7-12-1994

Interfacing a Computer to a Scanning Tunneling Microscope

Markus Jarasch
Portland State University

Recommended Citation

Jarasch, Markus, "Interfacing a Computer to a Scanning Tunneling Microscope" (1994). *Dissertations and Theses.* Paper 5047.
https://doi.org/10.15760/etd.6923

THESIS APPROVAL

The abstract and thesis of Markus Jarasch for the Master of Science in Physics were presented July 12, 1994, and accepted by the thesis committee and the department.

COMMITTEE APPROVALS: [redacted]
Erik Bodegom, Chair

[redacted]
Carl G. Bachhuber

[redacted]
Pavel K. Smejtek

[redacted]
Stanley S. Hillman
Representative of the Office of Graduate Studies

DEPARTMENTAL APPROVAL: [redacted]
Erik Bodegom, Chair
Department of Physics

****

ACCEPTED FOR PORTLAND STATE UNIVERSITY BY THE LIBRARY

by [redacted] on 12 December 1994

ABSTRACT

An abstract of the thesis of Markus Jarasch for the Master of Science in Physics presented July 12, 1994.

Title: Interfacing a Computer to a Scanning Tunneling Microscope

A program was written in 'C' to control the functions of an already existing Scanning Tunneling Microscope (STM). A DAS-1601 data acquisition card (from Keithley Data Acquisition) was installed together with its drivers for 'C' on a computer with a 486-DX motherboard. The computer was interfaced to the electronics of the STM. Images taken of HOPG (highly oriented pyrolitic graphite) were of a reasonable quality and showed atomic resolution.

INTERFACING A COMPUTER TO A SCANNING TUNNELING MICROSCOPE

by

MARKUS JARASCH

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

MASTER OF SCIENCE
in
PHYSICS

Portland State University
1995

ACKNOWLEDGMENTS

First and foremost, I want to thank Professor Erik Bodegom, who never got tired of encouraging me not to give up my project. He always took plenty of time for me and put lots of his own work into this project. He always found a solution when I was stuck. His patience, his noticeable courteousness and his friendliness were remarkable, even at the end of a long working day. These qualities created such a pleasant atmosphere that I always enjoyed working with him, talking to him about my project, and, above all, talking to him about the small details of life that make it so worth living.

I want to thank the Physics Department at Portland State University for the very friendly integration during my year as an exchange student, especially Margie Fyfield, who spent time and work on my project and Frances Boyd, who always brought sunshine to the office of the Physics Department with her cheerful and happy nature.

I also want to thank my roommates, who opened my mind to new ideas and new paths during late night discussions. They showed avid interest in the progress of my research and bought me a very useful tool that helped me with it. Last but not least, they never let me lose my joy in studying at Portland State University.

Finally, I want to thank my parents very much for their continuous support of my education and for making this year in the US possible for me.

TABLE OF CONTENTS

PAGE

CHAPTER

APPENDIX

LIST OF FIGURES

FIGURE PAGE

Chapter 1 Microscopy

Light Microscopes

It has always been a dream of scientists and nonscientists alike to see single atoms. However this was not possible until recently. About 100 years ago the German physicist Ernst Abbe stated that for any microscope that relies on lenses to focus light or other radiation, diffraction obscures details that are smaller than about one half the wavelength of the radiation. As optical instruments use visible light whose average wavelength is about 2000 times greater than the diameter of a typical atom; optical microscopes cannot resolve atomic structures (resolution power > 200 nm).

Electron Microscopes

This limitation led to the invention of the electron microscopes. According to quantum mechanics, light and other kinds of energy, such as electrons, show characteristics of both particles and waves. Therefore every electron has a certain wavelength according to its energy. Electrons with a lower energy possess a longer wavelength, electrons with higher energy a shorter wavelength. The principle of operation of an electron microscope is similar to that of an optical microscope, except that light waves are replaced by short wavelength electrons and glass lenses by electromagnetic ones. By using high-energy-electrons in TEMs a resolving power of about 0.2 nm could be achieved, that means atomic resolution.

However, the electron microscope (EM) has also some practical limitations and disadvantages:

- EM normally cannot resolve surface structures.
- As the EM requires operations in vacuum it is not possible to examine samples that produce any significant amount of vapor when placed in vacuum. Therefore biological samples must be dried and since water molecules are an important part of biological samples, this might change the sample in an undesirable way.
- The specimen is exposed to high-energy-radiation that might damage or alter the specimen.

Hence there is a need for another type of microscope. This turns out to be the scanning tunneling microscope.

Scanning Tunneling Microscopes

The principle of operation of a new generation of microscopes, the scanning tunneling microscope, makes it possible to avoid these difficulties. In 1982 Gerd Binnig and Heinrich Rohrer of the IBM Zurich Research Laboratory reported about this new kind of microscopy. The main difference between this microscope and all other ones is that it uses no free particles.[1,2] Consequently, there is no need for lenses and light or electron sources. Instead the bound electrons already existing in the conducting sample under investigation serve as the source of radiation.

The STM proved to be such a powerful tool in the field of surface science that Binnig and Rohrer received a Nobel prize just four years later in 1986. The STM makes use of a common phenomenon on the atomic scale called "tunneling" wherein their quantum nature permits electrons to penetrate (tunnel) into classically forbidden regions of space where the particle potential exceeds its total energy. At a sufficiently great distance from the nucleus of an atom, such a forbidden region exists for all bound electrons. The chance of finding an electron beyond the surface of a conductor decreases exponentially with increasing distance from the nucleus. If we position the tip of an atomically sharp metallic needle so close to the surface of the conducting material that the electron clouds of the atom at the probe tip and of the nearest atom of the specimen overlap and put a small potential difference between the two, electrons "tunnel" across the gap, which generates a tunneling current.

The tunneling current behaves as

$$I \propto \exp(-2\kappa d)$$

I: tunneling current

d: distance between tip and surface

κ: decay constant

and

$$\kappa = 1/\hbar * \sqrt{2m\Phi}$$

Φ: local work function

m: electron mass

$\hbar$: Planck's constant

That means the tunneling current is exquisitely sensitive to the width of the gap between the probe and the surface. For a gap distance of 5 Å, an applied bias of several tens of millivolts will induce nanoampere current flows. For a typical work function of 4 eV, $\kappa = 1.0$ Å^{-1}, the current decreases by the factor 10 when the gap d is increased by 1 Å. If the tunneling current is kept constant to within, say 2%, then the gap remains constant to within 0.01 Å.

In the constant height mode of operation the tip is moved back and forth across the specimen surface in a raster pattern by X and Y piezoelectric controls. The drive of the first STM of Binnig and Rohrer consisted of a tripod made of piezoelectric ceramics which contract or expand when voltage is applied, yielding a displacement control better than 0.1 Å. As the probe is maintained at a steady height, the current fluctuates dramatically, increasing when the tip passes over bumps such as surface atoms and decreasing as it crosses gaps between atoms. The plot of the tunneling current as a function of the X and Y piezo voltages is a kind of tunneling replica of the sample surface, which we refer to as a tunneling image. The data is processed by a computer and displayed on a screen or a plotter.

In the constant current mode a feedback mechanism senses the variations in tunneling current and varies the voltage applied to the Z piezo control to keep the current constant and maintain a constant gap between the probe and the surface. In this mode the feedback controlled Z piezo voltage is recorded as a function of the X and Y piezo voltages.

The constant current mode was first historically. Its advantage is the fact that one can track surfaces that are not automatically flat without scratching the tip. The "constant height mode" requires automatically flat surfaces because the tip is scanned across a surface at constant height and constant voltage while the current is monitored or at constant current while the voltage is monitored. This mode allows for much faster imaging (of atomically flat surfaces) since only the electronics, not the Z-control, musts respond to the atoms passing under the tip. This enables researchers to study processes in real time as well as reducing data collection time.

The tip preparation plays a crucial role in scanning tunneling microscopy as the shape of the tip limits the lateral resolution. It is typically a tungsten wire either ground on a grindstone or etched to a radius in the range 0.1-10 μm. To obtain atomic resolution the tip must have a single atom sitting securely on its top. However this situation is currently obtained only by luck. Paul K. Hansma[3] mentioned in his article about scanning tunneling microscopy three recipes to obtain such tips:

(1) Withdraw the tip more than 1000 Å from the sample. Apply a high enough voltage (of order of several hundred volts) to get a field emission current of a few microampere, and then turn down as soon as any sudden jump in the current occurs (useful only in high vacuum).

(2) Apply an oscillating voltage at ~1 kHz to the piezo in addition to the feedback voltage. Increase the magnitude of the oscillating voltage until there is an abrupt change in the feedback voltage.

(3) Set up in a tunneling configuration with the tip scanning over the surface and just wait about 10 min. to 1 h.

After the tip was formed by methods (1) or (2) it was prudent to move it laterally to a new region of the sample since these methods for tip forming could modify the surface directly under the tip.

More recently there is a growing consensus that an etched tip that appears sharp under a high-quality optical microscope (operated at 200x or above) will give atomic resolution almost every time either immediately or after a wait as listed in method (3) -- if it is not banged into the surface . 'Appears sharp' means that no radius can be observed at the end with the optical microscope. This in turn means that the tip radius is <2000 Å.[3]

Atomic Force Microscopes

While the STM is largely restricted to imaging electrical conductors, another kind of scanned probe microscope -- the atomic force microscope -- does not need a conducting specimen. This device also moves a tiny tip in a raster pattern over the specimen. However, in this case the atomically sharp tip is mounted on a strip of metal foil. The overlap of the electron cloud at the tip with the electron clouds of the surface atoms generates a repulsion and the foil is deflected. This deflection is, in turn, measured by a STM, the tip is just above the foil. A feedback mechanism holds the tunneling current between the tip and the foil constant by adjusting the voltage on the Z-piezo control, which moves the sample up and down so that the deflection of the foil remains constant. The atomic force microscope (AFM) “reads” the surface by recording the contours of the repulsive force.

Laser Force Microscopes

The AFM could in principle image any nonconducting or conducting sample. However, it makes contact with enough force to distort or move many biological molecules and contaminate or damage the sample. Wickramasingle et al. developed the laser force microscope, a scanned probe microscope that can scan surfaces without touching it[4].
The LFM makes use of the attractive force that develops between a surface and a probe that is from two to twenty nanometers away. On semiconductors and insulators this force is largely the result from the surface tension of water that condenses between the tip and the sample. However, van der Waals interactions contribute also. The effect of these forces on a vibrating probe is detected by the LFM. The probe consists of a tiny tapered tungsten wire with a downturned tip that is etched to a diameter that is less than fifty nanometers. At the base of the wire a piezoelectric transducer converts an alternating current with a driving frequency a little above the lowest mechanical resonance of the wire into a vibration of the wire. When the vibrating tip is brought close to the sample, the wire experiences a weak attractive force. This force changes the resonance condition slightly and therefore the amplitude of the vibration changes. The changes are however,

quite large due the sensitivity of the resonance condition. These changes in amplitude are detected by a laser sensor using interferometry and a feedback mechanism responds to changes in the behavior of the tip by varying the Z piezo voltage. Thus the vibration amplitude is stabilized and hence the probe-to-surface distance is kept constant.

The AFM and LFM are just two members of a whole family of scanned probe microscopes that are based on similar technology. Considering the relative ease with which these technologies can be implemented, it is expected that these microscopies will have a tremendous impact in the immediate future[5, 6, 7].

Chapter 2 Scope and background of the project

At Portland State University a STM has been in use for several years. The complete setup was bought from “STM Laboratory, Institute of Chemistry, Chinese Academy of Sciences” and is now used by Margie Fyfield, a Ph.D. student, to take images with atomic resolution.

The program that controls the tip movement and stores and displays the data is written in "C," a popular programming language, but the program has some disadvantages:

- some functions that appear in the menus simply do not work
- it is based on the use of an obsolete data acquisition card
- there is no manual available
- the program is poorly documented
- the structure of the program is unclear and it has a poor readability
- the compiler gives about 100 warning messages after compiling, showing a lack of good programming style
- it's very difficult to amend because of it's unclear structure and poor readability

The University of Portland (UP) has the electronics for a STM, though it is broken just now and has to be repaired first. UP also has a box that contains the piezo drives and the tip and sample holder, and it has the necessary computer hardware and a modern data acquisition card, a DAS-1601 from Keithley Data Acquisition -- the leading brand -- including the drivers for 'C.'

I was looking for a project that should have some certain characteristics:

- it should be interesting
- it must be doable in one year
- it should challenge me
- I wanted to learn something new that could be useful later in my job

The project that was suggested to me -- writing a new program for a STM -- seemed to fulfill my expectations and could, furthermore, profit both PSU and UP.

The first consideration was which language to use. As I did not know any computer language I chose to learn what arguably is the most widespread language, and which is also said to be quite efficient: 'C.' After finishing the book C by example[8] and having a look at the hardly readable program in use at PSU, I decided to start a totally new program. My aim was not to write an as fast, compressed and efficient program as possible, including many nice and colored menus. I was rather concerned about readability and well-structured programming with a clear and simple flow so that the program could be easily amended by a future programmer.

Chapter 3 The computer program

Introduction

The complete computer program listing is shown in Appendix B. The flow chart for this program is presented in Appendix A. In the next section, the menu will be described. The following section is devoted to the critical parameters for the data acquisition board that was used in these experiments. The next section deals with the computer language used, followed by a section on the global parameters used in the program.

The Menu

The computer menu appears as follows:

```
X-voltage starting at (- 5.00 - +4.99):
Y-voltage starting at (- 5.00 - +4.99):
scanning Range:  0.40
averaging over N measurements:  1
print screen after M lines:  10
Delay-counter = 1
Gain-nr.:  0
Channels:  x-chn0  y-chn1

min:  606    max:  720

single scanning at 'constant Height' mode
single scanning at 'constant cUrrent' mode
continuous scAnning at 'constant height' mode
cOntinuous scanning at 'constant current' mode
Savedata
Loaddata
Printscreen
Exit

your choice:  __
```

This menu shows the current values for respectively 'xVoltStart', 'yVoltStart', 'xVoltScanRange', 'scanTimes', 'printLines', 'delayCtr', 'gain, 'xChannel' and 'yChannel' and also for 'zADvalueMin' and 'zADvalueMax'. To change one of these variables the user has to input the character (followed by 'return') that is capitalized in the corresponding line:

'x' to change 'xVoltStart'

'y' to change 'yVoltStart'

'r' to change 'xVoltScanRange' (and 'yVoltScanRange')

'n' to change 'scanTimes' (by changing 'powerofTwo')

'm' to change 'printLines'

'd' to change 'delayCtr'

'g' to change 'gain'

'c' to switch the output channels for the x-voltage and y-voltage

In the lower part of the menu the possible functions of the program are listed. Again to choose one of these functions the user has to input the character (followed by 'return') that is capitalized in the corresponding line:

'h' for a single scan in the 'constant height' mode (z-voltage is read on input channel #1)

'u' for a single scan in the 'constant current' mode (channel #0) 'a' for continuous scanning in the 'constant height' mode (channel#1)

'o' for continuous scanning in the 'constant current' mode (channel#0)

's' to save the collected data

'l' to load data from a data file

'p' to display the current data on the screen

'e' to exit program

Data acquisition card

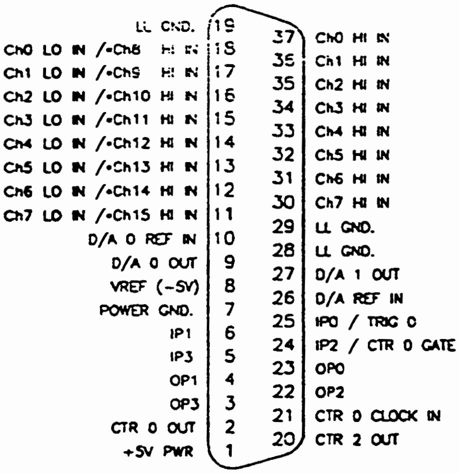
"The DAS-1601 and DAS-1602 boards (hereinafter referred to as the DAS-1600) are 12-bit analog and digital interface boards for the IBM PC/XT/AT and compatible computers. The DAS-1600 installs directly into a computer expansion slot, turning the computer into a fast, precise data acquisition and signal analysis instrument."[9]

"The DAS-1600 offers 8 differential or 16 single-ended analog inputs with 12-bit resolution at 100 ksamples/sec. The inputs can be set in unipolar (0 - 10 V) or bipolar(-10 - +10 V) modes. The input configuration is set by switches on the board. Input ranges are software-programmable. The DAS-1601 provides gains of 1, 10, 100 and 500 while the DAS-1602 offers gains of 1, 2, 4, and 8."[10]

"Two channels of 12-bit D/A converter are also provided. The outputs have switch-selectable ranges of 0 - 5, 0 - 10, -5 - +5, and -10 - +10 volts full scale. Regardless of range selected, all analog outputs are set to 0 V at power up."[11]

Analog and digital I/Os use a 37-pin, D-type connector that projects through the connector panel at the rear of the computer. The mating connector is a standard, 37-pin D-type female such as an ITT/Cannon #DC-37S for soldered connections."[12] See Fig. 1.

"Pins 11 through 18 perform a double function, depending on the setting of the Channel Configuration switch. In 8-channel differential configuration, these pins provide the low inputs of Channels 0 - 7 corresponding to the high inputs of these channels on pins 30 - 37. In 16-channel single-ended configuration, they provide additional channel-high inputs for Channels 8 - 15."[13]



Alternative connections used in 16--channel single--ended (SE) input configuration (set by 8/16 Switch).

Figure 1. Rear view of the I/O connector (after Ref. 12).

We used the DAS-1601 with following configuration:

Board type:	DAS-1601
Base Address:	&H300
Clock Select:	10MHz
Wait State:	No
A/D Mode:	Unipolar

A/D Config:	Single-ended
D/A 0 Mode:	Bipolar
D/A 1 Mode:	Bipolar
D/A 0 Ref:	-10.0
D/A 1 Ref:	-10.0
DMA Channel:	3
IRQ Channel:	7
Digital Cfg:	...
Number EXP16s:	0
EXP16 gains:	[N/A]
Number EXPGPs:	0
EXPGP Gains:	[N/A]
CJR Channel:	[N/A]
Number SSH4As:	0
SSH4A Gains:	[N/A]
SSH4A Mode:	[N/A]
SSH4A Timing:	[N/A]
A/D Gain:	[N/A]

The C-compiler used

The Microsoft QuickC 1.00 compiler was used and in addition, the library files 'DAS1600.LIB' and 'DASRFACE.LIB' provided by the ASO-1600 software package were included to compile the C program 'STM.C'. These files should be in one directory together with the Microsoft library file 'GRAPHICS.LIB'. To resolve the external functions and create the executable file 'STM.EXE' one has to type:

```
'QCL /C STM.C'

'LINK STM,,,DAS1600+DASRFACE+GRAPHICS'
```

Description of the program

The first lines include the standard header files "stdio.h", "graph.h", "math.h", "ctype.h", "string.h" and "dos.h" and the DAS-1600 Driver include file "userprot.h" provided by the ASO1600-package.

The #define-commands determine the size of the image and the minimal and maximal values for the output-voltages and thus the maximal scanning range. We have chosen an image consisting of 150x150 data points. To change this size you simply have to change the numbers behind "NUMBXSTEPS" and "NUMBYSTEPS" respectively. As a matter of convenience and readability '150' will be used instead of "NUMBXSTEPS" or "NUMBYSTEPS" in the following. The values for the maximal and minimal voltages are given by the data acquisition card used and its setup. These values are limited to values acceptable for the card.

Following is a list of the global variables, their type, their purpose and the functions that use and change them:

- DAS1600:

  type: DDH - user defined (in "userprot.h")

  fcts: gethandle()
  writexDAvalue()
  writeyDAvalue()
  readADvalue()

  purpose: used as argument of several external fcts (DAS1600_GetDevHandle(), K_DAWrite(), K_ADRead()), defined in "userprot.h"

- NumberOfBoards:

  type: character

  fcts: opendevice()

  purpose: used as argument in 'DAS1600_DevOpen()'

- ADvalue:

  type: long integer

  fcts: readADvalue()
  printmenu()
  scansample()

  purpose: the data acquisition card stores the result of an single AD-operation in this variable

- zADvalue[][]:

type: 150x150-field of integers

fcts: scansample()

savedata()

loaddata()

calczADvalueMin()

calczADvalueMax()

printscreen()

purpose: each element holds the value corresponding to one data point

- zADvalueMin:

type: integer

fcts: calczADvalueMin()

printscreen()

savedata()

loaddata()

purpose: holds minimal value of already collected data points, needed for standardization in 'printscreen()'

- zADvalueMax:

type: integer

fcts: calczADvalueMax()

printscreen()

savedata()

loaddata()

purpose: holds maximal value of already collected data points, needed for standardization in 'printscreen()'

- inf[]:

  type: character array

  fcts: savedata()

  loaddata()

  purpose: holds extension for data file

- dat[]:

  type: character array

  fcts: savedata()

  loaddata()

  purpose: holds extension for info file

- videoconfig vc:

  type: structure

  fcts: main()

  purpose: not used yet

- modes[]:

  type: integer array

fcts: main()

purpose: used as argument to change video mode

- *modenames[]:

  type: array of character pointers

  fcts: not used yet

  purpose: not used yet

- xVoltScanRange:

  type: floating point

  fcts: printmenu()

  calcxDAvalue()

  changeScanRange()

  checkValues()

  purpose: determines the scanning range in x-direction and thereby the magnification of the image on the screen

- yVoltScanRange:

  type: floating point

  fcts: printmenu()

  calcyDAvalue()

  changeScanRange()

  checkValues()

purpose: determines the scanning range in y-direction and thereby the magnification of the image on the screen

- xVoltStart:

  type: floating point

  fcts: printmenu()

  calcxDAvalue()

  changexVoltStart()

  checkValues()

  purpose: determines x-component of starting point of image

- yVoltStart:

  type: floating point

  fcts: printmenu()

  calcyDAvalue()

  changeyVoltStart()

  checkValues()

  purpose: determines y-component of starting point of image

- xChannel:

  type: integer

  fcts: printmenu()

  writexDAvalue()

switchChannels()

purpose: determines on which output channel x-voltage is output, possible values: 0 or 1

- yChannel:

type: integer

fcts: printmenu()

writeyDAvalue()

switchChannels()

purpose: determines on which output channel y-voltage is output, possible values : 0 or 1

- powerofTwo:

type: integer

fcts: scansample()

changeNumbofMeas()

purpose: 2 to the power of 'powerofTwo' measurements are made for each data point, possible values: 0, 1, 2, 3, ...

- scanTimes:

type: integer

fcts: printmenu()

scansample()

changeNumbofMeas()

purpose: 'scanTimes' measurements are made for each data point, possible values: 1, 2, 4, 8,... ; cannot be changed directly, just by changing 'powerofTwo'

- printLines:

type: integer

fcts: printmenu()

scansample()

changeLines()

checkValues()

purpose: after 'printLines' lines of data points are taken while scanning the image of the already collected data points is printed to the screen before scanning is continued

- gain:

type: integer

fcts: printmenu()

readADvalue()

changeGain()

checkValues()

purpose: for the DAS-1601: '0' corresponds to a gain of 1, '1' to 10, '2' to 100 and '3' to 500 for the DAS-1602: '0' corresponds to a gain of 1, '1' to 2, '2' to 4 and '3' to 8

- delayCtr:

    type: long integer

    fcts: printmenu()
    delay()
    changeDelayCtr()
    checkValues()

    purpose: after moving the tip to the desired place a delay is made in order to give the tip time to settle down before measuring

Following is a list of the functions, the global and local variables they use, receive and pass, the functions that each calls and is called from, and the task of each function.

- main():

    local variables: character choice

    global variables: vc

    calls: opendevice()
    gethandle()
    changeDelayCtr()
    changexVoltStart()
    changeyVoltStart()

changeGain()

changeScanRange()

changeNumbofMeas()

changeLines()

switchChannels()

savedata()

loaddata()

printscreen(), '149' is passed for 'yCount', e.g. the whole image is displayed

exitprg()

checkValues()

scansample(), the first passed parameter (0 or 1) determines the channel from which the z-voltage is read; the second (0 or 1) yields either a single scan (1) or continuous scanning (0)

description: at the beginning the hard- and software is initialized and communication to the board established by calling the external functions 'opendevice()' and 'gethandle()'; after a keyboard hit the point 'Menu' marks the beginning of a goto-loop; the screen is cleared and the menu printed ('printmenu()'); the program waits for the user to input a single character followed by 'return' and then executes a command block depending on the input due to a switch command; after executing the command block the goto statement makes the program begin at the point 'Menu' again; the command blocks of most cases are self explanatory.

case 'p': the screen is cleared; video mode "VRES16COLOR" chosen; the video configuration stored in the variable 'vc'; the logical origin set to 270 pixels to the right and 30 pixels down from the upper left corner of the monitor and finally the full image printed on the screen; after a keyboard hit the default video mode is restored and the program goes to the beginning of the goto-loop.

case 'h': single scanning at continuous height:

the values for the global variables are checked; if they are OK the function call returns a '1' and the program continues; otherwise, a '0' is returned; the program waits for a keyboard hit and then goes back to the menu; in the first case the function 'scansample()' is called and passed the value '1' for the variable 'channel' (e.g. channel #1 is used for reading the z-voltage) and '1' for the variable 'kbFlag' (e.g. a single scan).

case 'a': same as above, but this time a '0' is passed for 'kbFlag' (e.g. continuous scanning)

case 'v': similar to above, '0' for 'channel' (e.g. channel #0 is used for reading the z-voltage) and '1' for 'kbFlag'

case 'o': similar to above, '0' for 'channel' and '0' for 'kbFlag'

- opendevice()

  local variables: integer Err

  global variables: Number of Boards

  called by: main()

  calls: DAS1600_DevOpen()

  description: the hard- and software is initialized by calling the external function 'DAS1600_DevOpen()'; the first parameter must be the address of the configuration file starting at the directory from which the program was started; the second parameter is '&NumberofBoards'; in case of failure a bell is rung and the program exits; otherwise, a confirmation is sent to the screen.

- gethandle()

  local variables: integer Err

  global variables: DAS1600

  called by: main()

  calls: DAS1600_GetDevHandle()

description: communication is established by calling the external function 'DAS1600_GetDevHandle(); the first parameter is the number of the board (always '0' as just one board was used) and the second parameter is '&DAS1600'; in case of failure a bell is rung and the program exits; otherwise, a confirmation is sent to the screen.

- printmenu()

global variables: xVoltStart

yVoltStart

xVoltScanRange

scanTimes

printLines

delayCtr

gain

xChannel

yChannel

zADvalueMin

zADvalueMax

called by: main()

description: the first part prints the current values - which can be changed independently - for the scanning variables on each line; the capital letter in each line indicates which character is to be input in order to change a value;

the second part prints the other options on separate lines with one capital letter again indicating which character to input for each option.

- scansample()

  local variables: integer channel

  purpose: is passed from 'main()' to 'readADvalue()'; determines from which input channel z-voltage is read; possible values: 0 or 1

  integer kbFlag

  purpose: is passed from 'main()'; determines if scanning is repeated at end of loop (value '0') or if control returns to 'main()' (value '1')

  long integer xDAvalue

  purpose: is passed to 'writexDAvalue()' as argument; range: 0 - 4095; '0' corresponds to -5.00V, '4095' to +4.99V on output channel #'xChannel'; calculated and returned as return value from 'calcxDAvalue()'

  long integer yDAvalue

purpose: is passed to 'writeyDAvalue()' as argument; range: 0 - 4095; '0' corresponds to -5.00V, '4095' to +4.99V on output channel #'yChannel'; calculated and returned as return value from 'calcyDAvalue()'

integer xCount

purpose: counter for inner loop that controls x-voltage

integer yCount

purpose: counter for outer loop that controls y-voltage

long integer zTotal

purpose: holds sum of results of all measurements for one data point

integer avgCount

purpose: counter for loop of several measurements for one data point

character keyhit

purpose: stores keyboard hit of user while scanning

global variables: scanTimes

ADvalue

zADvalue[][]

powerofTwo

printLines

called by: main()

calls: calcyDAvalue()

writeyDAvalue()

calcxDAvalue()

writexDAvalue()

delay()

readADvalue()

printscreen()

calczADvalueMin()

calczADvalueMax()

description: 'Scan' marks the beginning of a goto-loop;

in the outer loop, which controls the y-voltage, 'yCount' counts from 0 to 149; 'yCount' is passed to 'calcyDAvalue()' and 'yDAvalue' holds the return value; possible values for 'yCount': 0 - 4095, depending on 'yVoltStart' and 'yVoltScanRange'; the y-Voltage corresponding to 'yDAvalue' is output on channel #'yChannel';

the inner loop, which controls the x-voltage, works analog; 'delay()' is called in order to give the tip time to move to the desired place;

every time the innermost loop is run the z-Voltage is measured, the result (possible: 0 - 4095) stored in the upper 12 bits of 'ADvalue' and 'ADvalue' summed to 'zTotal'; in order to obtain values from 0 to 4095 the upper 12 bits of 'ADvalue' must be shifted 4 bits to the right; 'zTotal' holds the sum of all the single measurements for one data point; in order to obtain values from 0 - 4095 'zTotal' must be shifted by 'powerofTwo' bits to the right and the result is stored in zADvalue[xCount][yCount];

the program checks after scanning one line if a multiple of 'printLines' lines has already been read and prints the partial image if so;

if the key 'q' has been hit during the scan the program returns immediately to 'main()';

if a key besides 'q' has been hit 'kbFlag' is set to '1' (i.e. the program returns to 'main()' after the last line is read and the whole image displayed on the screen);

the minimal and maximal value of zADvalue[][] are calculated and the whole image is printed on the screen;

if kbFlag equals '0' the program goes to 'Scan' and repeats the goto-loop; otherwise, it returns to 'main()'

- writexDAvalue()

  local variable: integer Err

  long integer xDAvalue (passed by value from 'scansample()')

purpose: determines the voltage that is output on channel #'xChannel'; possible values: 0 - 4095, '0' corresponds to -5.00V and '4095' to +4.99V

global variables: DAS1600

xChannel

called by: scansample()

calls: K_DAWrite()

description: the long integer 'xDAvalue' contains 12 bits of information (possible values: 0 - 4095); 'K_DAWrite()' expects these 12 bits to be in the upper 12 bits of the used long integer, therefore the bits of 'xDAvalue' must be shifted 4 bits to the left;

by calling 'K_DAWrite()' the voltage corresponding to 'xDAvalue' is output on channel #'xChannel'.

- writeyDAvalue()

local variable: integer Err

long integer yDAvalue (passed by value from 'scansample()')

purpose: determines the voltage that is output on channel #'yChannel'; possible values: 0 - 4095, '0' corresponds to -5.00V and '4095' to +4.99V

global variables: DAS1600

xChannel

called by: scansample()

calls: K_DAWrite()

description: the long integer 'yDAvalue' contains 12 bits of information (possible values: 0 - 4095); 'K_DAWrite()' expects these 12 bits to be in the upper 12 bits of the used long integer, therefore the bits of 'yDAvalue' must be shifted 4 bits to the left;

by calling 'K_DAWrite()' the voltage corresponding to 'yDAvalue' is output on channel #'yChannel'.

- readDAvalue()

local variable: integer Err

integer channel (passed from 'scansample()")

global variables: DAS1600

called by: scansample()

calls: K_ADRead()

description: by calling 'K_ADWrite()' the z-voltage is read from channel #'channel' at the gain corresponding to 'gain' and the result (possible: 0 - 4095) is stored in the upper 12 bits of the long integer 'ADvalue'.

- calczADvalueMin()

local variables: integer yCount (passed from calling function) purpose: specifies up to which line of the data field the minimum is looked for

integer xCtr

purpose: counter for inner loop that controls x-position of data point; counts from '1' to '149'

integer yCtr

purpose: counter for outer loop that controls y-position of data point; counts from '0' to 'yCount'

global variables: zADvalueMin

zADvalue[][]

called by: scansample()

printscreen()

description: initially the first data point ('zADvalue[0][0]') is assigned to 'zADvalueMin'; the following sorting routine looks for the smallest value for the data points from the first line to the 'yCount'-line

- calczADvalueMax()

local variables: integer yCount (passed from calling function)

purpose: specifies up to which line of the data field the maximum is looked for

integer xCtr

purpose: counter for inner loop that controls x-position of data point; goes from '0' to '149'

integer yCtr

purpose: counter for outer loop that controls y-position of data point; goes from '0' to 'yCount'

global variables: zADvalueMax

zADvalue[][]

called by: scansample()

printscreen()

description: initially the first data point ('zADvalue[0][0]') is assigned to 'zADvalueMax'; the following sorting routine looks for the biggest value for the data points from the first line to the 'yCount'-line

- calcxDAvalue

local variables: integer xCount (passed from calling function)

integer xDAvalueInt

long integer xDAvalue

global variables: xVoltStart

xVoltScanRange

called by: scansample()

description: depending on 'xVoltStart', 'xVoltScanRange' and 'xCount' a value is assigned to the integer 'xDAvalueInt'. '0' corresponds to -5.00V, '4095' to +4.99V. 'xDAvalueInt' is assigned to the long integer 'xDAvalue', which is then returned to the calling function.

- calcyDAvalue

  local variables: integer yCount (passed from calling function)

  integer yDAvalueInt

  long integer yDAvalue

  global variables: yVoltStart

  yVoltScanRange

  called by: scansample()

  description: depending on 'yVoltStart', 'yVoltScanRange' and 'yCount' a value is assigned to the integer 'yDAvalueInt'; '0' corresponds to -5.00V, '4095' to +4.99V; 'yADvalueInt' is assigned to the long integer 'yDAvalue', which is then returned to the calling function.

- printscreen()

  local variables: integer yCount (passed from calling function)

  purpose: the first 'yCount' lines are printed on the screen

  integer xscreenCtr

  purpose: counter for inner loop (0 - 149); determines x-position of data point

  integer yscreenCtr

purpose: counter for outer loop (0 - 'yCount'); determines y-position of data point

integer xscreenCtr2

integer yscreenCtr2

integer pixelcolor

purpose: a number between 0 and 15, which corresponds to a certain color, is assigned to each data point

floating point norm

purpose: standardization factor

floating point diff

floating point floatMax

purpose: assigns a floating point variable to the Maximum

floating point floatMin

global variables: zADvalueMin

purpose: assigns a floating point variable to the Minimum

zADvalueMax

zADvalue[]

called by: main()

scansample()

calls: calczADvalueMin()

calczADvalueMax()

description: a floating point normalization factor 'norm' is calculated so that 'pixelcolor' is assigned values from 0 to 15; if 'zADvalueMin' equals 'zADvalueMax' the control returns to the calling function without printing the image on the screen;

each data point corresponds to 2x2 pixels on the screen.

- savedata

local variables: character string fileName[]

purpose: contains name of data file without extension ".dat"

character string fileName2[]

purpose: contains name of info file without extension ".inf"

character string pointer *dataFileName

purpose: contains name of data file with extension ".dat"

character string pointer *infoFileName

purpose: contains name of info file with extension ".inf"

FILE *dataFptr

purpose: controls writing to data file on disk

FILE *infoFptr

purpose: controls writing to info file on disk

character answer

integer xCount

integer yCount

global variables: character dat[]

character inf[]

zADvalue[]

zADvalueMin

zADvalueMax

called by: main()

description: the user is asked to input the filename without extension; after testing the exit condition the extension ".dat" is added and a safety check done to see whether the file already exists;

the data file is opened and the data points 'zADvalue[]' are written to the data file from the left to the right, line after line with a space after each data point; finally 'zADvalueMin' and 'zADvalueMax' are stored and the data file closed;

same procedure for the info file; the body of the info file is empty so far, i.e. no information is written to the info file yet.

- loaddata

local variables: character string fileName[]

purpose: contains name of data file without extension ".dat"

character string fileName2[]

purpose: contains name of info file without extension ".inf"

character string pointer *dataFileName

purpose: contains name of data file with extension ".dat"

character string pointer *infoFileName

purpose: contains name of info file with extension ".inf"

FILE *dataFptr

purpose: controls reading from data file on disk

FILE *infoFptr

purpose: controls reading from info file on disk

character answer

integer xCount

integer yCount

global variables: character dat[]

character inf[]

zADvalue[]

zADvalueMin

zADvalueMax

called by: main()

description: the user is asked to input the filename without extension; after testing the exit condition the extension ".dat" is added, the data file is opened and the data points 'zADvalue[]' are read in from the data file from the left to the right, line after line with a space after each data point; finally 'zADvalueMin' and 'zADvalueMax' are read in and the data file closed;

same procedure for the info file; the body of the info file is empty so far, i.e. no information can be read from the info file yet.

- exitprg()

  local variables: character answer

  called by: main()

- delay()

  local variables: long integer i

  long integer k

  global variables: delayCtr

  called by: scansample()

- switchChannels()

  global variables: xChannel

  yChannel

  called by: main()

  description: by this routine the output channels for the x- and y-voltage are switched; the resulting image is rotated by 90 degrees.

- changeNumbofMeas()

  global variables: powerofTwo

  scanTimes

  called by: main()

- changeLines

  global variables: printLines

  called by: main()

- changeScanRange()

  global variables: xVoltScanRange

  yVoltScanRange

  called by: main()

- changeGain()

  global variables: gain

  called by: main()

- changexVoltStart

  global variables: xVoltStart

  called by: main()

- changeyVoltStart

  global variables: yVoltStart

  called by: main()

- changeDelayCtr()

  global variables: delayCtr

  called by: main()

- checkValues()

global variables: xVoltStart

yVoltStart

xVoltScanRange

yVoltScanRange

gain

delayCtr

printLines

called by: main()

description: some of the changeable global variables are checked to see whether they contain allowed values; if the check is OK the function returns '0'; otherwise, a '1'.

Chapter 4 Results

The program was tested at the STM of PSU to make images of HOPG (highly oriented pyrolitic graphite), a substance whose atoms are ordered in hexagonal arrays (see Fig. 2), as follows.

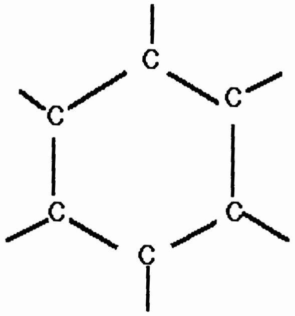


Figure 2. Schematic drawing of the configuration that gives rise to the hexagonal pattern in HOPG.

A thin slice of HOPG was glued on the sample holder by a silver glue. To get an atomically flat surface, we made use of the fact that the bindings of the carbon atoms are very strong within one layer. However, the bindings between layers are comparatively weak. If you rub a piece of scotch tape on the surface of the sample and remove the tape, several layers will stick to it and you obtain a smooth layer at the surface of the sample, that is, an atomically flat surface. The tip was made of tungsten wire that was clipped by scissors.

First, the old program with the Chinese data acquisition board was used to see whether the tip was of sufficient quality to produce images with atomic resolution. After this, the new program with the DAS-1601 from "Keithley Data Acquisition" controlled the operation of the STM. In this way, we made sure that causes of fault other than the program were excluded in case of failure.

The program proved successful. We obtained images showing the hexagonal structure of HOPG with atomic resolution. In the following, we will show two examples of the acquired data.

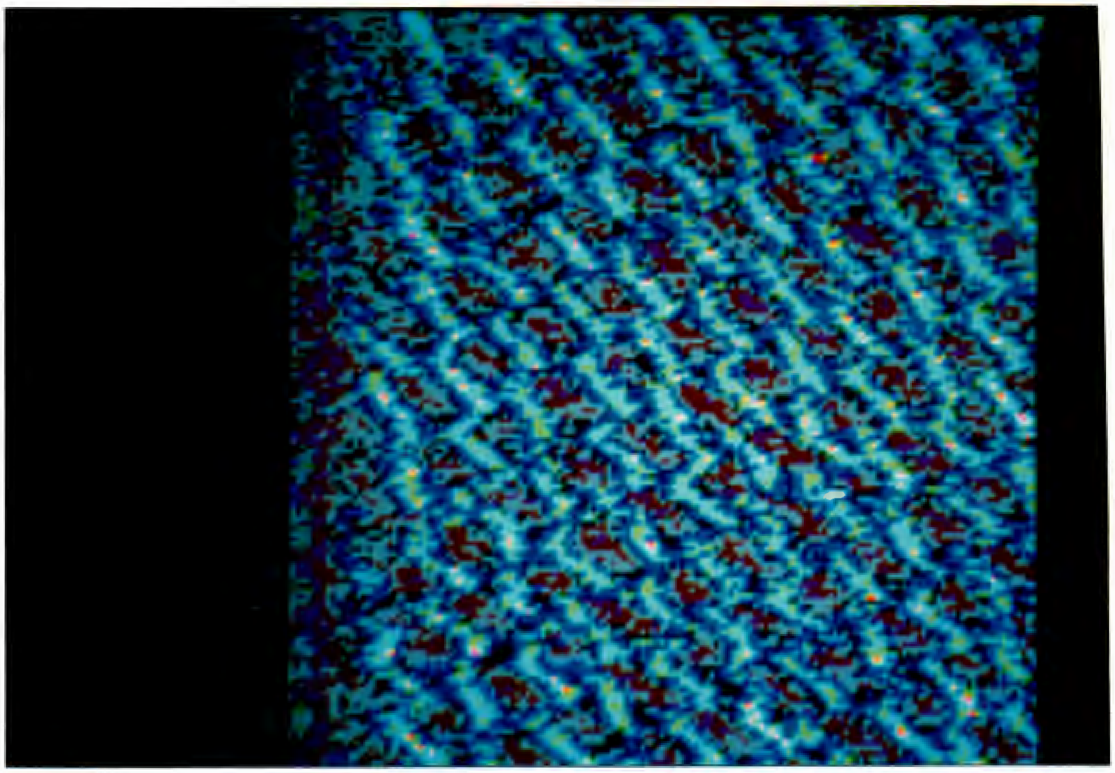


Figure 3. Image 'chris.dat'.

Data in Fig. 3 was obtained under the following scan conditions.

x-voltage starting at:	0.00V	N measurements:	2
y-voltage starting at:	0.00V	printed after M lines:	150
x-chn #1,y-.din #0		delay-counter:	0

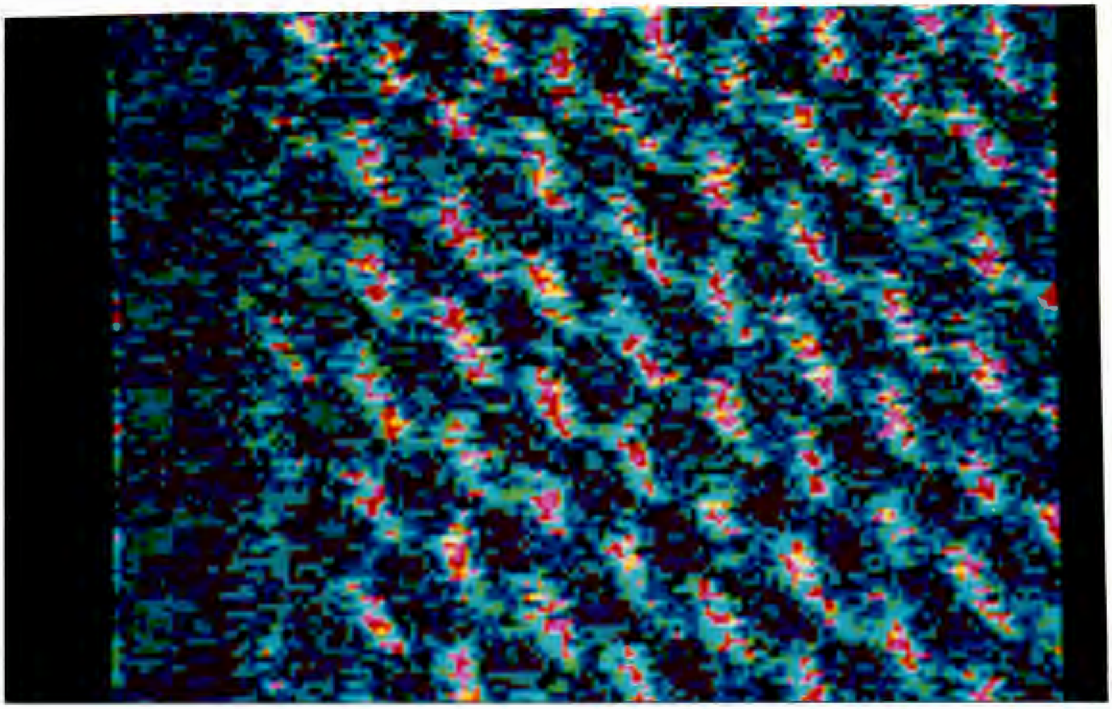


Figure 4. Image 'Karin.dat'

Data in Fig. 4 was obtained under the following scan conditions.

x-voltage starting at:	0.00V	N measurements:	2
y-voltage starting at:	0.00V	printed after M lines:	150
x-chn #1, y-chn #0		delay-counter:	0

The images are of medium quality. Considering the comparatively simple methods used of sample and tip preparation, and the fact that the data are not smoothed, this is a very good result. The distortion of the images at the left edge is due to the tip movement. Within one single line scan, the tip is moved by single steps in the x-direction, from the left to the right. As the tip is not totally rigid, it is bent slightly to the left due to its inertia. See Fig. 5.

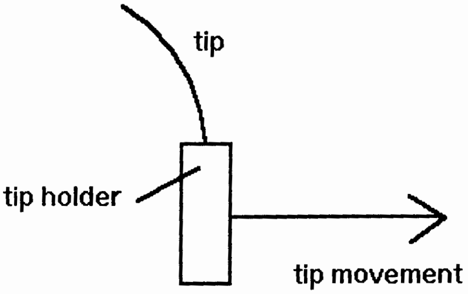


Figure 5. Tip motion during line scan.

At the end of the line the tip is moved by one step in y-direction and from the end of the line to the beginning of the next line, i.e., from the right to the left. Due to its inertia, this time the tip is slightly bent to the right. See Fig. 6.

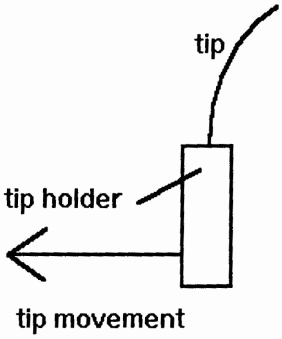


Figure 6. Tip motion during and just after the movement back to start the next line scan.

After some data points in this line, the tip has obtained its 'normal' position again, that is, being slightly bent to the left, and the distortion disappears.

Unlike the Chinese program, this program could measure several times for one data point before moving the tip to the reset data point. By averaging over more measurements, we hoped to improve the quality of the image. However, the best results were obtained for just one or two measurements per data point. If one sets N, equal or greater 8, the images were clearly compromized. This is probably due to thermal variations that have a greater influence the longer the scan takes.

For the same reason, it seemed advisable to choose small values for 'scanTimes', 'delayCtr', and 'x-Volt Scan Range' and to set 'printLines' to '150'.

For an image taken with the same values as above but with switched channels for the x-voltage and y-voltage you expect the same image rotated by 90 degrees. Actually, the images looked quite different; and the better scans were taken with the x-voltage output on channel #1 and the y-voltage on channel #0. This is similar to what is observed with the original software and thus seems to be a function of the STM proper.

It turned out that the starting values of the x-voltage and y-voltage did not have any significant influence on the images. For the same combination of the other variables but different 'xVoltStart' and 'yVoltStart', the images looked quite the same. This is not very surprising as the whole sample consisted of HOPG and therefore should show everywhere the same structure.

Chapter 5 Conclusions

The program has proven to work satisfactorily. It is well structured and easily readable and though it is quite short it can control the STM and the program produces images with atomic resolution. Even with our quick and simple tip and sample preparation, medium quality images are obtained without smoothing the data.

Although the program is not extremely complex and should not be considered the ultimate program for a STM, it should prove to be a strong building block for future improvements.

I want to list some possibilities for future improvements:

- The info file contains no information about the image and the way it is produced, yet. It is up to the user to be clear about which information should be in this file and to insert the corresponding commands in the program in the functions 'savedata()' and 'loaddata()'.

  Some suggestions are:

  - value of the changeable variables like 'xVoltStart', 'xVoltScanRange', 'gain' and so on
  - date and time image was taken
  - comment on which material was examined
  - name of user

- 'constant height' or 'constant current' mode

- The program seemed to produce better images the smaller the values for 'scanTimes', 'delayCtr', 'xVoltScanRange' and the bigger the values for 'printLines' were, i.e., the less time was taken to produce the image. This is probably due to thermal variations that become more important the longer the scan takes. As it generally takes the most time to display the image on the screen , it is recommended to assign the value '150' to 'printLines', i.e., just the image of all 150x150 data points is displayed after a single scan and no time is taken to display a partial image while scanning. Then the check after each line of scanning to see whether the image is to be displayed, could be omitted. Instead of calling the functions 'calcxDAvalue()' and 'calcyDAvalue()' to calculate 'xDAvalue' and 'yDAvalue' this task could be done in 'scansample()' directly, which might also improve the speed of the program slightly. Of course, the frequency of operation of the CPU is the main factor in determining the overall speed of the program.
- The image is displayed in the video mode: "VRES16COLOR", that is, 16 different colors are used to illustrate the differences in 'tunneling current' or 'z-voltage' for the signal data points. Looking at the images it might have been better to choose a video mode with gray scales, so darker spots would correspond to holes in the sample and brighter spots to bumps. Apparently, it is not as easy for the eye to observe structures that are represented by different colors as it is when the structures are represented by a gray scale.

- It could be useful to see two images on the screen at the same time. For example, one image could be an already existing image while the other one could display the latest collected data.
- The first few data points on the left side of each line are distorted due to the resetting of the tip from the last data point of a line to the first data point of the next line. Thus, the number of rows could be enlarged by, say ten, and the first ten data points of each line could be disregarded.
- At the end of a line the y-voltage is increased by one single step, while the x-voltage is reset from its maximal value ('xVoltStart' + 'xVoltScanRange') to its minimal value ('xVoltStart). One could insert a delay directly before the first data point of a line is measured to give the tip time to move to the corresponding place.

Other possible ways to improve the program will occur when the program is in use. Some improvements will depend on the given setup. The program was written such that it is as readable as possible and it should be no problem for an experienced programmer to amend this program and adjust it to the setup that will be used.

REFERENCES

1 Quate, F, Calvin: Vacuum tunneling: A new technique for microscopy, Physics Today, August 1986, pp. 26-33, 1986

2 Golovchenko, J., A.: The Tunneling Micrscope: A New Look at the Atomic World, Science, Vol. 232, pp. 48-53, 1986

3 Hansma, K., Paul; Tersoff, Jerry: Scanning tunneling microscopy, J. Appl. Phys., Vol.61, No.2, pp. R1-R23, 1987

4 Wickramasinghe, H., Kumar: Scanned-Probe Microscopes, Scientific American, October 1989, pp. 98-105, 1989

5 Demuth, J., E.; Koehler, U.; Hamers, R., J.: The STM learning curve and where it may take us, Journal of Microscopy, Vol. 152, Pt 2, November 1988, pp. 299-316, 1988

6 Feenstra, M., Randall: Scanning tunneling spectroscopy, Surface Science 299/300, pp. 965-979, 1994

7 Rohrer, H.: Scanning tunneling microscopy: a surface science tool and beyond, Surface Science 299/300, pp. 956-964, 1994

8 Perry, Greg: C by example, Que Corporation, Carmel, 1993

9 User Guide for the DAS-1600 Data Acquisition Board, Revision A, pp. 1-1, 1992

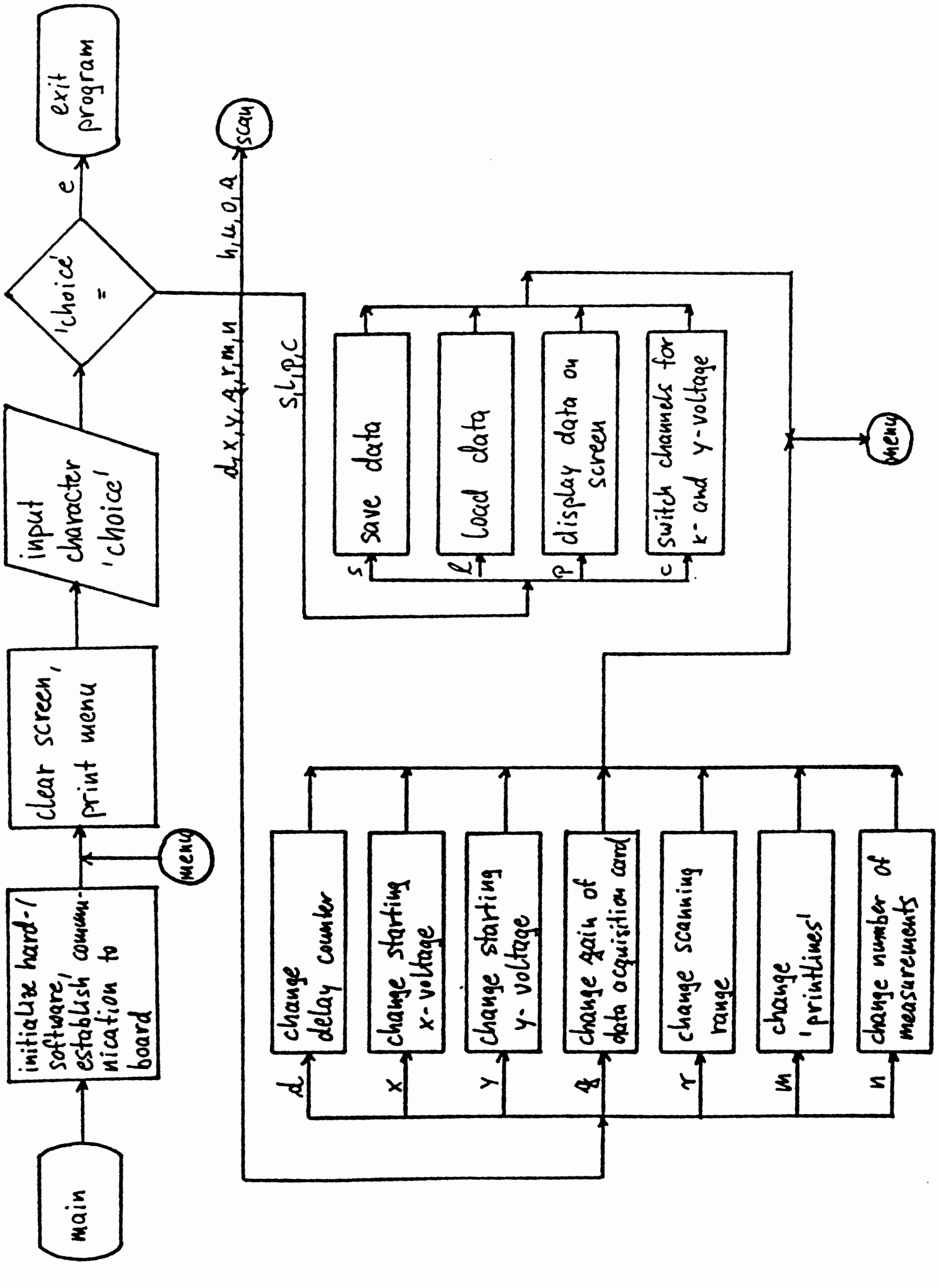
10 User Guide for the DAS-1600 Data Acquisition Board, Revision A, pp. 1-1, 1992

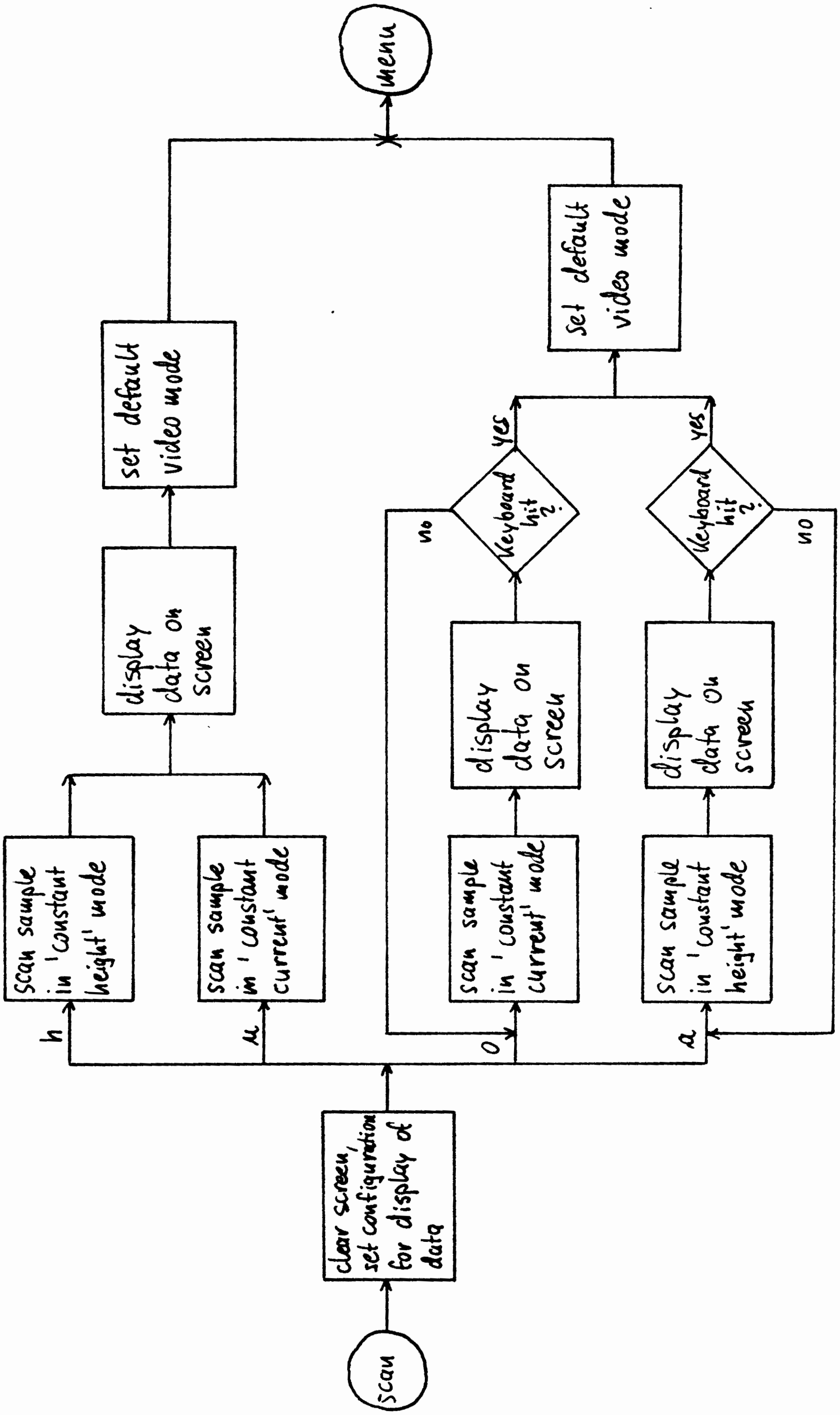
11 User Guide for the DAS-1600 Data Acquisition Board, Revision A, pp. 1-2, 1992

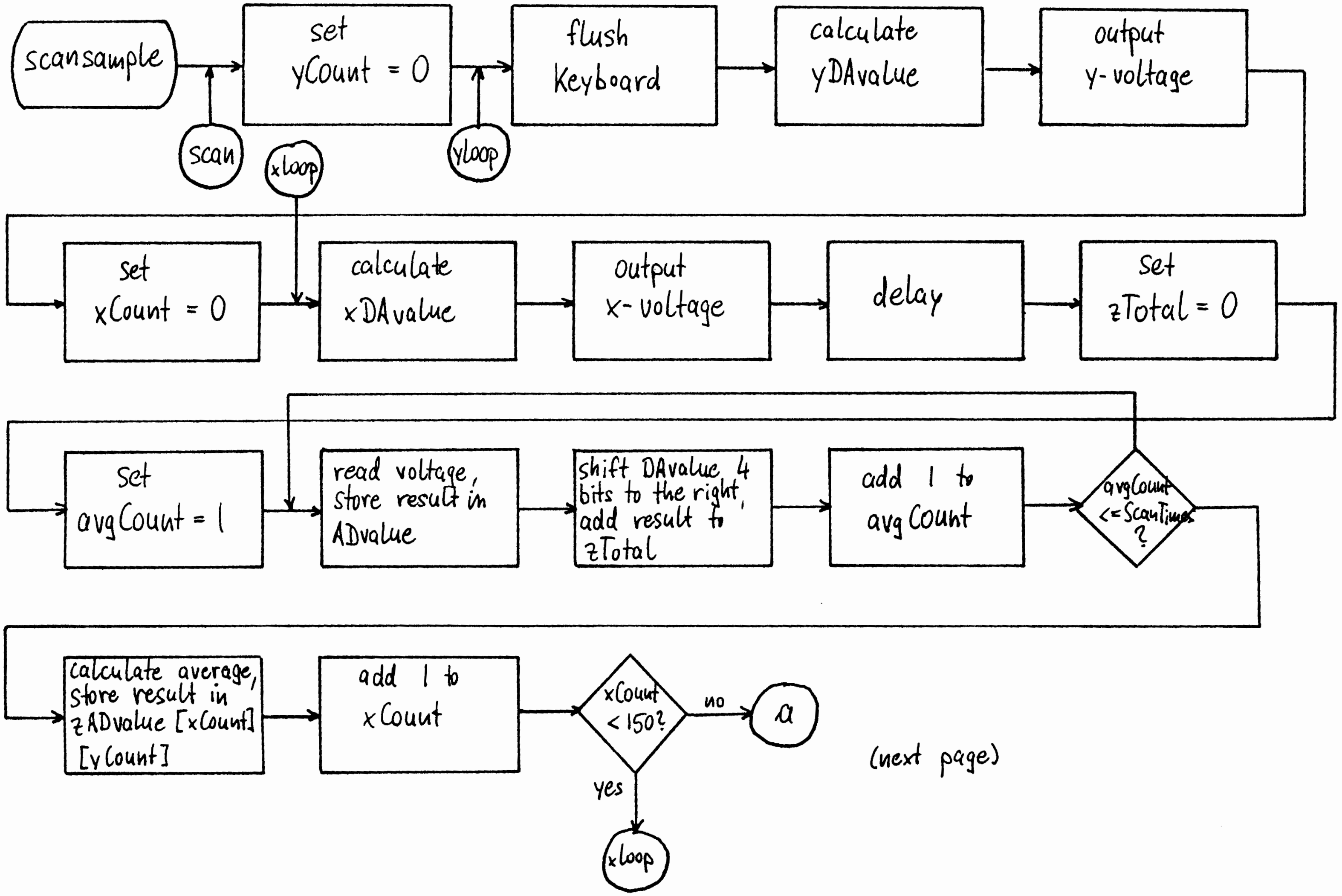
12 User Guide for the DAS-1600 Data Acquisition Board, Revision A, pp. 2-3, 1992

13 User Guide for the DAS-1600 Data Acquisition Board, Revision A, pp. 2-3, 1992

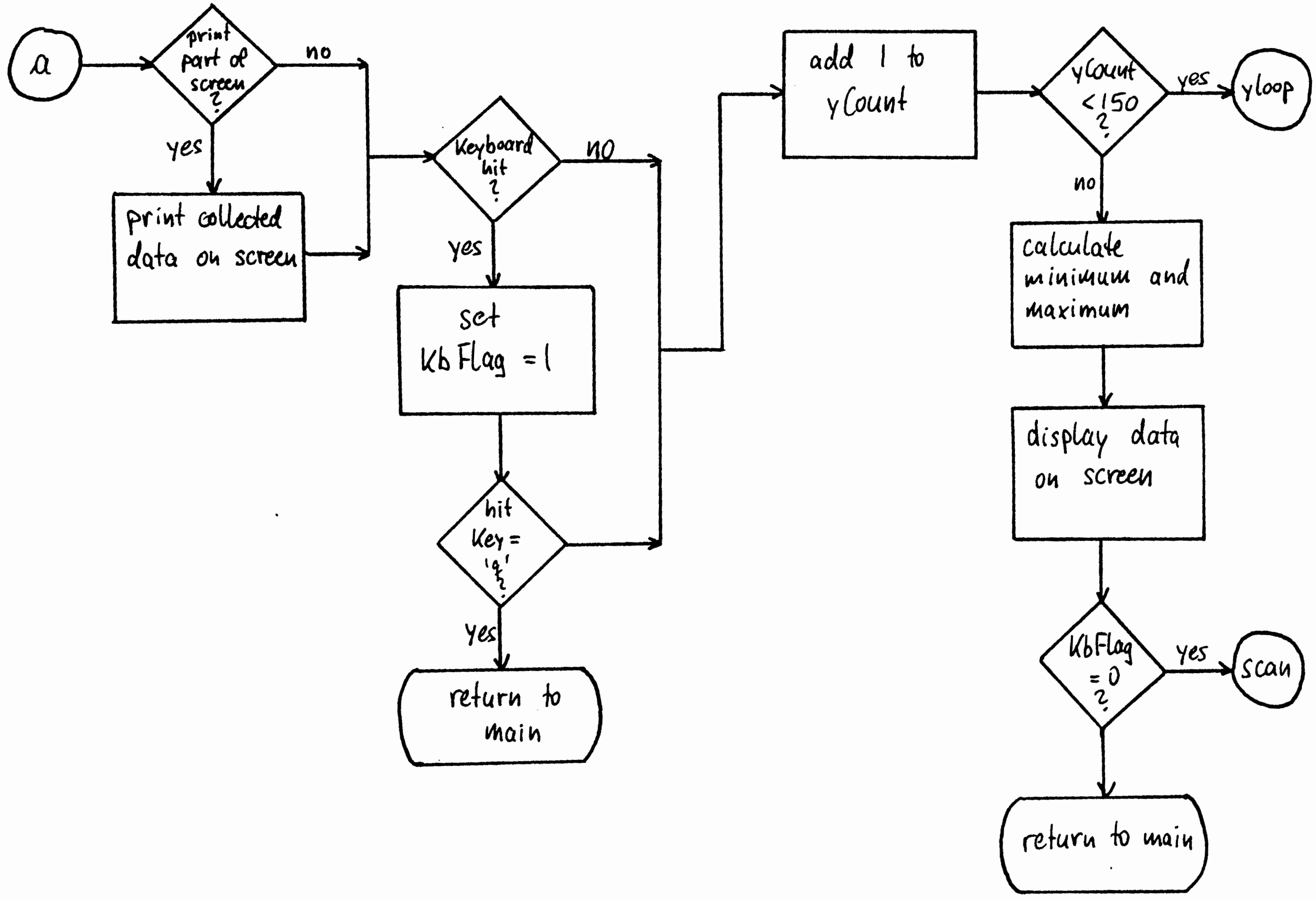
Appendix A Flow chart




scan
clear screen, set configuration for display of data
h
u
o
a
scan sample in 'constant height' mode
scan sample in 'constant current' mode
scan sample in 'constant current' mode
scan sample in 'constant height' mode
display data on screen
display data on screen
display data on screen
Keyboard hit ?
Keyboard hit ?
no
no
yes
yes
set default video mode
set default video mode
menu


scansample
scan
set yCount = 0
yloop
flush Keyboard
calculate yDAvalue
output y-voltage
xloop
set xCount = 0
calculate xDAvalue
output x-voltage
delay
set zTotal = 0
set avgCount = 1
read voltage, store result in ADvalue
shift DAvalue 4 bits to the right, add result to zTotal
add 1 to avgCount
avgCount <=ScanTimes ?
calculate average, store result in zADvalue [xCount] [yCount]
add 1 to xCount
xCount < 150?
no
a
yes
xloop

(next page)


a
print part of screen ?
no
yes
print collected data on screen
Keyboard hit ?
no
yes
set Kb Flag = 1
hit Key = 'q' ?
yes
return to main
add 1 to yCount
yCount <150 ?
yes
yloop
no
calculate minimum and maximum
display data on screen
KbFlag = 0 ?
yes
scan
return to main

Appendix B Listing of the computer program

```
/* MICROSOFT INCLUDE FILES */
#include "c:\qc1\include\stdio.h"
#include "c:\qc1\include\stdlib.h"
#include "c:\qc1\include\graph.h"
#include "c:\qc1\include\math.h"
#include "c:\qc1\include\ctype.h"
#include "c:\qc1\include\string.h"
#include "c:\qc1\include\dos.h"

/* DAS-1600/1400 DRIVER INCLUDE FILE */
#include "c:\qc1\stm\userprot.h"

/* DEFINE SIZE OF IMAGE */
#define NUMBXSTEPS 150     /* Number of rows      */
#define NUMBYSTEPS 150     /* Number of columns      */

/* DEFINE MAXIMAL SCANRANGE */
#define XVOLTMIN -5.0                    /* DETERMINED BY USED       */
#define YVOLTMIN -5.0                    /* DATAACQUISITION CARD */
#define XVOLTMAX 5.0                     /* AND ITS SETUP            */
#define YVOLTMAX 5.0                     /*                          */
#define XVOLTTOTALRANGE 10.0             /* XVOLTMAX - XVOLTMIN  */
#define YVOLTTOTALRANGE 10.0             /* YVOLTMAX - YVOLTMIN  */

/* GLOBAL VARIABLES */
DDH  DAS1600;                  /* DEVICE HANDLE                         */
char   NumberOfBoards ;        /* NUMBER OF BOARDS IN DAS1600.CFG */
long   ADvalue;                /* STRORAGE FOR ZA/D VALUE            */
int    zADvalue[NUMBXSTEPS][NUMBYSTEPS];
int    zADvalueMin;
int    zADvalueMax;
char   inf[]=".inf";
char   dat[] = ".dat";
struct videoconfig vc;
int    modes[16] = { _TEXTBW40, _TEXTC40, _TEXTBW80, _TEXTC80,
                     _MRES4COLOR, _MRESNOCOLOR, _HRESBW,
                     _TEXTMONO, _MRES16COLOR, _HRES16COLOR,
                     _ERESNOCOLOR, _ERESCOLOR, _VRES2COLOR,
                     _VRES16COLOR, _MRES256COLOR, _DEFAULTMODE};
char   *modenames[16] = {"TEXTBW40", "TEXTC40", "TEXTBW80", "TEXTC80",
                         "MRES4COLOR", "MRESNOCOLOR", "HRESBW",
                              "TEXTMONO", "MRES16COLOR", "HRES16COLOR",
                              "ERESNOCOLOR", "ERESCOLOR", "VRES2COLOR",
                              "VRES16COLOR", "MRES256COLOR",
                              "DEFAULTMODE"};

/* DEFAULT VALUES FOR VARIABLES */
float  xVoltScanRange =4.0;
float  yVoltScanRange =4.0;
float  xVoltStart     =0.0;
float  yVoltStart     =0.0;
int    xChannel       =  0;        /* X-PIEZO DRIVE CONTROLLED BY */
                                   /* CHANNEL #0 OF DAS-1600      */
int    yChannel       =  1;        /* Y-PIEZO DRIVE CONTROLLED BY */
                                   /* CHANNEL #1 OF DAS-1600      */
int    powerofTwo     =  0;        /* ONE MEASUREMENT             */
int    scanTimes      =  1;        /*            "                */
int    printLines     = 10;        /* PRINT SCREEN AFTER 10 LINES */
int    gain           =  1;        /* FOR DAS-1601 GAIN OF 10     */
long   delayCtr       =  1;

/* FUNCTION PROTOTYPING */
void   opendevice();
void   gethandle();
void   printmenu();
void   scansample(int channel, int kbFlag);
void   writexDAvalue();
void   writeyDAvalue();
void   readADvalue(int channel);
void   calczADvalueMin(int yCount);
void   calczADvalueMax(int yCount);
void   printscreen(int yCount);
void   savedata();
void   loaddata();
void   exitprg();
void   delay();
void   switchChannels();
void   changeNumbofMeas();
void   changeLines();
void   changeScanRange();
void   changeGain();
void   changexVoltStart();
void   changeyVoltStart();
void   changeDelayCtr();
long   calcxDAvalue(int xCount);
long   calcyDAvalue(int yCount);
int    checkValues();

/****************************************************************************/
/* BEGIN MAIN MODULE */

main()
{
 /* LOCAL VARIABLE */
 char choice;

 /* INITIALIZE HARD/SOFTWARE AND ESTABLISH COMMUNICATION TO
 BOARD */
 opendevice();
 gethandle();
 while (!kbhit());

 /* LOOP FOR PRINTING MENU AND GETTING CHOICE */
 Menu:
  _clearscreen (_GCLEARSCREEN);
  printmenu();
  choice = getchar();
  switch (choice)
  {
   case ('d')  : { changeDelayCtr();
                   break; }
   case ('x')  : { changexVoltStart();
                   break; }
   case ('y')  : { changeyVoltStart();
                   break; }
   case ('g')  : { changeGain();
                   break; }
   case ('r')  : { changeScanRange();
                   break; }
   case ('n')  : { changeNumbofMeas();
                   break; }
   case ('m')  : { changeLines();
                   break; }
   case ('c')  : { switchChannels();
                   break; }
   case ('s')  : { savedata();
                   break; }
   case ('l')  : { loaddata();
                   break; }
```

```
case ('p') : { _clearscreen (_GCLEARSCREEN);
                _setvideomode (modes[13]);
                _getvideoconfig (&vc);
                _setlogorg(270, 30);
                printscreen(NUMBYSTEPS-1);
                printf ("\n Press <space> to return to menu !");
                while (!kbhit());
               _setvideomode (_DEFAULTMODE);
                break; }
case ('e')   : { exitprg();
                break; }
case ('h')   : { if (checkValues())
                {
                  while (!kbhit());
                  break;
                }
                _clearscreen(_GCLEARSCREEN);
                _setvideomode (modes[13]);
                _getvideoconfig (&vc);
                _setlogorg(270, 30);
                printf ("\n Press <q> to quit !");
                scansample(1, 1);
                printf ("\n Press <space> to return to menu !");
                while (!kbhit());
                _setvideomode (_DEFAULTMODE);
                break; }
case ('a')   : { if (checkValues())
                {
                 while (!kbhit());
                 break;
                }
                _clearscreen(_GCLEARSCREEN);
                _setvideomode (modes[13]);
                _getvideoconfig (&vc);
                _setlogorg(270, 30);
                printf ("\n Press <q> to quit !");
                printf ("\n Press <space> to stop scanning !");
                scansample(1, 0);
                printf ("\n Press <space> to return to menu !");
                while (!kbhit());
                _setvideomode (_DEFAULTMODE);
                break; }
case ('u')   : { if (checkValues())
                {
                        while (!kbhit());
                        break;
                      }
                      _clearscreen(_GCLEARSCREEN);
                      _setvideomode (modes[13]);
                      _getvideoconfig (&vc);
                      _setlogorg(270, 30);
                      printf ("\n Press <q> to quit !");
                      scansample(0, 1);
                      printf ("\n Press <space> to return to menu !");
                      while (!kbhit());
                      _setvideomode (_DEFAULTMODE);
                      break; }
     case ('o')   : { if (checkValues())
                      {
                        while (!kbhit());
                        break;
                      }
                      _clearscreen(_GCLEARSCREEN);
                      _setvideomode (modes[13]);
                      _getvideoconfig (&vc);
                      _setlogorg(270, 30);
                      printf ("\n Press <q> to quit !");
                      printf ("\n Press <space> to stop scanning !");
                      scansample(0, 0);
                      printf ("\n Press <space> to return to menu !");
                      while (!kbhit());
                      _setvideomode (_DEFAULTMODE);
                      break; }
   }
  goto Menu;

  return;
}

/*********************************************************************/
/* INITIALIZE THE HARDWARE/SOFTWARE */

void opendevice()
{
 /* LOCAL VARIABLE */
 int Err;                /* FUNCTION RETURN ERROR FLAG */

 if (( Err = DAS1600_DevOpen ( "DAS1600.cfg", &NumberOfBoards )) !=0 )
```

```
  {
   putch (7);   /* RING BELL */
   printf ( " Error %X during DevOpen\n ", Err );
   exit (Err);
  }
  else
  {
   printf ("Device initialized\n");
  }

  return;
}

/*********************************************************************/
/* ESTABLISH COMMUNICATION WITH THE DRIVER THROUGH A DEVICE
HANDLE */

void gethandle()
{
 /* LOCAL VARIABLE */
 int Err;         /* FUNCTION RETURN ERROR FLAG */

 if (( Err = DAS1600_GetDevHandle( 0, &DAS1600 )) != 0 )
 {
  putch (7);   /* RING BELL */
  printf ( " Error %X during GetDevHandle ", Err );
  exit(1);
 }
 else
 {
  printf ("Communication established\n");
 }

 return;
}

/*********************************************************************/
/* PRINT MENU */

void printmenu()
{
 printf (" X-voltage starting at (-5.00 - +4.99): %.2f\n", xVoltStart);
 printf (" Y-voltage starting at (-5.00 - +4.99): %.2f\n", yVoltStart);
 printf (" scanning Range: %.2f\n", xVoltScanRange);
 printf (" averaging over N measurements: %d\n", scanTimes);
```

```
  printf (" print screen after M lines: %d\n", printLines);
  printf (" Delay-counter: %ld\n", delayCtr);
  printf (" Gain-nr.: %d\n", gain);
  printf (" Channels: x-chn%dy-chn%d\n", xChannel, yChannel);
  printf ("\n");
  printf (" min: %d max: %d\n", zADvalueMin, zADvalueMax);
  printf ("\n");
  printf (" single scanning at ‘constant Height’ mode\n");
  printf (" single scanning at ‘constant cUrrent’ mode\n");
  printf (" continuous scAnning at ‘constant height’ mode\n");
  printf (" cOntinuous scanning at ‘constant current’ mode\n");
  printf (" Savedata\n");
  printf (" Loaddata\n");
  printf (" Printscreen\n");
  printf (" Exit\n");
  printf ("\n your choice: ");

return;
}

/**********************************************************************/
/* SCAN SAMPLE */

void scansample(int channel, int kbFlag)
{
 long   xDAvalue;       /* STORAGE FOR X-D/A VALUE */
 long   yDAvalue;       /* STORAGE FOR Y-D/A VALUE */
 int    xCount;         /* COUNTER FOR X-VOLTAGE-LOOP */
 int    yCount;         /* COUNTER FOR Y-VOLTAGE-LOOP */
 long   zTotal = 0;     /* SUM OF Z-STORAGES */
 int    avgCount;       /* COUNTER OF LOOP FOR AVERAGING OVER
                        SEVERAL MEASUREMENTS */
 char   keyhit;

 Scan:
  for (yCount=0; yCount<NUMBYSTEPS; yCount++)     /* OUTER LOOP */
                                                  /* CONTROLS   */
  {                                               /* Y-VOLTAGE  */
   fflush(stdin);
   yDAvalue = calcyDAvalue (yCount);
   writeyDAvalue(yDAvalue);

   for (xCount=0; xCount<NUMBXSTEPS; xCount++)  /* INNER LOOP */
   {                                            /* CONTROLS   */
    xDAvalue = calcxDAvalue (xCount);           /* X-VOLTAGE  */
```

```
        writexDAvalue (xDAvalue);
        delay();
        zTotal = 0;

        for (avgCount=1; avgCount<= scanTimes; avgCount++)  /* MEASUREMENTS */
        {                                                   /* FOR ONE SINGLE */
          readADvalue(channel);                             /* DATA POINT     */

          /* STRIP CHANNEL TAG (LEAST SIGNIFICANT 4 BITS) OF Advalue */
          zTotal += ADvalue >> 4;
        }
        zADvalue[xCount][yCount] = (zTotal >> powerofTwo);
      }

      if ((((yCount+1)/printLines)*printLines)==(yCount+1))
      {
        printscreen(yCount);
      }

      if (kbhit())
      {
        kbFlag = 1;
        keyhit = getch();
        if (keyhit == 'q')
        {
          return;
        }
      }
    }

    calczADvalueMin(NUMBYSTEPS-1);
    calczADvalueMax (NUMBYSTEPS-1);
    printscreen(NUMBYSTEPS-1);

  if (kbFlag == 0)
  {
    goto Scan;
  }

  return;
}

/**************************************************************************/
/* OUTPUT xDAvalue TO DAC #xChannel */
```

```
void writexDAvalue(long xDAvalue)
{
 /* LOCAL VARIABLE */
 int Err;                  /* FUNCTION RETURN ERROR FLAG */

 /* THE DAC VALUE TO OUTPUT MUST BE CORRECTLY ALIGNED FOR
 OUTPUT; I.E.  THE ACTUAL 12-BIT DATA MUST BE IN THE UPPER 12 BITS
 OF THE USER INTEGER */
 xDAvalue = xDAvalue << 4;

 if ((Err = K_DAWrite (DAS1600, xChannel, xDAvalue)) != 0)
 {
  putch(7);
  printf (" Error in K_DAWrite operation.");
  exit(1);
 }

 return;
}

/*********************************************************************/
/* OUTPUT yDAvalue TO DAC #yChannel */

void writeyDAvalue(long yDAvalue)
{
 /* LOCAL VARIABLE */
 int Err;                  /* FUNCTION RETURN ERROR FLAG */

 /* THE DAC VALUE TO OUTPUT MUST BE CORRECTLY ALIGNED FOR
 OUTPUT; I.E.  THE ACTUAL 12-BIT DATA MUST BE IN THE UPPER 12 BITS
 OF THE USER INTEGER */
 yDAvalue = yDAvalue << 4;

 if ((Err = K_DAWrite (DAS1600, yChannel, yDAvalue)) != 0)
 {
  putch(7);                          /* RING BELL */
  printf (" Error in K_DAWrite operation.");
  exit(1);
 }

 return;
}

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

```
/* SAMPLE AND READ CHANNEL channel AT GAIN gain AND STORE SAMPLE
IN ADvalue */

void readADvalue(int channel)
{
 if ((Err = K_ADRead (DAS1600, channel, gain, &ADvalue)) != 0)
 {
  putch(7);                    /* RING BELL */
  printf ("Error in K_ADRead operation.");
  exit(1);
 }

 return;
}

/********************************************************************/
/* CALCULATE MINIMUM OF COLLECTED DATA POINTS */

void calczADvalueMin(int yCount)
{
 /* LOCAL VARIABLES */
 int xCtr, yCtr;

 zADvalueMin = zADvalue[0][0];
 for (yCtr=0; yCtr<=yCount; yCtr++)
 {
  for (xCtr=0; xCtr<NUMBXSTEPS; xCtr++)
  {
   if (zADvalue[xCtr][yCtr] < zADvalueMin)
   {
    zADvalueMin = zADvalue[xCtr][yCtr];
   }
  }
 }

 return;
}

/********************************************************************/
/* CALCULATE MAXIMUM OF COLLECTED DATA POINTS */

void calczADvalueMax(int yCount)
{
 /* LOCAL VARIABLES */
 int xCtr, yCtr;
```

```
  zADvalueMax = zADvalue[0][0];
  for (yCtr=0; yCtr<=yCount; yCtr++)
  {
    for (xCtr=0; xCtr<NUMBXSTEPS; xCtr++)
    {
      if (zADvalue[xCtr][yCtr] > zADvalueMax)
      {
        zADvalueMax = zADvalue[xCtr][yCtr];
      }
    }
  }

  return;
}

/*********************************************************************/
/* CALCULATE VALUE FOR WRITING TO X-CHANNEL */

long calcxDAvalue (int xCount)
{
  /* LOCAL VARIABLES */
  int    xDAvalueInt;
  long xDAvalue;

  xDAvalueInt = (xVoltStart + xVoltScanRange*xCount/(NUMBXSTEPS-1))
                    * 4095/XVOLTTOTALRANGE + 2048;
  xDAvalue = xDAvalueInt;

  return (xDAvalue);
}

/*********************************************************************/
/* CALCULATE VALUE FOR WRITING TO Y-CHANNEL */

long calcyDAvalue(int yCount)
{
  /* LOCAL VARIABLES */
  int     yDAvalueInt;
  long  yDAvalue;

  yDAvalueInt = (yVoltStart + yVoltScanRange*yCount/(NUMBYSTEPS-1))
                    * 4095/YVOLTTOTALRANGE + 2048;
  yDAvalue = yDAvalueInt;

  return (yDAvalue);
```

```
}

/*******************************************************************/
/* PRINT COLLECTED DATA POINTS ON SCREEN */

void printscreen(int yCount)
{
 /* LOCAL VARIABLES */
 int    xscreenCtr,  yscreenCtr;
 int    xscreenCtr2, yscreenCtr2;
 int    pixelColor;
 float  norm;
 float  diff;
 float  floatMax,    floatMin;
 float  calc;

 calczADvalueMin(yCount);
 calczADvalueMax(yCount);
 if (zADvalueMin == zADvalueMax)
 {
  return;
 }

 floatMax = zADvalueMax;
 floatMin  = zADvalueMin;
 diff      = floatMax - floatMin ;
 norm      = 16.0/diff;

 for (yscreenCtr=0; yscreenCtr<=yCount; yscreenCtr++)
 {
  yscreenCtr2 = 2 * yscreenCtr;
  for (xscreenCtr=0; xscreenCtr<NUMBXSTEPS; xscreenCtr++)
  {
   xscreenCtr2 = 2 * xscreenCtr;
   pixelColor = (zADvalue[xscreenCtr][yscreenCtr]-floatMin)*norm;
   _setcolor(pixelColor);
   _setpixel(xscreenCtr2, yscreenCtr);
   _setpixel(xscreenCtr2+1, yscreenCtr);
   _setpixel(xscreenCtr2, yscreenCtr2+1);
   _setpixel(xscreenCtr2+1, yscreenCtr2+1);
  }
 }

 return;
}
```

```
/*********************************************************************/
/* SAVE DATA TO DISK */

void savedata()
{
 /* LOCAL VARIABLES */
 char   fileName[9];
 char   fileName2[9];
 char   *dataFileName;
 char   *infoFileName;
 FILE   *dataFptr;
 FILE   *infoFptr;
 char   answer;
 int    Count, yCount;

 fflush(stdin);
 printf (" Enter name of file:  ('e' to exit)\n");
 gets(fileName);
 strcpy (fileName2, fileName);

 if (fileName[0]=='e')                    /* TEST OF EXIT CONDITION */
 {
  if (!fileName[1])
  {
   return;
  }
 }

 dataFileName = strcat(fileName, dat);
 infoFileName = strcat(fileName2, inf);

 if ((dataFptr=fopen(dataFileName, "r"))!=NULL)
 {
  printf (" File %s already exists.\n", dataFileName);
  printf (" Overwrite old file?\n");
  answer = getchar();
  if (answer != 'y')
  {
   return;
  }
 }

 if ((dataFptr=fopen(dataFileName, "w"))==NULL)
 {
  printf (" *** Error opening %s ***\n", dataFileName);
  while (!kbhit());
  return;
 }

 for (yCount=0; yCount<NUMBYSTEPS; yCount++)
 {
  for (xCount=0; xCount<NUMBXSTEPS; xCount++)
  {
   fprintf (dataFptr, "%d ", zADvalue[xCount][yCount]);
  }
 }
 fprintf (dataFptr, "%d ", zADvalueMin);
 fprintf (dataFptr, "%d ", zADvalueMax);

 fclose(dataFptr);

 if ((infoFptr=fopen(infoFileName, "w"))==NULL)
 {
  printf ("*** Error opening %s ***\n", infoFileName);
  while (!kbhit());
  return;
 }

 /* BODY OF WRITING TO INFOFILE */

 fclose(dataFptr);

 return;
}

/*********************************************************************/
/* LOAD DATA FROM DISK */

void loaddata()
{
 /* LOCAL VARIABLES */
 char   fileName[9];
 char   fileName2[9];
 char   *dataFileName;
 char   *infoFileName;
 FILE   *dataFptr;
 FILE   *infoFptr;
 int    xCount, yCount;
 int    zADvalueStorage;
```

```
fflush(stdin);
printf (" Enter name of file:  ('e' to exit)\n");
gets(fileName);
strcpy (fileName2, fileName);

if (fileName[0]=='e')                          /* TEST OF EXIT CONDITION */
{
 if (!fileName[1])
 {
  return;
 }
}

dataFileName = strcat(fileName, dat);
infoFileName = strcat(fileName2, inf);

if ((dataFptr=fopen(dataFileName, "r"))==NULL)
{
 printf (" *** Error opening %s ***\n", dataFileName);
 while (!kbhit());
 return;
}

for (yCount=0; yCount<NUMBYSTEPS; yCount++)
{
 for (xCount=0; xCount<NUMBXSTEPS; xCount++)
 {
  fscanf(dataFptr, "%d ", &zADvalueStorage);
  zADvalue[xCount][yCount] = zADvalueStorage;
 }
}
fscanf(dataFptr, "%d ", &zADvalueMin);
fscanf(dataFptr, "%d ", &zADvalueMax);

fclose(dataFptr);

if ((infoFptr=fopen(infoFileName, "r"))==NULL)
{
 printf (" *** Error opening %s ***\n", infoFileName);
 while (!kbhit());
 return;
}

/* BODY OF READING FROM INFOFILE */
```

```
 fclose(dataFptr);

 return;
}

/***********************************************************************/
/* EXIT PROGRAMM */

void exitprg()
{
 /* LOCAL VARIABLE */
 char answer;

 printf (" Press 'y' to exit");
 scanf(" %c", &answer);
 if (answer == 'y')
 {
  exit(0);
 }

 return;
}

/***********************************************************************/
/* LOOP FOR DELAY */

void delay()
{
 /* LOCAL VRIABLES */
 long i, k;

 for (i=0; i<=delayCtr; i++)
 {
  k=i;
 }

 return;
}

/***********************************************************************/
/* SWITCH CHANNELS FOR X- AND Y-WRITE */

void switchChannels()
{
 if (xChannel==0)
 {
    xChannel=1;
    yChannel=0;
  }
  else
  {
    xChannel=0;
    yChannel=1;
  }

  return;
}

/**********************************************************************/
/* CHANGE NUMBER OF MEASUREMENTS */

void changeNumbofMeas()
{
  printf (" Average over how many measurements ?\n");
  printf (" Two to the power of: ");
  scanf(" %d", &powerofTwo);
  scanTimes = pow (2, powerofTwo);

  return;
}

/**********************************************************************/
/* CHANGE : PRINT SCREEN AFTER HOW MANY LINES */

void changeLines()
{
  printf (" Print screen after how many lines? ");
  scanf(" %d", &printLines);

  return;
}

/**********************************************************************/
/* CHANGE SCANRANGE */

void changeScanRange()
{
  printf (" Scanrange: ");
  scanf(" %f", &xVoltScanRange);
  yVoltScanRange = xVoltScanRange;
```

```
 return;
}

/**********************************************************************/
/* CHANGE GAIN */

void changeGain()
{
 printf (" Gain-Nr (0-3): ");
 scanf(" %d", &gain);

 return;
}

/**********************************************************************/
/* CHANGE STARTING VALUE OF XVOLTAGE */

void changexVoltStart()
{
 printf (" x-Voltage starting at: ");
 scanf(" %f", &xVoltStart);

 return;
}

/**********************************************************************/
/* CHANGE STARTING VALUE OF YVOLTAGE */

void changeyVoltStart()
{
 printf (" y-Voltage starting at: ");
 scanf(" %f", &yVoltStart);

 return;
}

/**********************************************************************/
/* CHANGE DELAY COUNTER */

void changeDelayCtr()
{
 printf (" Delay-Counter: ");
 scanf(" %ld", &delayCtr);

 return;
}
```

```
/***************************************************************************/
/* CHECK IF VALUES FOR VARIABLES ARE IN VALID RANGE */

int checkValues()
{
  if ((xVoltStart<XVOLTMIN) || (xVoltStart>XVOLTMAX))
  {
    printf (" *** Invalid starting value for x-voltage ***\n");
    return (1);
  }

  if ((yVoltStart<YVOLTMIN) || (yVoltStart>YVOLTMAX))
  {
    printf (" *** Invalid starting value for y-voltage ***\n");
    return (1);
  }

  if   (((xVoltStart+xVoltScanRange)<XVOLTMIN)  ||
        ((xVoltStart+xVoltScanRange)>XVOLTMAX)||
        ((yVoltStart+yVoltScanRange)<YVOLTMIN)  ||
        ((yVoltStart+yVoltScanRange)>YVOLTMAX))
  {
    printf (" *** Invalid voltage range ***\n");
    return (1);
  }

  if ((gain>3) || (gain<0))
  {
    printf (" *** Invalid value for gain-nr.  ***\n");
    return (1);
  }
  if (delayCtr<0)
  {
    printf (" *** Delay-counter must be positive! ***\n");
    return (1);
  }
  if (printLines<=0)
  {
    printf (" *** printLines must be positive! ***\n");
    return (1);
  }
  return (0);
}

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