grey Scale Display\n" );
                printf( "       3. Retrieve and Grey Scale Display of the Data after
Filtering\n" );
                printf( "       4. Log Conversion\n" );
                printf( "       5. End the session\n" );
                n = get_int( "your choice", 0, 5 );
                switch( n ){
                        case 0:
                                retrieve_atomsurf();
                                break;
                        case 1:
                                store_atomsurf();
                                break;
                        case 2:
                                display_atomsurf();
                                getch();
                                _clearscreen( _GCLEARSCREEN );
                                _setvideomode( _DEFAULTMODE );
                                break;
                        case 3:
                                display_atomsurf_f();
                                getch();
                                _clearscreen( _GCLEARSCREEN );
                                _setvideomode( _DEFAULTMODE );
                                break;
                        case 4:
                                convert_to_log();
                                break;
                        case 5:
```

```
                        ans=waitYN("Termination of the session -
sure?(Y/N/<Enter>)",'N');
                        if(ans=1)flag = 0;
                        break;
                default:
                        printf( "Wrong input.\n");
                        break;
            }
        }
}
```

C.8 DSP Board Interface Subroutines

```
/*************************************************************************
*
*       int ctohex( char arg1 )
*
*       Description:    Convert the charater value to hex value.
*
*       Input:          Input character value.
*
*       Return:         the converted hex value.
*
*/

static unsigned int sa;
static unsigned int nb;
static int memtype;

int ctohex (char c)
{
       if ((c>='A') && (c<='F')) return (c-'A'+10);
       else return c-48;
}

/*************************************************************************
*
*       unsigned int getword( FILE *fp )
*
*       Description:    Catch four chars. from the file and return the value.
*
*       Input:          File pointer( .stk file )
*
*       Return:         16-bit unsigned int value
*
*/
unsigned int getword ( FILE *fp )
{
       unsigned char   dummy;
       int             temp;

       dummy = getc( fp );
       temp=ctohex(dummy)*256*16;
       dummy = getc( fp );
       temp=temp + ctohex(dummy)*256;
       dummy = getc( fp );
       temp=temp + ctohex(dummy)*16;
       dummy = getc( fp );
       temp=temp + ctohex(dummy);
       return(temp);
}
```

```
/*****************************************************************************
*
*       void dmget( FILE *arg1, struct HEX40 *arg2 )
*
*       Description:    Read the info. from the specified file and store in
*                       HEX 40 structure variable.
*
*       Input:          1. File pointer for the file to be read
*                       2. The HEX40 format, the place where data to be stored.
*
*       Return:         None
*
*/
void dmget ( FILE *fp, struct HEX40 *inst)
{
       unsigned char dummy;
       int temp;

       inst->USW=getword( fp );
       inst->MSW=getword( fp );
       dummy = getc( fp );
       temp=ctohex(dummy);
       dummy = getc( fp );
       inst->LSB=temp*16+ctohex(dummy);
       dummy = getc( fp );              /* reading newline char */
}

/*****************************************************************************
*
*       void pmget( FILE *arg1, struct HEX48 *arg2 )
*
*       Description:    Read the info. from the specified file and store in
*                       HEX 48 structure variable.
*
*       Input:          1. File pointer for the file to be read
*                       2. The HEX48 format, the place where data to be stored.
*
*       Return:         None
*
*/
void pmget (FILE *fp, struct HEX48 *inst)
{
       unsigned char dummy;

       inst->USW = getword( fp );
       inst->MSW = getword( fp );
       inst->LSW = getword( fp );
       dummy = getc( fp );
}

/*****************************************************************************
*
*       void getheader( FILE *arg1 )
*
*       Description:    Get header part in the .stk file. Store necessary info.
*
*       Input:          file to be read.
*       Return:         None.
*
*/
void getheader ( FILE *fp )
{
       unsigned char dummy;
       long temp;
```

```
        int n;

        dummy = getc( fp );       /* width of address and length fields */
        dummy = getc( fp );
        dummy = getc( fp );       /* reserved */
        dummy = getc( fp );

        dummy = getc( fp );         /* get two bytes of memory type */
        temp=ctohex(dummy)*16;
        dummy = getc( fp );
        memtype=(int)(temp)+ctohex(dummy);

        dummy = getc( fp );       /* user defined flags */
        dummy = getc( fp );

        temp=0;                 /* the next 8 chars are the start adr*/
        for (n=7; n>=0; n--){
               dummy = getc( fp );
               temp=temp+((long)(ctohex(dummy)) << (4*n));
        }
        sa=temp;

        temp=0;                 /* the last 8 chars are the # of bytes */
        for (n=7; n>=0; n--){
               dummy = getc( fp );
               temp=temp+((long)ctohex(dummy) << (4*n));
        }
        nb=temp;
        dummy = getc( fp );       /* reading the newline char */
}

/*************************************************************************
*
*       void dnld( void )
*
*       Description:  This module is used to download a .STK file developed on the PC
*                     to the PM and DM scratch rams of the BDSP-801.
*                     See the 21020  software development manual for a description of
the .STK
*                     file content, which is the byte-stacked format.
*
*       Return:         None
*
*/
void dnld( void )
{
        int n, i, count;
        int err, dummy, dum;
        char fname[40];
        FILE *fp;
        struct HEX40 dmarr[2000];
        struct HEX48 pmarr[2000];

        printf("enter file name:  \n");
        scanf( "\n", &dum );
        gets(fname);
        fp = fopen( fname, "r" );
        if (!fp) {
               printf("invalid file name....get some sleep and try again\n");
               goto end;
        }
        for( i = 1; i < 3; i++ ){
lp:      getheader(fp);
        printf( "The number of bytes = %d\n", nb );
```

```
        if (nb==0) goto end;
        if (memtype==0x00) {
                for (n=0; n<(nb/5); n++){
                      count++;
                       dmget( fp, dmarr+n );
                }
                err=loadmem_dm(sa,nb/5,dmarr,1);
        }
        if (memtype==0x80) {
                for (n=0; n<(nb/6); n++){
                      count++;
                       pmget( fp, pmarr+n );
                }
                err=loadmem_pm(sa,nb/6,pmarr,0);
        }
        goto lp;
        }
        end:printf( "line# = %d\n", count );
        printf("The program has been downloaded....\n");
}

/*****************************************************************************
*
*       void down( char *arg1 )
*
*       Description:    This is another version of dnld in which user inputs
*                       the file name.
*
*       Input:          String( char* )
*
*       Return:         None
*
*/
/* Maximum number of line it can get from the .stk file is 1000 */
void down( char *file_name )
{
        int n, i, count;
        int err, dummy, dum;
        FILE *fp;
        struct HEX40 dmarr[1000];
        struct HEX48 pmarr[1000];

        fp = fopen( file_name, "r" );
        if (!fp) {
               printf("error in reading data from the file %s\n", file_name);
               exit( 1 );
        }
        for( i = 1; i < 3; i++ ){
lp:             getheader(fp);
               if (nb==0) goto end;
               if (memtype==0x00) {
                      for (n=0; n<(nb/5); n++){
                             count++;
                             dmget( fp, dmarr+n );
                      }
                      err=loadmem_dm(sa,nb/5,dmarr,1);
               }
               if (memtype==0x80) {
                      for (n=0; n<(nb/6); n++){
                             count++;
                             pmget( fp, pmarr+n );
                      }
                      err=loadmem_pm(sa,nb/6,pmarr,0);
               }
```

```
                goto lp;
        }
        fclose( fp );
        end:;
}

/*****************************************************************************
*
*       void down( char *arg1 )
*
*       Description:    This is another version of dnld in which user inputs
*                       the file pointer.
*
*       Input:          FILE *
*
*       Return:         None
*
*/
void downfp( FILE *fp )
{
        int n, i, count;
        int err, dummy, dum;
        struct HEX40 dmarr[2000];
        struct HEX48 pmarr[2000];

        for( i = 1; i < 3; i++ ){
lp:             getheader(fp);
                if (nb==0) goto end;
                if (memtype==0x00) {
                        for (n=0; n<(nb/5); n++){
                                count++;
                                dmget( fp, dmarr+n );
                        }
                        err=loadmem_dm(sa,nb/5,dmarr,1);
                }
                if (memtype==0x80) {
                        for (n=0; n<(nb/6); n++){
                                count++;
                                pmget( fp, pmarr+n );
                        }
                        err=loadmem_pm(sa,nb/6,pmarr,0);
                }
                goto lp;
        }
        end:;
}

unsigned int cntlword=0x0;

/* routine to release reset on the bdsp-801 */

/*****************************************************************************
*
*       void run( void )
*
*       Description:    Execute the DSP code downloaded in the program memory.
*                                                       The code is executed from
the memory location, 0x0.
*                                                       The module sets the 1st bit
0x1 in control register of DSP board.
*
*       Return:         None
*
*/
```

```
void run (void) {
       cntlword=cntlword | 0x01;
       outp( CNTL, cntlword );
}

/*****************************************************************************
*
*       void reset( void )
*
*       Description:    Reset the DSP board. The module toggles the first bit of
control
*                                                       register to 0x0 to stop
DSP's current operation.
*
*       Return:         None
*
*/
void reset (void){
       cntlword=cntlword & 0xFE;
       outp (CNTL,cntlword); }

/*****************************************************************************
*
*       void stat( void )
*
*       Description:    Print the status register value on the screen.
*
*       Return:         None
*
*/
void stat (void) {
       int st;
       st=inpw(STAT);
       printf("The status word is: %x\n",st);}

/*****************************************************************************
*
*       int statreg( void )
*
*       Description:    Read the status register value and the return the value.
*
*       Return: the status register value in 16 bit integer.
*
*/
int statreg (void) {
       int st;
       st=inpw(STAT);
       return(st); }

/*****************************************************************************
*
*       void timing( void )
*
*       Description:    Print the timing register value on the screen.
*
*       Return:         None
*
*/
void timing (void) {
       int tm;
       tm=inpw(TMG);
       printf("The timing register is: %x\n",tm);}

/*****************************************************************************
```

```
*
*       int loadmem_i(unsigned int arg1, unsigned int arg2,
*                     unsigned int *arg3, unsigned int arg4)
*
*       Description:    This routine is used if the memory transfer is a write
*                       from an array of 16-bit fixed-point values, represented as
*                       unsigned integers.
*
*       Input:          1. The starting address in DSP board where the data go to.
*                       2. The number of the data.
*                       3. The data array.
*                       4. The bank number.( either Data Mem(1) or Program Mem(0) )
*
*       Return:         err=1 if the error in bank input isn't correct.
*                       err=0 otherwise.
*
*/
int loadmem_i(unsigned int start_adr, unsigned int num_words,
          unsigned int * datarray, unsigned int bank)
{  int rw;
   int k;
   int err=0;
   int reg;
   int adreg;
   int adr;
   switch (bank) {
        case 1: reg=DMDM; adreg=DMAD; rw=DMWR; break;
        case 0: reg=PMDM; adreg=PMAD; rw=PMWR; break;
        default : err=1;
                  goto done;}
        adr=start_adr;
        for (k=0; k<num_words; k++)
        {
              outpw(adreg,adr);
              outpw(reg,datarray[k]);
              outpw(rw,0x0);
              adr++;
        }
done: return(err);
}

/*************************************************************************
*
*       int loadmem_l(unsigned int arg1, unsigned int arg2,
*                     unsigned long *arg3, unsigned int arg4)
*
*       Description:    This routine is used if the memory transfer is a write
*                       from an array of 32 bit fixed-point values, represented as
*                       unsigned long.
*
*       Input:          1. The starting address in DSP board where the data go to.
*                       2. The number of the data.
*                       3. The data array.
*                       4. The bank number.( either Data Mem(1) or Program Mem(0) )
*
*       Return:         err=1 if the error in bank input isn't correct.
*                       err=0 otherwise.
*
*/
int loadmem_l(unsigned int start_adr, unsigned int num_words,
          unsigned long * datarray, unsigned int bank)
{  int rw;
   int k;
   int err=0;
```

```
   int regh;
   int regl;
   int adreg;
   int adr;
   switch (bank) {
        case 1: regl=DMDM; regh=DMDH; adreg=DMAD; rw=DMWR; break;
        case 0: regl=PMDM; regh=PMDH; adreg=PMAD; rw=PMWR; break;
        default : err=1;
                  goto done;}
        adr=start_adr;
        for (k=0; k<num_words; k++)
        {
              outpw(adreg,adr);
              outpw(regl,(int)(datarray[k] & 0xFFFF));
              outpw(regh,(int)((datarray[k] & 0xFFFF0000)>>16));
              outpw(rw,0x0);
              adr++;
        }
done: return(err);
}

/*************************************************************************
*
*       int loadmem_pm(unsigned int arg1, unsigned int arg2,
*                      struct HEX48 *arg3, unsigned int arg4)
*
*       Description:    This routine is used if the memory transfer is a write
*                       from an array of 48-bit instruction values, represented as
*                       struct HEX48.
*
*       Input:          1. The starting address in DSP board where the data go to.
*                       2. The number of the data.
*                       3. The 48 bit instruction value array.
*                       4. The bank number.( either Data Mem(1) or Program Mem(0) )
*
*       Return:         err=1 if the error in bank input isn't correct.
*                       err=0 otherwise.
*
*/
int loadmem_pm(unsigned int start_adr, unsigned int num_words,
                          struct HEX48 *datarray, unsigned int bank)
{ int k;
  int err=0;
  int adr;
  if (bank==0) {
       adr=start_adr;
       for (k=0; k<num_words; k++){
             outpw(PMAD,adr);
             outpw(PMDL,datarray[k].LSW);
             outpw(PMDM,datarray[k].MSW);
             outpw(PMDH,datarray[k].USW);
             outpw(PMWR,0x00);
             adr++;
       }
       goto done;
  }
  err=1;
  done:  return(err);
}

/*************************************************************************
*
*       int loadmem_dm(unsigned int arg1, unsigned int arg2,
*                      struct HEX40 *arg3, unsigned int arg4)
```

```
*
*       Description:    This routine is used if the memory transfer is a write
*                       from an array of 40 bit DM values, represented as
*                       struct HEX40.
*
*       Input:          1. The starting address in DSP board where the data go to.
*                       2. The number of the data.
*                       3. The 40 bit data value array.
*                       4. The bank number.( either Data Mem(1) or Program Mem(0) )
*
*       Return:         err=1 if the error in bank input isn't correct.
*                       err=0 otherwise.
*
*/
int loadmem_dm(unsigned int start_adr, unsigned int num_words,
                          struct HEX40 * datarray, unsigned int bank)
{ int k;
  int err=0;
  int rw;
  int hreg;
  int mreg;
  int lreg;
  int adreg;
  int adr;
  switch (bank) {
       case 0: hreg=PMDH; mreg=PMDM; lreg=PMDL; rw=PMWR; adreg=PMAD; break;
       case 1: hreg=DMDH; mreg=DMDM; lreg=DMDL; rw=DMWR; adreg=DMAD; break;
       default: err=1; goto done;
  }
  adr=start_adr;
  for (k=0; k<num_words; k++){
         outpw(adreg,adr);
         outpw(lreg,datarray[k].LSB);
         outpw(mreg,datarray[k].MSW);
         outpw(hreg,datarray[k].USW);
         outpw(rw,0x00);
         adr++;
  }
  done: return(err);
}

/*************************************************************************
*
*       int loadmem_f(unsigned int arg1, unsigned int arg2,
*                       float *arg3, unsigned int arg4)
*
*       Description:    This routine is used if the memory transfer is a write
*                       from an array of 32-bit floating values, represented as
*                       float.
*
*       Input:          1. The starting address in DSP board where the data go to.
*                       2. The number of the data.
*                       3. The floating point data array.
*                       4. The bank number.( either Data Mem(1) or Program Mem(0) )
*
*       Return:         err=1 if the error in bank input isn't correct.
*                       err=0 otherwise.
*
*/
int loadmem_f(unsigned int start_adr, unsigned int num_words,
                          float *datarray, unsigned int bank)
{ int k;
  int err=0;
  int rw;
```

```
  int hreg;
  int lreg;
  int adreg;
  int adr;
  union f_l temp;
  unsigned int dh;
  unsigned int dl;

  switch (bank) {
       case 0: hreg=PMDH; lreg=PMDM; rw=PMWR; adreg=PMAD; break;
       case 1: hreg=DMDH; lreg=DMDM; rw=DMWR; adreg=DMAD; break;
       default: err=1; goto done;
  }
  adr=start_adr;
  for (k=0; k<num_words; k++){
         temp.f=datarray[k];
         dl = (int)( temp.l & 0xffff );
         dh = (int)(( temp.l & 0xffff0000 )>> 16);
         outpw(adreg,adr);
         outpw(lreg,dl);
         outpw(hreg,dh);
         outpw(rw,0x00);
         adr++;
  }
  done: return(err);
}

/*************************************************************************
*
*       int readmem_pm(unsigned int arg1, unsigned int arg2,
*                      struct HEX48 *arg3, unsigned int arg4)
*
*       Description     This routine is used if the memory transfer is a read
*                       from the bdsp801 to an array of 48-bit DSP instruction
values,
*                       represented as the class HEX48.
*
*       Input:          1. The starting address in DSP board where the data are read.
*                       2. The number of the data.
*                       3. The 48-bit instruction value array where the data is to be
stored.
*                       4. The bank number.( either Data Mem(1) or Program Mem(0) )
*
*       Return:         err=1 if the error in bank input isn't correct.
*                       err=0 otherwise.
*
*/
int readmem_pm(unsigned int start_adr, unsigned int num_words,
                          struct HEX48 * datarray, unsigned int bank)
{ int k;
  int err=0;
  int adr;
  int tmp;
  if (bank==0) {
       adr=start_adr;
       for (k=0; k<num_words; k++){
              outpw(PMAD,adr);
              tmp=inpw(PMRD);
              datarray[k].LSW=inpw(PMDL);
              datarray[k].MSW=inpw(PMDM);
              datarray[k].USW=inpw(PMDH);
              adr++;
       }
       goto done;
```

```
  }
  err=1;
  done:  return(err);
}

/*************************************************************************
*
*       int readmem_dm(unsigned int arg1, unsigned int arg2,
*                      struct HEX40 *arg3, unsigned int arg4)
*
*       Description     This routine is used if the memory transfer is a read
*                       from the bdsp801 to an array of 40-bit DSP data,
*                       represented as the class HEX40.
*
*       Input:          1. The starting address in DSP board where the data are read.
*                       2. The number of the data.
*                       3. The 40-bit data array where the data is to be stored.
*                       4. The bank number.( either Data Mem(1) or Program Mem(0) )
*
*       Return:         err=1 if the error in bank input isn't correct.
*                       err=0 otherwise.
*
*/
int readmem_dm(unsigned int start_adr, unsigned int num_words,
                    struct HEX40 * datarray, unsigned int bank)
{ int k;
  int err=0;
  int rw;
  int hreg;
  int mreg;
  int lreg;
  int adreg;
  int adr;
  int tmp;
  switch (bank) {
       case 0: hreg=PMDH; mreg=PMDM; lreg=PMDL; rw=PMRD; adreg=PMAD; break;
       case 1: hreg=DMDH; mreg=DMDM; lreg=DMDL; rw=DMRD; adreg=DMAD; break;
       default: err=1; goto done;
  }
  adr=start_adr;
  for (k=0; k<num_words; k++){
         outpw(adreg,adr);
         tmp=inpw(rw);
         datarray[k].LSB=inpw(lreg);
         datarray[k].MSW=inpw(mreg);
         datarray[k].USW=inpw(hreg);
         adr++;
  }
  done:  return(err);
}

/*************************************************************************
*
*       int readmem_l(unsigned int arg1, unsigned int arg2,
*                      unsigned long *arg3, unsigned int arg4)
*
*       Description     This routine is used if the memory transfer is a read
*                       from the bdsp801 to an array of 32-bit fixed point value,
*                       represented as the unsigned long.
*
*       Input:          1. The starting address in DSP board where the data are read.
*                       2. The number of the data.
*                       3. The 32-bit fixed point data array where the data is
stored.
```

```
*                       4. The bank number.( either Data Mem(1) or Program Mem(0) )
*
*       Return:         err=1 if the error in bank input isn't correct.
*                       err=0 otherwise.
*
*/
int readmem_l(unsigned int start_adr, unsigned int num_words,
          unsigned long * datarray, unsigned int bank)
{ int k;
  int err=0;
  int rw;
  int hreg;
  int lreg;
  int adreg;
  unsigned dh;
  unsigned dl;
  int adr;
  int tmp;
  switch (bank) {
       case 0: hreg=PMDH; lreg=PMDM; rw=PMRD; adreg=PMAD; break;
       case 1: hreg=DMDH; lreg=DMDM; rw=DMRD; adreg=DMAD; break;
       default: err=1; goto done;
  }
  adr=start_adr;
  for (k=0; k<num_words; k++) {
         outpw(adreg,adr);
         tmp=inpw(rw);
         dl=inpw(lreg);
         dh=inpw(hreg);
         datarray[k]=((long)(dh)<<16) + dl;
         adr++;
        }
  done:  return(err);
}

/*************************************************************************
*
*       int readmem_i(unsigned int arg1, unsigned int arg2,
*                       unsigned int *arg3, unsigned int arg4)
*
*       Description     This routine is used if the memory transfer is a read
*                       from the bdsp801 to an array of 16-bit fixed point value,
*                       represented as the unsigned int.
*
*       Input:          1. The starting address in DSP board where the data are read.
*                       2. The number of the data.
*                       3. The 16-bit fixed point data array where the data is
stored.
*                       4. The bank number.( either Data Mem(1) or Program Mem(0) )
*
*       Return:         err=1 if the error in bank input isn't correct.
*                       err=0 otherwise.
*
*/
int readmem_i(unsigned int start_adr, unsigned int num_words,
          unsigned int * datarray, unsigned int bank)
{ int k;
  int err=0;
  int rw;
  int reg;
  int adreg;
  unsigned dh;
  unsigned dl;
  int adr;
```

```
  int tmp;
  switch (bank) {
       case 0: reg=PMDM; rw=PMRD; adreg=PMAD; break;
       case 1: reg=DMDM; rw=DMRD; adreg=DMAD; break;
       default: err=1; goto done;
  }
  adr=start_adr;
  for (k=0; k<num_words; k++){
         outpw(adreg,adr);
         tmp=inpw(rw);
         datarray[k]=inpw(reg);
         adr++;
        }
  done:  return(err);
}

/*************************************************************************
*
*       int readmem_f(unsigned int arg1, unsigned int arg2,
*                     float *arg3, unsigned int arg4)
*
*       Description     This routine is used if the memory transfer is a read
*                       from the bdsp801 to an array of 32-bit floating point value,
*                       represented as the float.
*
*       Input:          1. The starting address in DSP board where the data are read.
*                       2. The number of the data.
*                       3. The 32-bit floating point data array where the data is
stored.
*                       4. The bank number.( either Data Mem(1) or Program Mem(0) )
*
*       Return:         err=1 if the error in bank input isn't correct.
*                       err=0 otherwise.
*
*/
int readmem_f(unsigned int start_adr, unsigned int num_words,
              float * datarray, unsigned int bank)
{ int k;
  int err=0;
  int rw;
  int hreg;
  int lreg;
  int adreg;
  int adr;
  int tmp;
  unsigned int dh;
  unsigned int dl;
  union f_l dattmp;

  switch (bank) {
       case 0: hreg=PMDH; lreg=PMDM; rw=PMRD; adreg=PMAD; break;
       case 1: hreg=DMDH; lreg=DMDM; rw=DMRD; adreg=DMAD; break;
       default: err=1; goto done; }
  adr=start_adr;
  for (k=0; k<num_words; k++) {
         outpw(adreg,adr);
         tmp=inpw(rw);
         dl=inpw(lreg);
         dh=inpw(hreg);
         dattmp.l = ((long)dh << 16) +dl;
         datarray[k] = dattmp.f;
         adr++;
       }
```

```
  done: return(err);
}
```

C.9 Programming Support Routines

```
/*****************************************************************************
*
*       int waitYN( char *arg1, char arg2 )
*
*       Description:    Wait for the user to press Y or N.
*
*       Input:          1. prompt
*                       2. default value Y or N.
*
*       Return:         Selected value.
*
*/
int waitYN( char *prompt, char defaultValue)
{
       char key;

       printf("%s", prompt);
       do{
              key=toupper(getch());
              switch (key){
                     case '\r':      key=defaultValue; break;
                     case '0':       key='N'; break;
                     case '1':       key='Y'; break;
              }
       }while(!((key=='Y')||(key=='N')));
       printf("%c\n",key);
       if(key=='Y')
              return 1;
       return 0;
}

/*****************************************************************************
*
*       get_string - get string from user with prompt
*
*       Return pointer to string of input text, prompts user with string passed
*       by caller. Indicates error if string space could not be allocated. Limited
*       to 80 char input.
*
*       char* get_string( char* prompt_string )
*
*       prompt_string string to prompt user for input
*
*/
char *get_string( char* title_string )
{
       char* alpha;

       alpha = ( char* )malloc( 80 );
       if( !alpha ){
              printf( "\nString allocation error in get_string\n" );
              exit( 1 );
       }
       printf("\nEnter %s : ", title_string );
       gets( alpha );

       return( alpha );
}
```

```
/*************************************************************************
*
*       get_int - get integer from user with prompt and range
*
*       Return integer of input text, prompts user with prompt string and range of
*       values ( upper and lower limits ) passed by caller.
*
*       int get_int( char * title_string, int low_limit, int up_limit )
*
*       title_string    string to prompt user for input
*       low_limit       lower limit of acceptable input ( int )
*       up_limit        upper limit of acceptable input ( int )
*
*/
int get_int( char* title_string, int low_limit, int up_limit )
{
       int     i, error_flag;
       char*   get_string( char* input );
       char    *cp, *endcp;
       char    *stemp;

/* check for limit error, low may equal high but not greater */
       if( low_limit > up_limit ){
              printf( "\nLimit error, lower > upper\n");
              exit( 1 );
       }
       stemp = ( char* )malloc( strlen( title_string ) + 60 );
       if( !stemp ){
              printf( "\nString allocation error in get_int\n" );
              exit( 1 );
       }
       sprintf( stemp, "%s [%d...%d]", title_string, low_limit, up_limit );
/* get the string and make sure i is in range and valid */
       do{
              cp = get_string( stemp );
              i = ( int )strtol( cp, &endcp, 10 );
              error_flag = ( cp == endcp ) || ( *endcp != '\0' ); /*detect errors */
              free( cp );
       } while( i < low_limit || i > up_limit || error_flag );

/* free temp string and return result */
       free( stemp );
       return( i );
}

/*************************************************************************
*
*       get_long - get unsigned long from user with prompt and range
*
*       Return integer of input text, prompts user with prompt string and range of
*       values ( upper and lower limits ) passed by caller.
*
*       int get_long( char * title_string, unsigned long low_limit,
*                                               unsinged long int up_limit )
*
*       title_string    string to prompt user for input
*       low_limit       lower limit of acceptable input ( int )
*       up_limit        upper limit of acceptable input ( int )
*
*/
unsigned long get_long( char* title_string, unsigned long low_limit, unsigned long
up_limit )
{
```

```
        unsigned long   i, error_flag;
        char*           get_string( char* input );
        char            *cp, *endcp;
        char            *stemp;

/* check for limit error, low may equal high but not greater */
        if( low_limit > up_limit ){
                printf( "\nLimit error, lower > upper\n");
                exit( 1 );
        }
        stemp = ( char* )malloc( strlen( title_string ) + 60 );
        if( !stemp ){
                printf( "\nString allocation error in get_int\n" );
                exit( 1 );
        }
        sprintf( stemp, "%s [%lu...%lu]", title_string, low_limit, up_limit );
/* get the string and make sure i is in range and valid */
        do{
                cp = get_string( stemp );
                i = strtoul( cp, &endcp, 10 );
                error_flag = ( cp == endcp ) || ( *endcp != '\0' ); /*detect errors */
                free( cp );
        } while( i < low_limit || i > up_limit || error_flag );

/* free temp string and return result */
        free( stemp );
        return( i );
}

/*****************************************************************************
*
*         get_float - get float from user with prompt and range
*
*        Return integer of input text, prompts user with prompt string and range of
*        values ( upper and lower limits ) passed by caller.
*
*        int get_int( char * title_string, int low_limit, int up_limit )
*
*        title_string    string to prompt user for input
*        low_limit       lower limit of acceptable input ( int )
*        up_limit        upper limit of acceptable input ( int )
*
*/
float get_float( char* title_string, float low_limit, float up_limit )
{
        int     error_flag;
        char*   get_string( char* input );
        char    *cp, *endcp;
        char    *stemp;
        float   f;

/* check for limit error, low may equal high but not greater */
        if( low_limit > up_limit ){
                printf( "\nLimit error, lower > upper\n");
                exit( 1 );
        }
        stemp = ( char* )malloc( strlen( title_string ) + 60 );
        if( !stemp ){
                printf( "\nString allocation error in get_float\n" );
                exit( 1 );
        }
        sprintf( stemp, "%s [%3.1f...%3.1f]", title_string, low_limit, up_limit );
/* get the string and make sure i is in range and valid */
        do{
```

```
            cp = get_string( stemp );
            f = atof( cp );
            free( cp );
        } while( f < low_limit || f > up_limit );

/* free temp string and return result */
        free( stemp );
        return( f );
}

/*****************************************************************************

The Scanning Tunneling Microscope Project

Laboratory for Manufacturing and Productivity
The Department of Mechanical Engineering
Massachusetts Institute of Technology

        Header File
        constm.h

Description  :

This has the prototypes of the subroutines used in the STM control program.
When you want to add the subroutine in the program, you should register
the function in this header file with a proper description.

Written by Jungmok Mitchell Bae
4/17/94

*****************************************************************************/
/* The Library functions provided by Microsoft Quick C */
#include <stdio.h>
#include <conio.h>
#include <graph.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <dos.h>

/* The menu functions */
int mainmenu(void);
void constant_height_mode( void );
void constant_current_mode( void );
void wide_scanning_mode( void );
void atomic_encoder_mode( void );
void data_interpret_mode( void );
void manual_scanning_mode( void );

/* The scanning tunneling microscope control functions */
void coarse_approach1_atomsurf( void );
void parameter_set( void );
void move_inchworm_motor( void );
void fine_approach_atomsurf( void );
void pidcontrol_atomsurf(void);
void logtable_download( void );

/* The constant height routines */
void ramp_scan_atomsurf(void);
void tri_scan_atomsurf( void );
void ramp_auto_scan_atomsurf( void );
void tri_auto_scan_atomsurf( void );

/* The constant current routines */
```

```
void cc_pidcontrol_atomsurf( void );
void cc_tri_scan_atomsurf(void);
void cc_ramp_scan_atomsurf(void);
void manual_pid_tuning( void );
void graph_init( void );
void plot( unsigned long *yval, int xrange, int tsamp, float ref, float prev );
void store_stepresp( unsigned long *RespData );

/* The data presentation functions */
void display_atomsurf( void );
void display_atomsurf_f( void );
void linear_interpo( float cn1, float cn2, float cn3, float cn4 );
void convert_to_log( void );

/* The data store & retrieve functions */
void store_atomsurf( void );
void retrieve_atomsurf( void );

/*support routines*/
int waitYN(char *prompt, char defaultValue );
```

Bibliography

[1] Mircea Fotino. "Tip Sharpening by Normal and Reverse Eletrochemical Etching" Review of Scientific Instruments, 64(1), 159, January, 1993.

[2] I.H.Musselman, P.A. Peterson and P.E. Russell. "Fabrication of Tips with Controlled Geometry for Scanning Tunneling Microscopy" Precision Engineering, Vol. 12, No. 1, January, 1990.

[3] J.P.Ibe, P.P.Bev.Jr, S.S.Brandow, R.A.Brizzolara, N.A.Burnham, D.P.DiLella, K.P.Lee, C.R.K.Marrian, and R.J.Colton. "On the Electrochemical Etching of Tips for Scanning Tunneling Microscopy" J. Vac. Sci. Technol. Surf, Films (USA), Vol. 8, No. 4, p. 3570-5, July-Aug, 1990.

[4] J.Tersoff and D.R.Hamann. "Theory of the Scanning Tunneling Microscope" Physical Review B, Vol. 31 No. 2, 15, January, 1985.

[5] J.Bardeen. "Tunneling from a Many-Particle Point of View" Physical Review Letters, Vol. 6, 57(1961).

[6] N.Garcia, C.Ocal, and F.Flores. "Model Theory for Scanning Tunneling Microscopy: Application to Au(110)(1×2)" Physical Review Letters, Vol. 50, No. 25, 20. June, 1983.

[7] Y.Kuk and P.J.Silverman. "Scanning Tunneling Microscope Instrumentation" Review of Scientific Instruments, 60(2), February, 1989.

[8] C.B.Duke. *Tunneling in Solids* Academic Press, New York, Vol. 10, 1969.

[9] G.Binnig and D.P.E. Smith. "Single-Tube-Dimensional Scanner for Scanning Tunneling Microscopy" Review of Scientific Instruments, 57(8), August, 1986.

[10] D.R.Baselt, S.M.Clark, M.G.Youngquist, C.F.Spence, and J.D.Baldeschwieler. "Digital Signal Processor Control of Scanned Probe Microscopes" Review of Scientific Instruments, 64(7), July, 1993.

[11] S.M.Clark, D.R.Baselt, C.F.Spence, M.G.Youngquist and J.D.Baldeschwieler. "Hardware for Digitally Controlled Scanned Probe Microscope" Review of Scientific Instruments, 63(10), October, 1992.

[12] B.A.Morgan and G.W.Stupian. "Digital Feedback Control Loop for Scanning Tunneling Microscopes" Review of Scientific Instruments, 62(12), December, 1991.

[13] P.Heuell, M.A.Kulakov, and B.Bullemer. "An Adaptive Scan Generator for a Scanning Tunneling Microsocpe" Review of Scientific Instruments, 65(1), January, 1994.

[14] R.D.Cutkosky. "Versatile Scan Generator and Data Collector for Scanning Tunneling Microscopes" Review of Scientific Instruments, 61(3), July, 1991.

[15] C.J.Chen. "Introduction to Scanning Tunneling Microscopy" Oxford University Press, Inc., 1993.

[16] J.Y.Pahng. "Surface Modification with a scanning Tunneling Microscope" M.S. Thesis, Massachusetts Institute of Technology, Cambridge, 1994, February.

[17] Analog Devices, Inc. *ADSP-21000 Family User's Manual* DSP Division, Norwood, Massachusetts, 1991.

[18] P.Horowitz, H.Winfield. *The Art of Electronics* Second Edition, Cambridge University Press, New York, 1989.

[19] Analog Devices, Inc. *ADSP-21000 Family Assembler Tools and Simulator Manual* DSP Division, Norwood, Massachusetts, 1991.

[20] J.Gort. *21020 DSP Board User's Manual* J.R.G Signal Processing, 1993.

[21] L.L.Soethout. J.W.Gerritsen. P.P.M.C.Groeneveld, B.J.Nelissen and H. Van Kempen. "STM Measurement on Graphite Using Correlation Averaging of the Data" Journal of Microscopy, Vol. 152, Pt. 1, pp. 251-258, October, 1988.

[22] S.Park, C.Quate. "Tunneling Microscopy of Graphite in Air" Appl.Phys.Lett.48(2), 13. January, 1986.

[23] K.Kobayashi and M.Tsukada, "Effect of Microscopic Tip Electronic State on STM Image of Graphite" Journal of the Physical Society of Japan, Vol. 58, No. 7, July, 1989. pp. 2238-2241.

[24] A.Selloni. P.Carnevali. E.Tosatti, and C.D.Chen, in *Proceedings of the 17th International Conference on the Physics of Semiconductors*, edited by J.D.Chadi and W.A.Harrison (Springer, New York, 1985), P.11.

[25] D.Pohl, R.Moller, "Tracking Tunneling Microscopy" Review of Scientific Instruments, 59(6), 1988.

[26] R.Weast, *Handbook of Chemistry and Physics* 54th Edition, CRC Press, Cleveland, 1973.