11-14-2002

System Integration and Testing using Object Oriented Programming based Control

Prashant P. Datar
University of South Florida

SYSTEM INTEGRATION AND TESTING USING OBJECT ORIENTED PROGRAMMING BASED CONTROL

by

PRASHANT P. DATAR

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering
Department of Electrical Engineering
College of Engineering
University of South Florida

Major Professor: Wilfrido A. Moreno, Ph.D.
James T. Leffew, Ph.D.
Grisselle Centeno, Ph.D.

Date of Approval:
November 14, 2002

Keywords: device communication, environment control, modular programming, software instrument control, visual c++

Table Of Contents

List Of Tables

List Of Figures

List Of Acronyms

IC	Integrated Circuit
CMP	Chemical Mechanical Planarization
OOP	Object Oriented Programming
OOPL	Object Oriented Programming Language
USB	Universal Serial Bus
NIC	Network Interface Card
GPIB	General Purpose Interface Bus
IEEE	Institute of Electrical and Electronic Engineers
PC	Personal Computer
ASCII	American Standard Code for Information Interchange
DTE	Data Terminal Equipment
DCE	Data Communication Equipment
TD	Transmit Data
RD	Receive Data
RTS	Request To Send
CTS	Clear To Send
DSR	Data Set Ready
DTR	Data Terminal Ready
CD	Carrier Detect
RI	Ring Indicator

HPIB	Hewlett Packard Interface Bus
TTL	Transistor Transistor Logic
DIO	Data Input Output
DAV	Data Available
NFRD	Not Ready For Data
NDAC	Not Data Accepted
ATN	Attention
EOI	End Or Identify
IFC	Interface Clear
REN	Remote Enable
SRQ	Service Request
RAM	Random Access Memory
DAQ	Data Acquisition Card
PI	Proportional-Integral
PD	Proportional-Derivative
PID	Proportional-Integral-Derivative
MFC	Microsoft Foundation Classes
OCX	Object Control
COM	Component Object Model
MSE	Mean Square Error
DOF	Degree Of Freedom

System Integration And Testing Using Object Oriented Programming Based Control

Prashant P. Datar

ABSTRACT

Various techniques are used in the process of software development. The requirements of the system being designed and the constraints dictate the selection of a particular method to be used. This thesis attempts to explain the various types of development techniques available to software designers and programmers. It places specific emphasis on the Object Oriented style of design that is presently widely used in all areas of industry.

Object Oriented Programming (OOP) involves a number of new concepts that make software design and development more modular. The actual problem is broken down into a number of smaller components and the functionality of each component is coded separately. These pieces of code are then integrated to form the final application. All the concepts that make this type of programming possible are explained.

The thesis presents a detailed account of the development process of a system used to make measurements on polyurethane pads that are used in the Chemical Mechanical Planarization (CMP) process. The setup uses a combination of a number of instruments to provide excitation to the pad and measure its response. A computer controls all these instruments using a single application. Microsoft Visual C++ was used to develop this application. It makes extensive use of a Graphic User Interface (GUI), Microsoft Foundation Classes (MFC) and driver libraries from instrument manufacturers in order to present a user-friendly interface to the operator.

System Integration, which is the technique used to make the instruments involved interact with the software is explained. The application involves the use of a number of C++ classes and dialog boxes. Each of these is explained along with the underlying algorithms.

1 Introduction

The objective of this work was to explain the various stages involved in the development of software that interactively manages an industrial or research based system. This type of application software is significantly different from programs that are written to speed up computation or manage a large inventory. In these types of applications, the data is presented to the computer and the software either manipulates or organizes the information in a manner required to provide the desired results.

This thesis deals with software development techniques that provide automation for applications that control and operate equipment so as to minimize the level of supervision required. The development starts with a clear understanding of the problem, evaluating the various method of programming, choosing a means of implementation and finally carrying out extensive testing. A system built to study the characteristics of polyurethane pads that are used in Chemical Mechanical Planarization (CMP) explains the entire process. [1]

1.1 Wafer Polishing

Pads made of polyurethane material are used in the Integrated Circuit (IC) industry to polish wafers between successive manufacturing steps. An integrated circuit is manufactured using a series of steps. Each step adds or prepares the wafer for a future step to add an element or group of elements that

form the actual circuit from the wafer design. Some of the steps in the process deposits material on top of the devices already laid on the wafer (deposition) or increase the thickness of some, but not all, of the materials already deposited (growth), while other steps cut or perforate some areas in order to create a communication path between layers (etching).

Figure 1.1: Integrated Circuit Wafer Polishing

The problem arises when one of the processes does not result in a planar surface. When that happens the next steps in the process can result in a badly laid area, a deformation that results in a bad connection in the circuit or an unwanted connection. In order to correct a problem, there must be a way to ensure that all the layers created in the wafer are free of deformations, which will ensure a high yield and reliability in the production of the IC.

CMP is the process whereby most of the deformations in a wafer surface are eliminated. CMP results in a planar surface layer over which any other layer

can be laid, which results in a wafer surface with a very low possibility of deformation.

Changes in physical factors such as temperature and humidity produce physical effects on the CMP pads. When this happens the surface-uniformity of wafers polished using these pads cannot be guaranteed. Therefore, it is necessary to determine how the pads respond to ambient changes and control the polishing process in such a way as to maintain wafer quality. [1]

1.2 System Requirements

A large number of instruments were used to carry out experiments on the pads. A single program was to control all the instruments through a host computer. The software was to be designed in such a way that experiments required minimal user intervention. The operator only had to specify the area under study and the required temperature and humidity for the run. The equipment had to be controlled so the experiment was carried out under constant conditions. The program required the ability to store readings to a file and manipulate the readings statistically in order to produce results. The algorithms and concepts used to develop the software are explained in the following chapters. Results of testing and conclusions are presented in chapter six.

2 Programming Concepts

Programming techniques can be classified depending upon their efficiency and level of complexity.

- Unstructured programming
- Procedural programming
- Modular programming
- Object Oriented programming

2.1 Unstructured Programming

Programming starts with the construction of small and simple programs. Such programs usually consist of a single *main-program*, which contains a sequence of statements that modify data that is global to the program. This type of programming presents a considerable disadvantage once the physical size of the program starts getting larger. For example, consider a certain sequence of statements that is to be repeated at different locations in a program. This sequence will have to be copied at the different locations. This requirement has lead to the idea of extracting such sequences, naming them and providing a method to call and return from these *procedures*. [2]

2.2 Procedural Programming

Procedural programming places recurring sequences of statements in a single place. A *procedure-call*, hereafter referred to simply as a *call*, is used to

invoke the procedure. After the sequence is processed, flow of control proceeds to the position immediately after the position where the call was made.

Procedural programming allows programs to be written in a more structured and error free format. Each procedure executes a particular part of the program. Therefore, debugging of the program is simplified and it is easier to locate and correct any errors. Therefore, a program can be viewed as a sequence of procedural calls. The main program is responsible for passing data to the individual calls. The data is processed by the procedures and the results are presented once the program has finished.

The procedural approach results in a single program that is divided into small pieces called procedures. To enable the use of general procedures or groups of procedures in other programs, they must be individually available. For that reason, modular programming allows grouping of procedures into modules. [2]

2.3 Modular Programming

In modular programming, procedures of a common functionality are grouped together into separate *modules*. Therefore, a program no longer consists of a single part. With modules it is divided into several smaller parts that interact through procedure calls in order to form the whole program.

Each module can have its own data. This allows each module to manage an internal *state*, which is modified by calls to procedures of the module. However, there is only one state per module and each module exists at most once in the whole program. [2]

2.4 Object Oriented Programming (OOP)

Object Oriented Design is a software design method that models the characteristics of abstract or real objects through the use of classes and objects. Objects are a key to an understanding of OOP. An object is a software bundle of related variables and methods. Software objects are often used to model real-world objects found in everyday life. Important terms and concepts used in OOPS require special attention.

2.4.1 Objects

All real world objects have two characteristics, state and behavior. If a car is considered as an object it will have a current state, which could be indicated by parameters such as gear, engine speed, number of wheels and number of gears. Additionally, the car will exhibit a behavior, which could be described by actions such as change of gear, brake application or change in engine speed. Software objects are modeled after real world objects. Software objects also possess states and exhibit behavior. A software object describes its state through the use of one or more variables. A variable is an item of data named by an identifier. A software object implements its behavior with methods. A method is a function (subroutine) associated with an object.

Variables describe the state of an object while methods describe its behavior. Changing the values of the associated variables changes the state of an object. The variables of an object are not directly accessible to the outside world. They can only be accessed through the object's methods. Therefore, the methods form a kind of protective casing for the variables. This packaging of

variables is referred to as *encapsulation*. Encapsulation is a simple, yet powerful, idea that provides two advantages to software programmers.

- *Modularity*: The source code for an object can be written and maintained independently of the source code for other objects. Therefore, an object can be easily accessed by various parts of the system.
- *Information Hiding*: An object has a public interface that other objects can use to communicate with it. However the object can maintain private information and methods that can be changed at any time without affecting the other objects that depend on it. [3]

2.4.2 Messages

Software objects interact and communicate with each other through the use of *messages*. In a program, objects usually appear as components of a larger application that contains many objects. Interactions between these objects achieve complex behavior and higher order functionality. Objects communicate with each other by using messages. Along with the message the receiving object needs information that tells it what to do. There are three components that comprise a message.

- Object to which the message is addressed
- Method to be executed
- Parameters required by the method

Messages provide two important benefits. Since an object's behavior is expressed through its methods, message passing supports all possible interactions between objects. Additionally, objects are not required to be in the

same process or even on the same machine in order to send and receive messages. [3]

2.4.3 Class

A class is a blueprint or prototype, which defines the variables and the methods that are common to all objects of a certain kind. A class defines the variables and methods that determine the state and behavior of objects. However, different objects belonging to a class will not have the same state simultaneously. For example, a carmaker's particular car model can be considered as a class with each car of that model an object. Although all the cars will look similar, their states will be different at a given time. In OOPS terminology an object is said to be an *instance* of a class.

Classes can also define *class variables*. These are variables whose value remains the same for all objects of the class. If one object changes this value all objects receive the new value. This saves memory since all the objects of one class share the variable and hence only one copy of that variable needs to be created. For example, if there is a variable that stores the number of gears in a car, the value will be the same for all objects of that car class. Hence, only one class variable needs to be declared for this purpose. Similarly a *class method* can also be defined. Class methods can be invoked directly from a class instead of having to be invoked in each individual instance of the class. [3]

2.4.4 Inheritance

A class inherits its state and behavior from its *super-class*. Inheritance provides a powerful and natural mechanism for organizing and structuring

software programs. Objects are defined in terms of classes. If automobiles is considered as a class, then cars, trucks and busses can be considered as derived classes from the automobiles class. Automobiles is a super-class cars, trucks and busses become *sub-classes* of the automobile class. Each sub-class inherits variables and methods from its super-class. Sub-classes can *override* inherited qualities and provide specialized implementations.

The number of inheritances is not limited to one. The inheritance tree or class hierarchy can be as tall as the programmer wishes. Inheritance offers various advantages.

Sub-classes provide specialized behaviors from the basis of common elements provided by the super-class. Through the use of inheritance, programmers can reuse the code in the super-class many times. Programmers can implement super-classes, which are called *abstract classes*, in order to define generic behaviors. The abstract super-class defines and may partially implement the behavior. However, much of the class is undefined and unimplemented. Other programmers fill in the details with specialized sub-classes. [3]

2.5 Object Oriented Languages (OOPLs)

The industry offers programmers a wide choice of OOPLs such as C++, Java, Smalltalk, Delphi, Eiffel and Python. Each of these languages support the aforementioned concepts used in Object Oriented Programming (OOP). However, the most widely used, of these programming languages, are C++ and Java. The C++ language was first designed and implemented by Bjarne

Stroustrup. Java is an OOPL introduced by Sun Microsystems.

Alan Kay invented Smalltalk in the 1970s at Xerox's Palo Alto Research Center. It is unique in that it uses nouns from the English language for objects and verbs for messages. This language was also the inspiration and technical basis for the Macintosh and subsequent windowing based systems.
Delphi is another true object oriented and compiled language. However, it only allows Single Inheritance. It is a form-based language with distributable components with syntax similar to BASIC.

2.6 Reasons For Choosing VC++

Object Oriented Programming concepts were developed in the late 1980s. At that time C did not support the technology and hence could not garner any of its benefits. C++ was developed in order to overcome this shortcoming of C. As the technology grew new graphical interfaces were developed. An object oriented as well as a visual programming language was needed to keep pace with the new standard. Therefore, Visual C++ was born. Visual C++ is C++ with capabilities that support the graphical user interface technology required for OLE, OCX, ActiveX and Database programming. Microsoft offers VC++ as part of its Visual Studio. Additionally, Ms-VC++ programmers can also make good use of Microsoft Foundation Classes (MFCs) in applications. [4]

The nature of this project was such that it needed to make use of hardware to monitor and control the testing. VC++ was chosen since it is a high-level language that allows the hardware to be accessed in a powerful and efficient manner. Additionally, it offers standard libraries to enable serial and

parallel communication, which proved to be a great convenience. VC++ also possesses the capability to create low-level code that is associated with such things as Operating Systems, Device drivers, Dynamic Link Libraries (DLLs), Internet Servers, Objects and Database Systems. VC++ uses MFCs effectively, which makes the creation and customization of dialog boxes very easy and straightforward. Dialog boxes allow the application interface to be made user friendly.

3 Communication Interfaces

Various interfaces exist for facilitating communication between a host computer and its peripheral devices. These range from the serial and parallel interfaces developed over time to the ones more recently evolved such as Universal Serial Bus (USB), Network Interface Cards (NIC), Optical and Wireless interfaces. Factors such as the required speed of data transfer, distance from the host computer and cost decide which interface is best suited for a particular application. The Serial interface, where the data is transferred bit by bit from one machine to another, is typically the slowest. In the case of parallel transmission, the data is transferred in Bytes (8-bits simultaneously). Almost every PC has these interfaces as residents. Ports such as optical and USB are available on higher end machines where the time taken for data transfer is critical. Optical interfaces are typically used for accessing storage arrays and connecting high-speed networking equipment. NIC cards and wireless NIC cards are used to build scalable and reliable networks of computers.

This research required extensive use of RS-232 protocols to control instruments. Additionally, the General Purpose Interface Bus (GPIB), a parallel communication interface from the Institute of Electrical and Electronic Engineers, was implemented to interface those devices that offered a parallel interface capability. A proper understanding of both these protocols was required. The

following sections explain these two widely used techniques in device communication.

3.1 Serial Communication

All IBM Personal Computers (PC) and compatible computers are typically equipped with two serial ports. A Serial port sends and receives data one bit at a time over two wires. Two-way (Full Duplex) communications is possible with only three wires; one to send, one to receive and a common signal ground. The serial port on a PC is usually full duplex, which is capable of both transmission and reception. There are a large number of serial communication protocols available such as RS-232, RS-485, RS-422, and RS-449. These protocols differ on the basis of control signals and signal levels (voltages) employed. This chapter concentrates on the most widely used, RS-232C, protocol. The following subsections explain the terminology commonly used in serial communication.

3.1.1 Data Bits

The measurement of the actual data processed in a transmission is given in terms of the number of data bits transferred. When the computer sends a packet of information, the amount of actual data may not be 8 bits. Standard values for the data packets are 5, 7 and 8 bits. The setting chosen depends upon the type of information being transferred. For example, standard ASCII uses values from 0 to 127, which requires 7 bits. Extended ASCII uses values from 0 to 255, which requires 8 bits. If the data being transferred is simple text (standard ASCII), then 7 bits of data per packet is sufficient for communication. A packet refers to a single byte transfer, including start/stop bits, data bits and a

parity bit. Since the number of actual bits depends on the protocol selected, the term packet is used to cover all instances.

3.1 2 Stop Bits

Stop bits are used to signal the end of communication for a single packet. Typical values are 1, 1.5 and 2 bits. Since the data is clocked across the lines and each device has its own clock, it is possible for the two devices to become slightly unsynchronized. Therefore, the stop bits not only indicate the end of transmission they also give the computers some room for error in the clock speeds. The more bits that are used for stop bits the greater the lenience in synchronizing the different clocks.

3.1.3 Parity Bits

These bits are used as a simple form of error checking. There are four types of parity: even, odd, marked, and spaced. The option of using no parity is also available. For even and odd parity, the serial port sets the parity bit, which is the last bit after the data bits, to a value to ensure that the transmission has an even or odd number of logic high bits. For example, when even parity is chosen, the parity bit is transmitted with a value of zero if the number of preceding one's is an even number. For the binary value of 01100011 the parity bit would be zero. If even parity were in effect and the binary number 11010110 were sent, then the parity bit would be one. Odd parity is just the opposite. The parity bit is zero when the number of one bits in the preceding word is an odd number. Parity error checking is very rudimentary. Parity can detect single bit errors but it cannot actually locate the bit that is in error. Also, if an even number of bits were

in error then the parity bit would not reflect any error at all. Marked and spaced parity does not actually check the data bits. They simply set the parity bit high for marked parity or low for spaced parity. This allows the receiving device to know the state of a bit, which enables it to determine if noise is corrupting the data or if the clocks of the transmitting and receiving devices are out of synchronization.

3.1.4 DTE And DCE Devices

DTE stands for Data Terminal Equipment and *DCE* stands for Data Communication Equipment. The PC is a DTE device while the Modem is typically a DCE device. DTE devices use a male connector while DCE devices use a female connector.

3.1.5 Baud Rate

Baud rate is a measure of the number of times per second a signal in a communications channel varies or makes a transition between states. States are defined in terms of frequencies, voltage levels, or phase angles. One baud is one such change. If the signal on a line changes three hundred times per second, the baud rate is 300 baud. Baud-rate differs from bit-rate. Baud rate gives the rate of transmission of symbols unlike bit-rate. The number of bits in a symbol depends upon the modulation scheme used.

3.1.6 RS-232 Protocol

The signals that control data transfer for the RS-232 protocol are presented in Figures 3.1 and 3.2 on page 16.

The TD (Transmit Data) wire is the one through which data from a DTE device is transmitted to a DCE device. The TD line is kept in a mark condition by

the DTE device when it is idle. The RD (Receive Data) wire is the one on which data is received by a DTE device. The DCE device keeps this line in a mark condition when idle.

1	Protective Ground
2	Transmitted Data (TD) Outgoing Data (from a DTE to a DCE)
3	Received Data (RD) Incoming Data (from a DCE to a DTE)
4	Request To Send (RTS) Outgoing flow control signal controlled by the DTE
5	Clear To Send (CTS) Incoming flow control signal controlled by the DCE
6	Data Set Ready (DSR) Incoming handshaking signal controlled by the DCE
7	Signal Ground Common reference voltage
8	Carrier Detect (CD) Incoming signal from a modem
20	Data Terminal Ready (DTR) Outgoing handshaking signal controlled by the DTE
22	Ring Indicator (RI) Incoming signal from a modem

Figure 3.1: 25-pin Connector On A DTE Device (PC Connection) [5]

Pin Number	Direction of signal:
1	Carrier Detect (CD) (from DCE) Incoming signal from a modem
2	Received Data (RD) Incoming Data from a DCE
3	Transmitted Data (TD) Outgoing Data to a DCE
4	Data Terminal Ready (DTR) Outgoing handshaking signal
5	Signal Ground Common reference voltage
6	Data Set Ready (DSR) Incoming handshaking signal
7	Request To Send (RTS) Outgoing flow control signal
8	Clear To Send (CTS) Incoming flow control signal
9	Ring Indicator (RI) (from DCE) Incoming signal from a modem

Figure 3.2: 9-pin Connector On A DTE Device (PC Connection) [5]

The RTS (Ready to Send) line and the CTS (Clear to Send) line are used when "hardware flow control" is enabled in both the DTE and DCE devices. The DTE device puts this line in a mark condition to tell the remote device that it is ready to receive data. If the DTE device is not able to receive data, possibly because of it's buffer being full, it will put this line in the space condition as a signal to the DCE to stop sending data. When the DTE device is ready to receive more data, after data has been removed from its receive buffer, it will place this line back in the mark condition. The complement of the RTS wire is CTS. The DCE device puts the CTS line in a mark condition to tell the DTE device that it is ready to receive data. Likewise, if the DCE device is unable to receive data, it will place this line in the space condition. Together, these two lines make up what is called RTS/CTS or *hardware flow control*. The RS-232 protocol offers this type of flow control, as well as Xon/Xoff or *software flow control*. Software flow control uses special control characters, which are transmitted from one device to another to tell the device to stop or start sending data. With software flow control the RTS and CTS lines are not used.

The intended function of the DTR (Data Transfer Ready) line is very similar to the RTS line. DSR (Data Set Ready) is the companion to DTR in the same way that CTS is to RTS. Some serial devices use DTR and DSR as signals to simply confirm that a device is connected and turned on. DTR is set to the mark state when the serial port is opened and is left in the mark state until the port is closed. The DTR and DSR lines were originally designed to provide an alternate method of hardware handshaking. It is pointless to use both RTS/CTS

and DTR/DSR for flow control signals at the same time. Therefore, DTR and DSR are rarely used for flow control.

Carrier Detect (CD) is used by a modem to signal that it has a made a connection with another modem or has detected a carrier tone. The Ring Indicator (RI) line is toggled by a modem when an incoming call rings the receivers phone. The Carrier Detect (CD) and the Ring Indicator (RI) lines are only available with connections to a modem since modems transmit status information to a PC when either a carrier signal is detected or when the line is ringing.

Signal voltage levels and the data format for the RS-232 protocol are presented in Appendix A.

3.2 GPIB IEEE-488 Bus

The GPIB IEEE-488 bus was developed to connect and control programmable instruments and to provide a standard parallel interface for communication between instruments from different sources. This versatile interface was originally developed by Hewlett Packard (HP) and was referred to as the HP Interface Bus (HPIB). The IEEE renamed it as GPIB. The GPIB uses standard TTL negative logic.

A generalized block diagram of the GPIB IEEE-488 bus is presented in Figure 3.3. At power-up, the GPIB IEEE-488 interface that is programmed to be the *System Controller* becomes the *Active Controller* in charge. The System Controller may optionally Pass Control to another controller, which would then become the Active Controller.

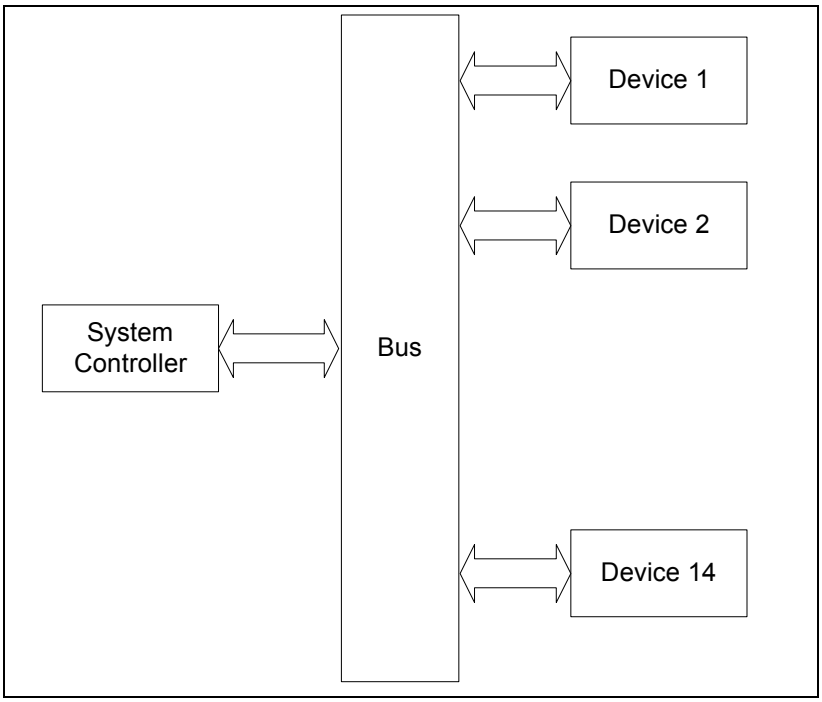

Figure 3.3: Block Diagram Of The GPIB IEEE-488 Bus

There are three types of devices that can be connected to the GPIB IEEE-488 bus. These devices are termed Listeners, Talkers or Controllers depending on their function. Some devices include more than one of these functions. The standard allows a maximum of thirtyone devices to be connected to the same bus.

It is possible to have several Controllers on the bus but only one may be active at any given time. The Active Controller may pass control to another controller, which in turn can pass it back or on to another controller. A Listener is a device that can receive data from the bus when instructed by the controller and a Talker transmits data on to the bus when instructed. The Controller can set up a talker and a group of listeners so that it is possible to send data to groups of devices.

The GPIB IEEE-488 interface system consists of 16 signal lines and 8 ground lines. The 16 signal lines are divided into 3 groups.

3.2.1 Data Lines

The lines DIO1 through DIO8 are used to transfer addresses, control information and data. The IEEE 488 standard defines the formats for addresses and control bytes. Data formats are undefined and may be ASCII (with or without parity) or binary. DIO1 is the Least Significant Bit, which will correspond to bit 0 on most computers.

3.2.2 Handshake Lines

The handshaking process is outlined as follows. When the Controller or a Talker wishes to transmit data on the bus, it sets the DAV (Data Not Valid) line high and checks to see that the Not Ready For Data (NRFD) and Not Data Accepted (NDAC) lines are both low. If the check is successful it puts the data on the data lines. When all the devices that can receive the data are ready, each releases its NRFD line. When the last receiver releases NRFD the Controller or Talker takes DAV low, which indicates that valid data is on the bus. In response each receiver takes NRFD low again to indicate it is busy and releases NDAC when it has received the data. When the last receiver has accepted the data, NDAC will go high and the Controller or Talker can set DAV high again to transmit the next byte of data. If after setting the DAV line high the Controller or Talker senses that both NRFD and NDAC are high, then an error will occur. Additionally, if any device fails to perform its part of the handshake and releases either NDAC or NRFD, data cannot be transmitted over the bus. Eventually a

time-out error will be generated. The speed of the data transfer is controlled by the response of the slowest device on the bus. Therefore, it is difficult to estimate data transfer rates on the GPIB IEEE-488 bus since they are always device dependent.

3.2.3 Interface Management Lines

The five interface management lines (ATN, EOI, IFC, REN, SRQ) manage the flow of control and data bytes across the interface. The Attention (ATN) signal is asserted by the Controller to indicate that it is placing an address or control byte on the data bus. ATN is released to allow the assigned Talker to place status or data on the data bus. The Controller regains control by reasserting ATN. This process is normally performed synchronously with the handshake to avoid confusion between control and data bytes. The End Or Identify (EOI) signal has two uses. A Talker may assert EOI simultaneously with the last byte of data to indicate end-of-data. The Controller may assert EOI along with ATN to initiate a parallel poll. Although many devices do not use parallel poll, all devices should use EOI to end transfers. The Interface Clear (IFC) signal is asserted only by the System Controller in order to initialize all device interfaces to a known state. After releasing IFC, the System Controller is the Active Controller. Only the System Controller asserts the Remote Enable (REN) line. Its assertion does not place devices into remote control mode. Assertion of REN simply enables a device to go into remote mode when addressed to listen. When in remote mode a device should ignore its local front panel controls. The Service Request (SRQ) line functions like an interrupt. It

may be asserted by any device to request the Controller to perform some action. The Controller must determine which device is asserting SRQ by conducting a serial poll. The requesting device releases SRQ when it is polled.

The GPIB IEEE-488 standard allows up to thirty-one devices to be interconnected on one bus. Each device is assigned a unique primary address, which ranges from 0-31 by setting the address switches on the device. A secondary address may also be specified, which also ranges from 0-31. The GPIB IEEE-488 standard greatly simplifies the interconnection of programmable instruments by clearly defining mechanical, hardware and electrical protocol specifications.

The cables used to connect devices to the GPIB IEEE488 Bus are pictured in Figure 3.4.

Figure 3.4: The GPIB IEEE-488 Connectors

The connector pin diagram for a GPIB IEEE 488 Bus connector is presented in Figure 3.5.

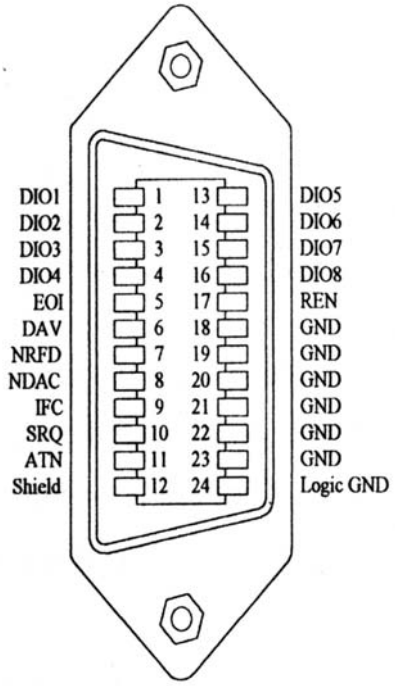

Figure 3.5: GPIB IEEE-488 Connector Pin Diagram

4 System Integration

System Integration for this project involved the task of integrating all the hardware and instruments together and bringing them under the control of a single software program. The following is an enumeration of tasks involved with the project, for which the use of hardware became necessary.

- Controlling the position of the sensor above the pad
- Controlling the pad position relative to the sensor
- Generating an excitation signal for the pad
- Measuring the response produced by the pad
- Controlling and tracking the humidity and temperature levels during the experiment

The entire system was comprised of a total of seven instruments and some additional components. The system required integration at two different levels; physical and logical. Physically the instruments were connected on two different buses, which depended on the type of communication interface they required. The two bus systems used were the RS-232 Serial interface and the GPIB IEEE-488 interface. On top of this physical layer was the logical layer that provided interaction control between the instruments. The various objects and classes created through the use of Microsoft Visual C++ implemented the logical interaction of the instruments.

A model of the setup used to carry out experiments on the pads is presented in Figure 4.1. In addition to the pad shape depicted in the Figure 4.1, pads can be semi-circular as well as circular without the gap in the center.

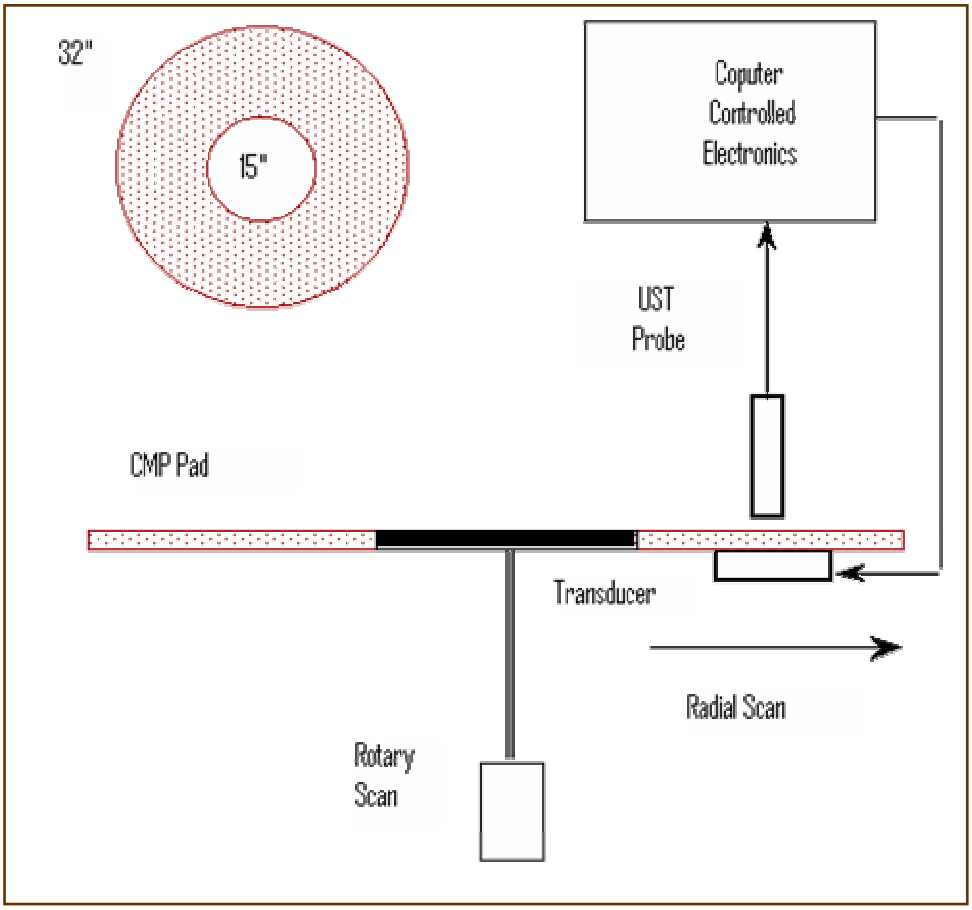

Figure 4.1: CMP Pad Test Setup

The assembly shown was mounted on a finely polished steel table. The supports for the table could be adjusted so that the pad was always perfectly horizontal.

Temperature and the humidity control was required in the vicinity of the pads in order to study the effects of these physical factors on the pads. Therefore, the test setup depicted in Figure 4.1 was enclosed inside a wooden insulating chamber.

4.1 Instruments Used

This sub-section provides information about the hardware used in the system. Table 4.1 provides basic information on all devices utilized during the testing.

Table 4.1: Instruments Integrated Into The System

Instrument	Manufacturer	Function	Interface
VP 9000 Motor Controller	Velmex, Inc.	Control of motors used to move sensor	Serial RS-232C
PS2520G Programmable Power Supply	Tektronix	Provide power to hold pad while measurement is being carried out	GPIB IEEE – 488
Lock-In Amplifier 7260	EG&G (AMETEK)	Generate excitation signal. Measure response	GPIB IEEE – 488
Data Acquisition Card 6035E	National Instruments	Control of switches used for tracking humidity	PCI
Temperature Controller and Sensor	Newport Instruments	Temperature control and measurement	Serial RS-232C
SERVO – 260 Studio Amplifier	Samson	Amplify Oscillator Output	-
Humidity Sensor	Honeywell	Measure humidity	Analog I/O

4.1.1 VP-9000 Motor Controller

The VP 9000 is a programmable stepper motor controller, which is capable of running up to four motors alternately. The controller uses a powerful microprocessor, support circuitry and has 64 Kilobytes of nonvolatile Random Access Memory (RAM) for storing setup parameters and programs. Commands and data can either be entered using the RS-232 Serial Interface or by using the front panel menu. An alphanumeric display displays the motor positions and

setup parameters. A host computer can send commands to the controller through its RS-232 serial interface.

Commands can specify motion in absolute as well as relative indices. An absolute index is a move relative to the absolute zero position. A relative index is a move in a certain direction and for a certain distance from the present position. The instrument manual provides tables to estimate the number of steps required to cover a given distance. Of the four available lines on the VP 9000 only three were used. The instrument controlled the position of the sensor relative to the center of the pad (X-axis), the height of the sensor above the pad surface (Z-axis) and the angle of the pad with respect to the sensor position (Radial axis). Detailed information about how this control was implemented using the class *CMotion* is provided in later sections [6]. The VP 9000 VELMEX Motor Controller is pictured in Figure 4.2.

Figure 4.2: VP 9000 VELMEX Motor Controller

4.1.2 PS2520G Power Supply

The PS2520G is a Programmable Power Supply from Tektronix. It offers three power outputs. Two of these supply voltage from 0 to 36 V and current from 0 to 1.5 A. The third one has a higher current capability of 0 to 3 A and

supplies voltages from 0 to 6 V. The instrument has a Light Emitting Diode (LED) display to indicate the voltage and current levels.

The PS2520G has a GPIB IEEE-488 interface that enables a host computer to control it through the use of Standard Commands for Programmable Instruments (SCPI). This device was used to provide a 12 V output to open a Vacuum Valve. This enabled the vacuum pump to create enough suction for the pad under test to be grabbed and held in place while a measurement was being performed on it [7]. The PS2520G Programmable Power Supply is pictured in Figure 4.3.

Figure 4.3: PS2520G Programmable Power Supply

4.1.3 Lock-In Amplifier 7260

A lock-in amplifier can be used for two basic purposes. To recover a signal in the presence of overwhelming background noise or to provide high resolution measurements of relatively clean signals over several orders of

magnitude and frequency. Modern instruments like the 7260 offer many additional features. These instruments are used in varied fields of research such as Optics, Electrical Engineering, Fundamental Physics and Material Sciences. The Lock-In 7260 provides the following functions:

- Precision Oscillator
- Vector Voltmeter
- Phase Meter
- AC Signal Recovery
- Frequency Meter
- Transient Recorder
- Spectrum Analyzer
- Noise Meter

The Lock-In amplifier was central to the proper functioning of the system. This project used its Oscillator and Voltmeter sections. The Oscillator generated a precise 26 KHz signal with amplitude of 0.5 V (peak to peak) that was amplified and used to excite the pad under test. The response from the pad was detected by an ultrasonic transducer probe and applied to a channel on the Lock-In amplifier. The amplifier was set to display and transmit this reading to the host computer, see Figure 4.3.

This Lock-In amplifier has a GPIB IEEE-488 interface for communication purposes. There are specific commands to control the instrument remotely. The *CLockin* class in the software was used to communicate with this device. [8]

Figure 4.4: DSP Lock-In Amplifier 7260 [9]

4.1.4 Data Acquisition Card (DAQ)

The National Instruments 6035E Data Acquisition Card features sixteen channels (eight differential) of 16-bit analog input, two channels of 12-bit analog output, a 68-pin connector and eight lines of digital I/O.

This card was primarily used by the system to implement humidity control. A humidity sensor generated a signal, which is mathematically related to the actual humidity. The signal was read by the analog input of the DAQ. The digitized signal value was sent to the program, which mathematically converted it to the actual humidity value. Based on whether the humidity level was to be increased or decreased, the program instructed the DAQ to output an electrical voltage on its corresponding analog outputs. The DAQ output voltage activated relays in order to enable the required system. [10]

The DAQ functioned in conjunction with the National Instruments SC-2050 I/0 Board and the SC-2062 Relay Board. National Instruments provides driver software for the DAQ. Library routines from the DAQ software were used in the VC++ program to implement humidity control. [11]

4.1.5 Temperature Controller And Sensor

The iSeries CNi16D temperature controller fully controlled the temperature during the experiments. The temperature controller provides an RS-232 interface, which enabled remote control by the use of commands specified in its configuration manual.

The instrument has analog inputs and outputs. A thermocouple is connected to the analog input to sense the temperature. The analog output operates a relay that supplies power to the heater. Depending upon the current temperature, once the setpoint is specified, the instrument enables/disables the relay. A temperature dead-band can be specified in which the controller maintains the state of the relay.

The optional analog output can be programmed within a range of 0-10 Vdc or 0-20 mA. It is selectable as either a control output or as a calibrated retransmission of the process value, which is a unique feature among controllers. The type of control is also selectable. On/Off, Proportional-Integral (PI), Proportional-Derivative (PD) and Proportional-Integral-Derivative (PID) can be selected [12]. The iSeries CNi16D Temperature Controller is pictured in Figure 4.5.

Figure 4.5: The ISeries CNi16D Temperature Controller

4.1.6 Humidity Sensor

The HIH-3610 monolithic Integrated Circuit (IC) humidity sensor is designed specifically for high volume Original Equipment Manufacturer (OEM) users. Direct input to a controller or other device is made possible by this sensor's linear voltage output. With a typical current draw of only 200 μA, the HIH-3610 is ideally suited for low drain, battery powered systems. The sensor requires a supply voltage of 4.0 to 5.8 V and operates over a temperature range of -40° F to 185° F. The HIH-3610-001 Humidity Sensor is pictured in Figure 4.6.

Figure 4.6: HIH-3610-001 Humidity Sensor

The Relative Humidity (RH) is derived from the mathematical relation

RH = (((Voltage/5.0)-0.016)/0.0062),

here Voltage is the voltage value obtained from the output of the sensor. The voltage output of the sensor is picked up by the DAQ analog input, which is read by the program. [13]

4.2 Hardware Integration

Device manufacturers outfit their instruments with communication capabilities in order to enable operators to control the instruments remotely. Ethernet, USB, Serial and Parallel interfaces are examples of communication connection capabilities that are routinely provided. Many instruments that use the GPIB IEEE-488 interface have the capability to decode the SCPI instruction set. Others have very specific command formats, which are specified by their manufacturers. In order to control such devices from a host computer, the communication link has to be established. Once the communication link is established the software is simply required to send instructions to the instrument in the form of strings. Manufacturers provide libraries that include commands specifically designed to make the instrument perform specific tasks. These library files were linked into the project so that the commands offered could be used freely anywhere within the program.

4.2.1 GPIB IEEE-488 Bus

The Lock-In amplifier and the Tektronix Power Supply were two devices connected to this bus. Table 4.2 specifies the GPIB address that was used for each device on the bus.

Table 4.2: GPIB Device Addresses

Device	GPIB Address
Computer (GPIB Card) Controller	21
EG&G Lock-In amplifier 7260	12
Tektronix PS2520G Power Supply	6

The devices on a GPIB bus could be arranged in two configurations; linear and star. The linear configuration configured the devices in a daisy chain. The connector from the controller is attached to one device. The second device is connected to the first and the chain continues. In a star configuration all the devices are connected to the controller using dual-ended stackable connectors. The two configurations are depicted in Figure 4.6. This project used the star configuration. The Controller card was directly connected to the Lock-In amplifier and the Power Supply.

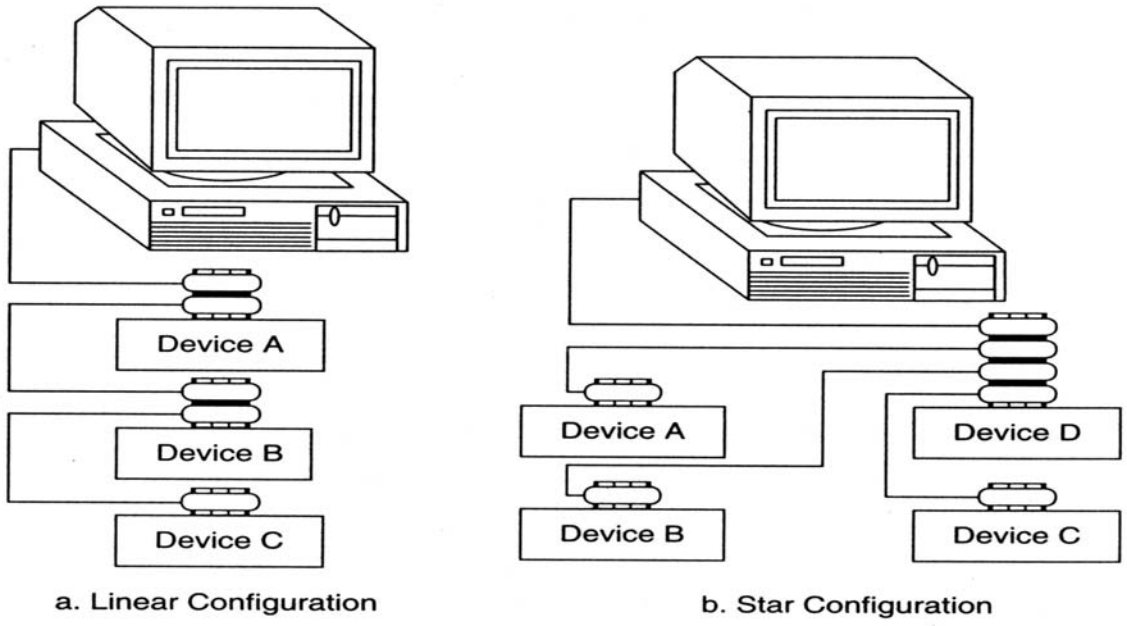

Figure 4.7: GPIB Bus Configurations

4.2.2 RS-232 Serial Bus

Unlike the GPIB, the RS-232 interface does not specify a designated controller. Communication between devices consists of passing serial data with start and stop bits.

The Temperature Controller and the VP 9000 Motor Controller used the RS-232 protocol. These devices were connected to ports COM1 and COM5 respectively. In order to provide additional COM ports the National Instruments PCI-232/8 port extender was used. This PCI card and its cables provide eight serial ports that can be used simultaneously.

The VP 9000 was connected to the serial port on the host using a 9-pin RS-232 connector. A simple three-wire connection (Transmit, Receive and Ground) was used between the Temperature Controller and the host. Newport provides software, which allows the Temperature Controller to be configured from the computer through the RS-232 bus.

4.3 Software Integration

Software Integration achieved the logical interconnection of the instruments used in the project. A single program was required to control all the devices and processes involved. A modular, object oriented approach, was used to develop the software that made full use of the advantages provided by Visual C++. Visual C++ classes uniquely represented every device involved in the system.

The software was designed through the use of modules, which constituted variables and methods that defined the behavior of individual instruments. Each

of the modules could be invoked and controlled by the main program. The program presents a user-friendly interface to the operator. The level of automation provided was such that it was possible to conduct experiments over a range of values of the process parameters without the need for user intervention.

The following sections provide a detailed explanation of each interface in the program in the same order as they would be presented to the operator while running the application.

4.3.1 Login Dialog

The login window is a part of the *CLoginDlg* class. This class is derived from the *CDialog* class, which is a part of the Microsoft Foundation Classes (MFC). Therefore, it inherits all the functionality of the CDialog class. The user is required to enter a valid identification and password in this box in order to gain access to the system. It provides a simple way of protecting the application and its hardware from unauthorized use. The login window is presented in Figure 4.8.

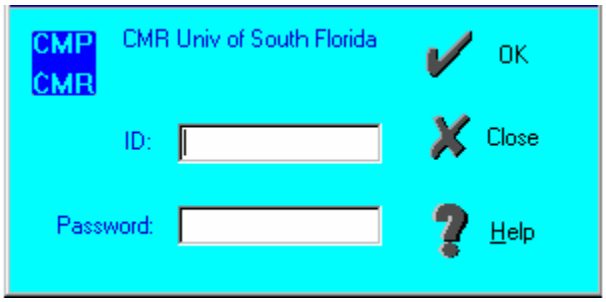

Figure 4.8: Login Window

4.3.2 Main

The main dialog box is a part of the *CMain* class, which is also derived from the Cdialog class. The Main window is presented after the user gains

access to the system from the login window. The CMain class is a fundamental component of the software. This class contains procedures to initialize all of the hardware in the system. Upon initialization of this class, communication between the host and the serial devices (CNi16D and VP 9000) is initiated by defining the required parameters. Main also controls access to the processes that the system offers. The control box is opened, for all hardware to be used, from the menu on the main window. The main window is presented in Figure 4.9.

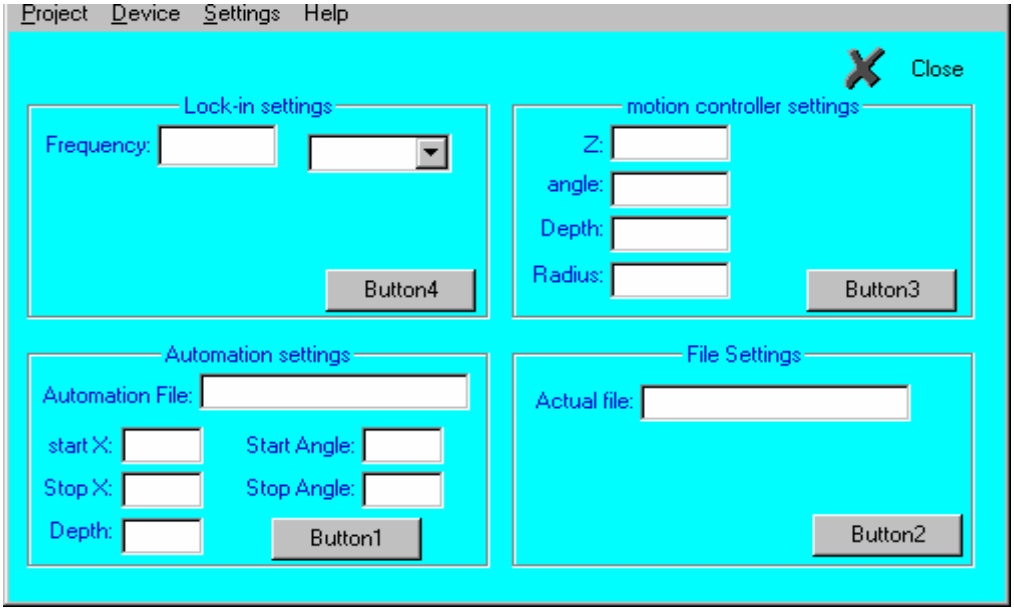

Figure 4.9: Main Window

The Main Dialog displays the settings of all instruments. It also prevents contention between the dialog boxes and mediates where control exists at every point in the program. If at any time one class has control over another class then a second object of that class is prevented from being created by the Main class. This action prevents interference between objects.

4.3.3 Motion

The Motion dialog box is a front end that was used to control the position of the three axes through the use of the VP 9000 motor controller. The *CMotion* class contains functions to direct, control and display the position of each axis. The position of the actuator on the axis is displayed in terms of distance. The VP 9000 keeps track of the position in terms of steps. The distance per step, for each axis, is presented in Table 4.3.

Table 4.3: Step Sizes For Axes Controlled By The VP 9000

Axis	Step size
Radial (X)	6.5 microns
Vertical (Z)	2.5 microns
Angular	0.5 degrees

When the Motion dialog box is opened from the main window the class is provided a link to the corresponding serial port from the main window. This link is used to enable communication with the host through the RS-232 protocol. Upon initialization, the motors controlling the X and Z-axes are reset to the zero position.

This window offers the operator multiple options for moving the axes. The speed of motion of the axes can be controlled. It is also possible to make coarse and fine adjustments to the axes position by using the *big* and *little* boxes. The software allows both coarse and fine adjustment of the axis position. The Motion window with these controls is presented in Figure 4.10.

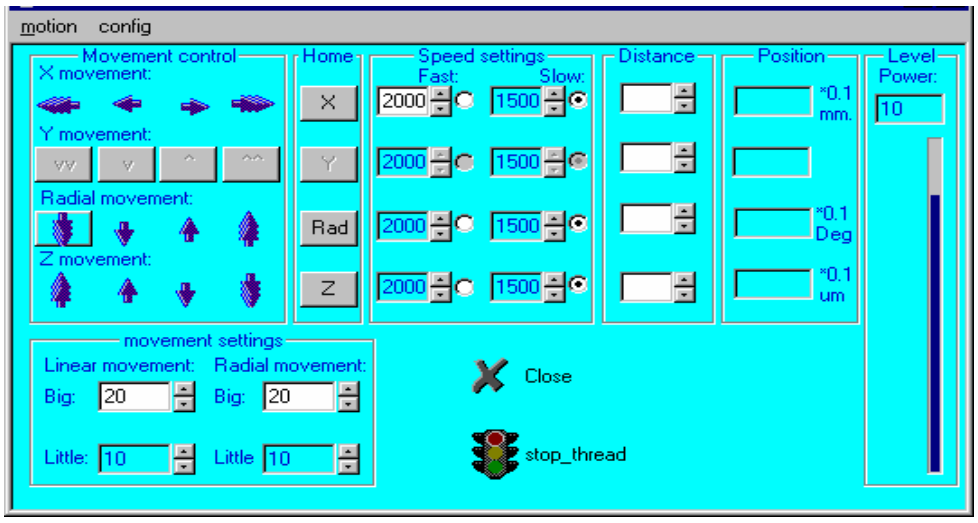

Figure 4.10: Motion Window

4.3.4 Lock-In

The dialog box for the *CLockin* class enables the operator to adjust the frequency and amplitude of the signal generated by the oscillator section of the Lock-In amplifier. This window is opened using the main window's menu. The Lock-In window is presented in Figure 4.11.

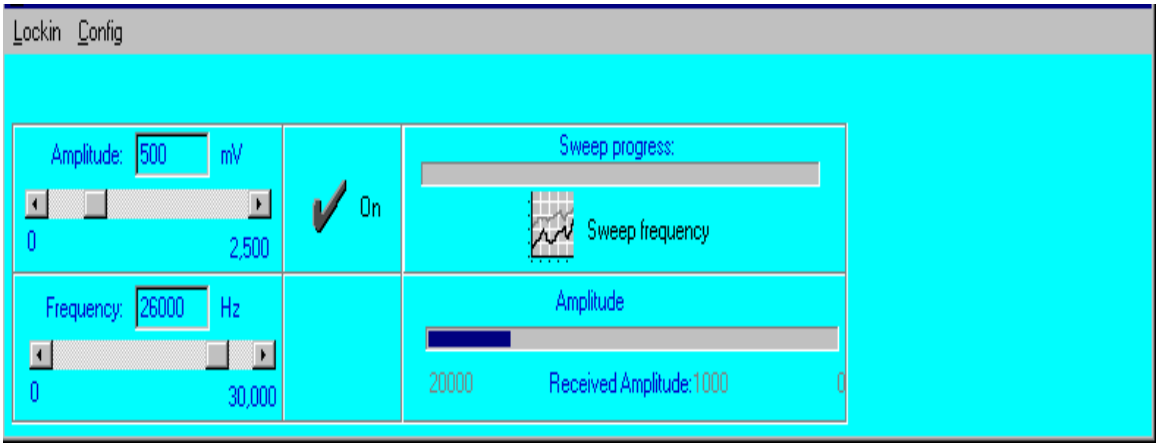

Figure 4.11: Lock-In Window

At initialization the amplitude and frequency, of the signal generated, are set to 0.5 V (Root Mean Square) and 26KHz respectively. The device is set in the voltage mode and input A is monitored.

4.3.5 Automation

The Automation Dialog box is called from the Main window. This is one of the principal windows in the software that controls the different types of experiments that could be carried out. The parameters for a set of experiments can be specified in the window itself or can be read from a file. Upon initialization all the devices in the system are initialized and their windows are created but kept hidden. The *CAutomation* class uses objects of all the device classes to control the run. The resulting readings are stored in a file. This class can also spawn a window in which a three dimensional graph of the readings is plotted. The Automation window is presented in Figure 4.12.

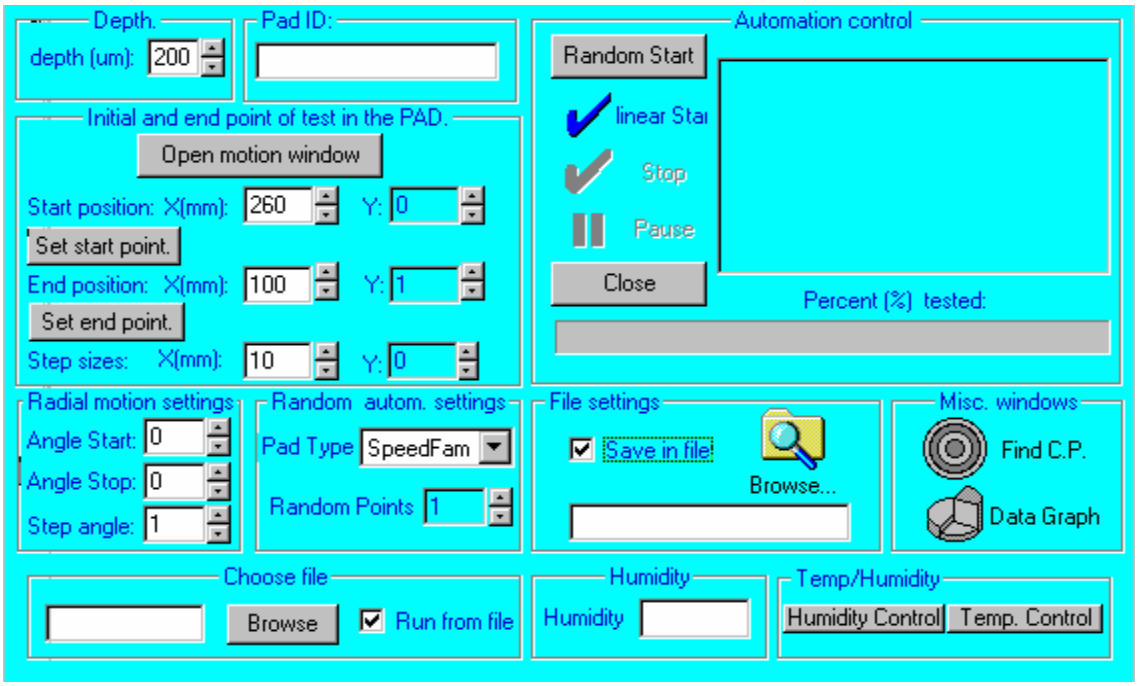

Figure 4.12: The Automation Window

This class includes various VC++ *threads* for carrying out different kinds of experiments. Threads are also spawned to control physical factors such as temperature and humidity. The humidity value is updated periodically during

every run. The temperature can be read from the CNi16D instrument.

4.3.6 Automation Graph

The *CAutomationGraph* class includes functions that are used to plot data in three dimensions. It incorporates a special tool, the 3-D Graph Object, which was developed by National Instruments.

This tool provides the capability to present and spatially format the data gathered using the ultrasound signal on the PAD. The tool provides an enhanced type of OCX (object control). An OCX control exchanges data between two applications or between an application and an operating system. It incorporates a concept called Component Object Model (COM), which is a specification that defines a standard binary interface between objects.

COM defines the interface modules and standard structures to pass data and also some additional function calls used by the application. The use of COM allows bypassing the language barriers between applications. Objects based in COM can be placed inside any application in any graphical language and will work in the same way in all of them.

The 3-D graph can present multiple plots simultaneously while allowing the characteristics of each one to be changed separately. This tool uses special structures that provide information about the position of each point in each of the graphs.

A 3-D graph is used off-line to present data already collected and saved. It is also used on-line in real time while the data is read from the Lock-In.The latter uses more resources from the system, which slows the process slightly.

The use of a faster computer or the use of more time between updates provides a solution to this problem.

The block in which this object is embedded also complies with the premise of reusability. This means that any other system can use this block separately, with no changes to the code, and maintain the same capabilities and interfaces. Using three dimensions for the presentation of data enables the user to obtain more information from different angles. The transfer of data to a graphic object is through special types of variables called variants, which allow the same data to have different data types. The 3-D Graph window is presented in Figure 4.13.

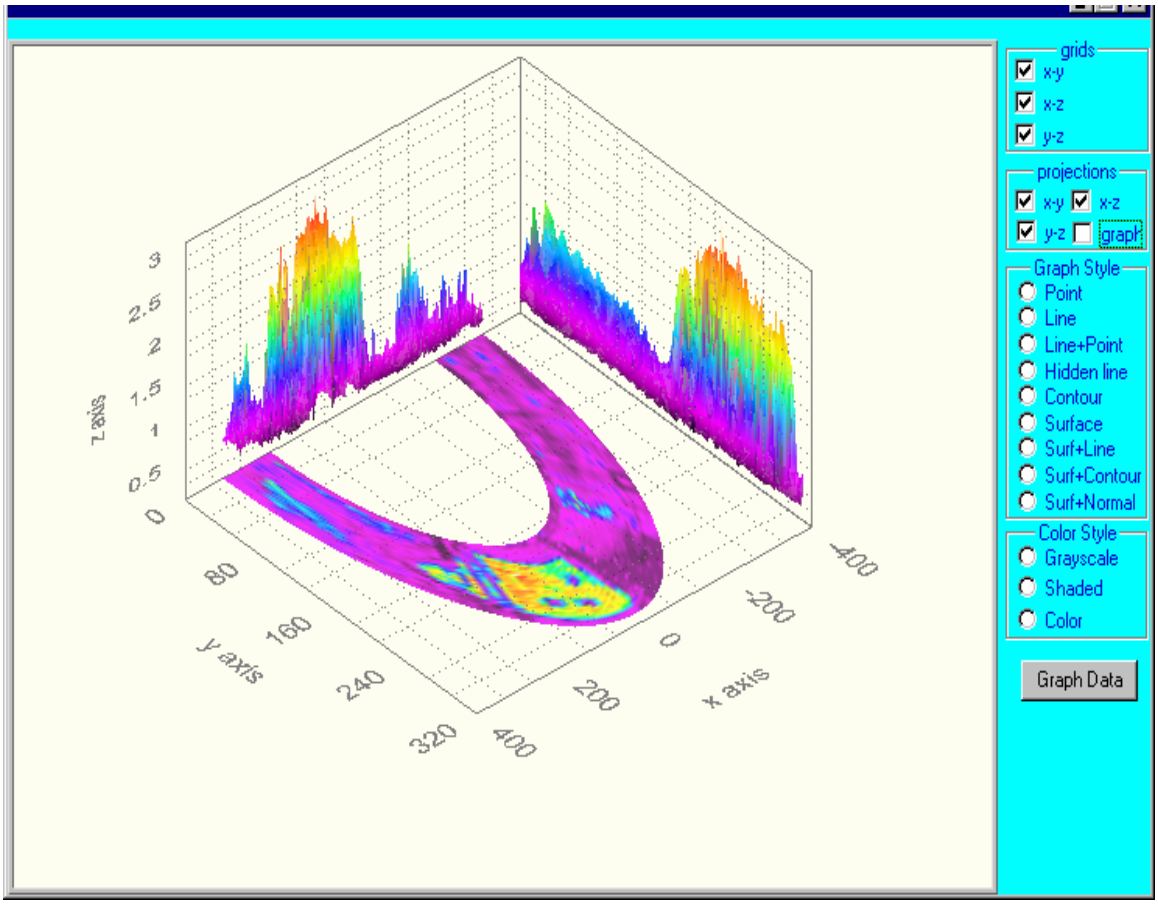

Figure 4.13: 3-D Graph Window With Axes Projections

4.3.7 Statistical Analysis Tool

Measurements obtained from the pads yield results in terms of millivolts. The readings from each run are stored in separate files. The Statistical Analysis Tool uses these data files to check the uniformity of the pad surface. Voltage readings obtained at each point depend on the thickness of the pad at that point. If the readings are approximately the same in value the pad has a uniformly thick surface with very few perturbations.

This program uses the F-test to analyze the pad's uniformity. In statistics, an F-test is usually used to test for the equality in the standard deviations of two populations.

4.3.7.1 Testing Algorithm

The F-value for the test is calculated by taking the ratio of the Mean Square Error (MSE) of each line and the total MSE. This value is compared with the value obtained form the F-table. If the calculated value is larger than the one obtained from the table the pad surface is non-uniform.

In statistical terminology the Degree of Freedom (DOF) is a measure of variability that expresses the number of options available within a variable or space. In a sy0stem with N states the degree of freedom is N. In this case, if "a" is the number of lines and "n" is the number of points on each line on which measurements are made and

$$N=n*a$$

then the $DOF_{numerator}$ is (a-1) and $DOF_{denominator}$ is (N-a).

This tool is a dialog box in which the file containing the readings corresponding to the required experiment is selected. The program reads the measured values from the file, performs the F-test and displays the results. The calculated and observed F-values are displayed along with a comment about the pad's uniformity. The flowchart in Figure 4.14 depicts the procedure used.

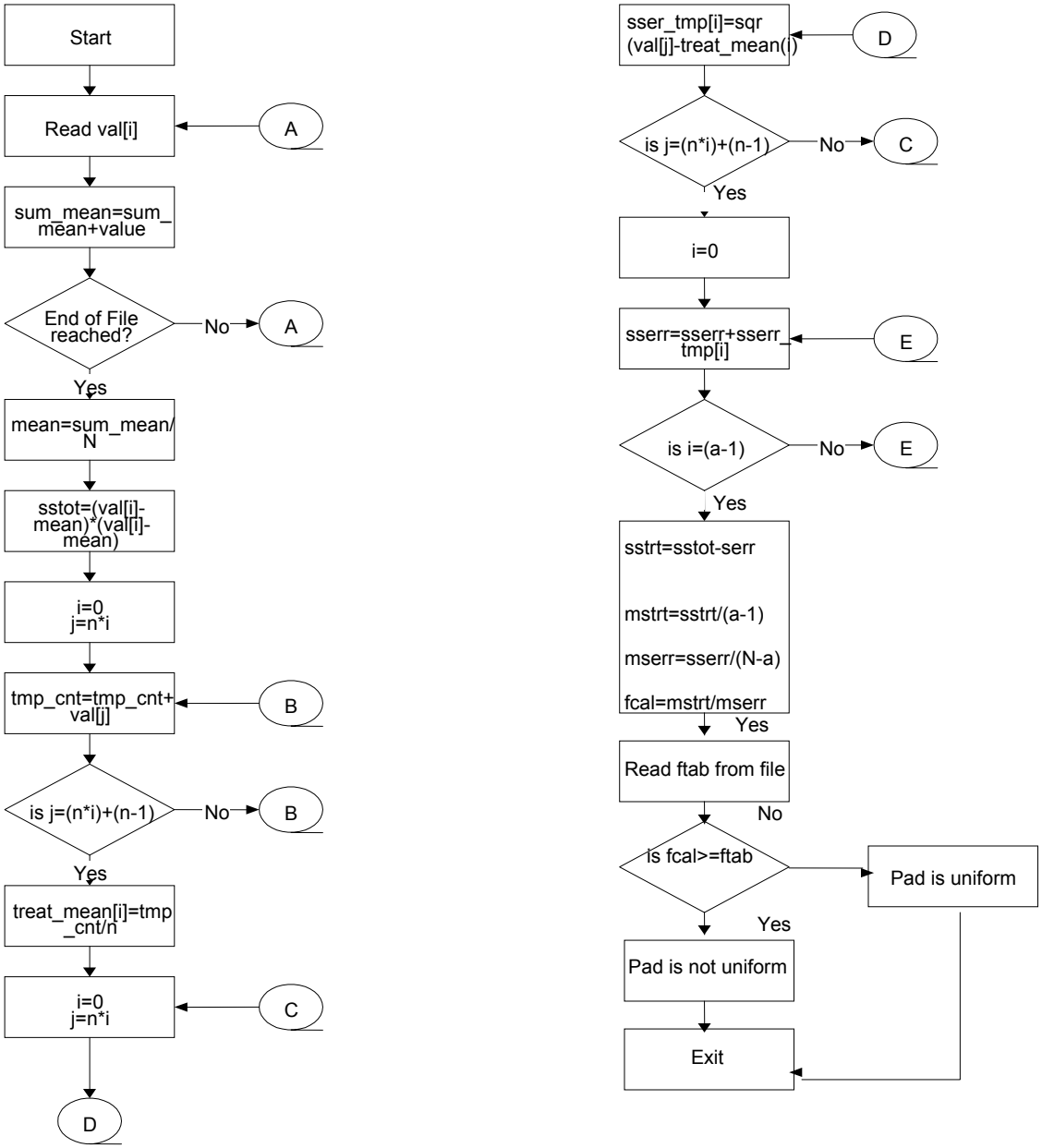

Figure 4.14: Flowchart For Pad Uniformity Testing

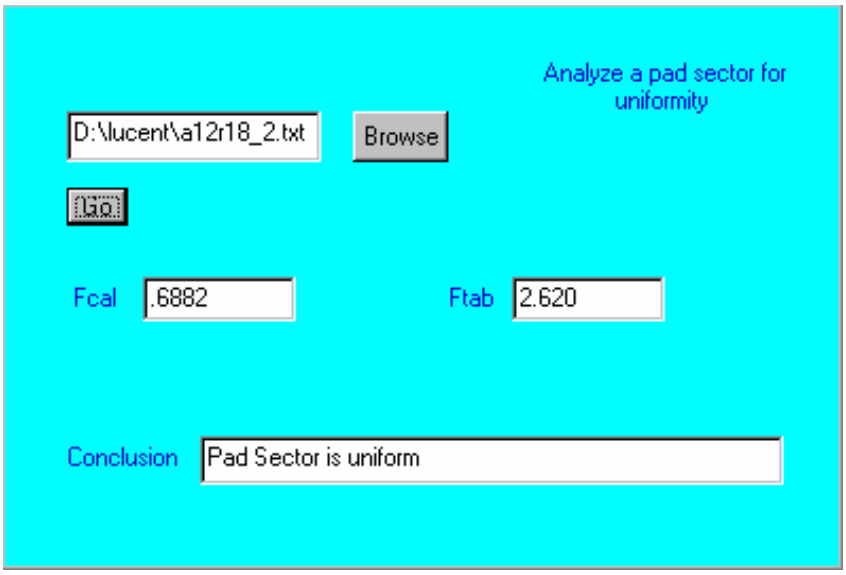

Figure 4.15: The Statistical Analysis Tool

4.3.8 Temperature Control

The *Tmpcntrl* class maintains the variables and methods for setting and maintaining the temperature at a desired value during a run. Upon initialization, communication is established with the CNi16D via the RS-232 protocol. Independent functions cause the transfer of commands to and data from the instrument.

During a run the control rests with the *CAutomation* class. This class communicates with all the instruments by making use of objects of the respective classes. If any command needs to be sent to an instrument the corresponding function is called. Temperature control, by its nature, is a process that runs continuously during the entire course of the experiment. This requires the control to remain with the class that implements temperature control. However, this is not possible. Therefore, the CAutomation class spawns a VC++ *thread* for this purpose. Threads run parallel to the main program and are controlled by the

process that initiates them. Threads are widely used in this application.

Temperature control is accomplished by comparing the current temperature with the required temperature in order to develop the proper correction. Based on the result of this comparison the CNi16D is instructed to either activate or deactivate the relay that supplies power to the heater element. Control is implemented only for raising the temperature and maintaining it at a desired value. There is no capability for cooling. There is a dead-band of 2° F before the relay is reactivated when the environment starts to cool. The temperature controller uses the On/Off mode. PI, PD or PID mode can also be used if required to control temperature.

4.3.9 Humidity Control

Humidity control is similar to temperature control. It needs to be implemented in such a way that it runs in the background, as a process, throughout the duration of the experiment. Unlike temperature, the humidity can be controlled in both directions. The *CHumcontrol* class provides control for humidification as well as dehumidification.

Upon initialization of this class the voltage from the HIH-3610 humidity sensor is read by the DAQ and the corresponding humidity value is displayed on the screen. Humidity Control is implemented in the form of a thread spawned from the CAutomation class. The thread compares the current humidity with a desired value and activates the required apparatus accordingly. The Humidity Controller has a dead-band of 3 percentage points within which the setup maintains its state. The environment control setup is presented in Figure 4.16.

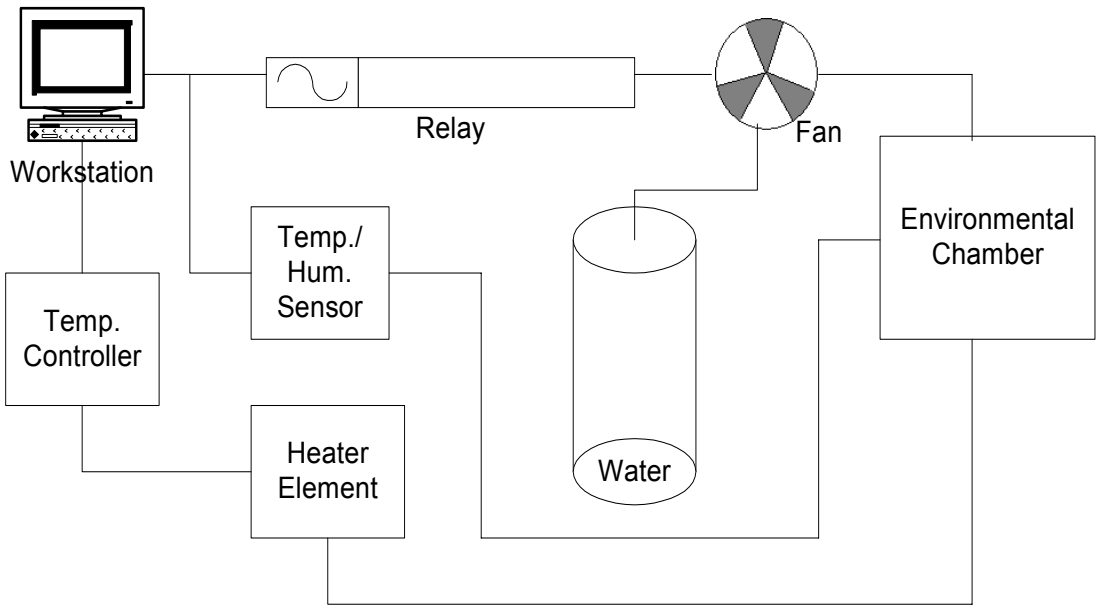

Figure 4.16: Setup For Temperature And Humidity Control

4.3.10 Class Interaction

The interaction between all the aforementioned classes can be illustrated using a class interaction diagram. Each block in the diagram represents a class in the software. The dialog boxes of the classes under Main are opened using the menu on the Main dialog box. The Automation dialog box can be used to open the temperature, humidity and the statistical tool windows. The 3-D graph object can be updated during a run to reflect the latest data. The Class Interaction Diagram is presented in Figure 4.17. The process steps used to collect data will be addressed in Chapter 5.

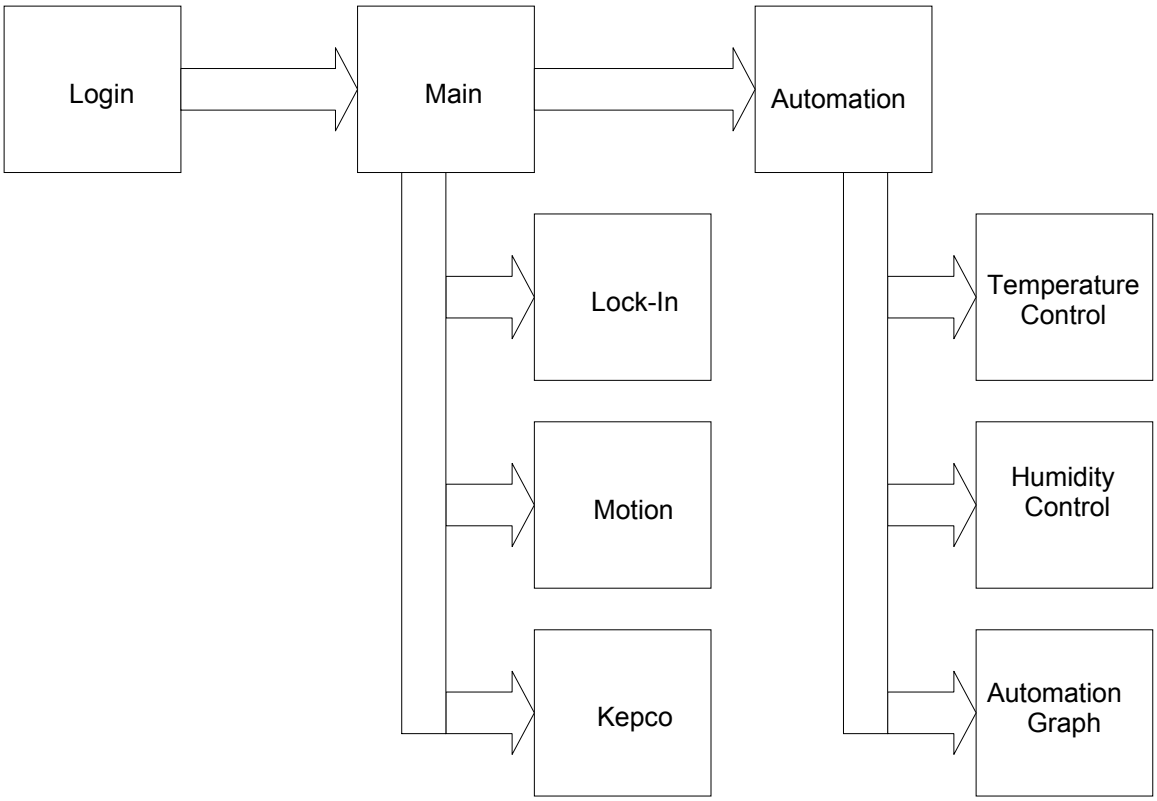

Figure 4.17: Class Interaction Diagram

5 System Processes

The system can scan the pads under test in a number of different ways in order to carry out an investigation of their properties. A regular scan with constant angular and radial increments is most often used. A constant-radius scan measures pad response at all points at a fixed distance from the center. These two scans produce the maximum correlation between data taken on the same test sample at different times due to the low uncertainty in the point position. A random scan measures data without following a fixed pattern. This type of scan can be used to examine surface uniformity by comparing the results of two different tests over the same area. A spiral scan starts from the inside and moves outward in a spiral. A spiral scan covers the maximum surface area. The spiral scan requires the radial and angular motors to work simultaneously making the associated algorithm more intricate.

5.1 Testing Procedure

In order to begin testing on a new pad its size and geometry have to be known. The depth at which the readings are taken on a pad is a critical parameter. If the probe is pushed too much into the pad it can create a permanent stress at that point, which will alter the mechanical properties of the pad. Hence, it is necessary to know the exact position of the pad's surface relative to the position of the probe on the Z-axis.

5.1.1 Contact Point

The *Contact Point* is the distance of the pad's surface from the zero position on the Z-axis. The Contact Point of a pad depends on its thickness. Prior to beginning experiments, it is necessary to determine the value of the Contact Point in order to establish confidence in the depth parameter during a run. An iterative function is used to determine the value of the Contact Point. The procedure involves taking a set of readings and fitting them to a line. After the next reading is taken, its deviation from the line's equation is determined and a new line is plotted if the reading exceeds a pre-defined threshold. A snapshot of this function being executed is presented in Figure 5.1.

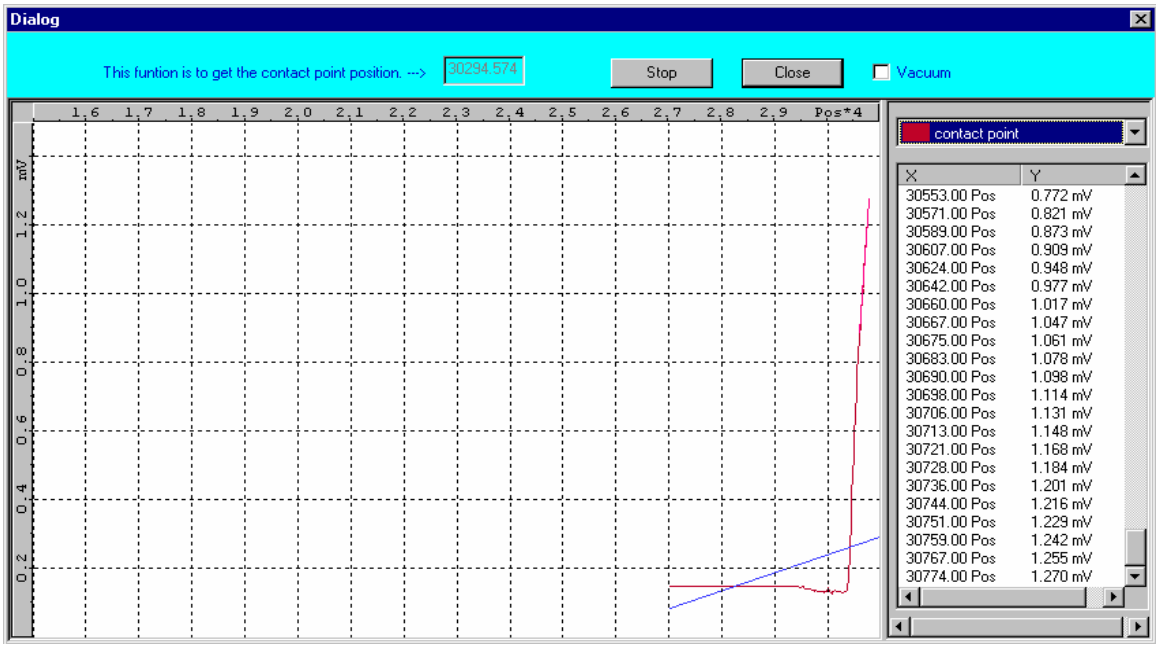

Figure 5.1: Find Contact Point Function

5.1.2 The Automation Thread

The software required numerous controlling and monitoring techniques in order to work in tandem. Therefore, the concept of threads was used

extensively. The automthread was used to make regular and repeatable scans on the pad as explained in section 5.1.1.

The thread was started after the radial and angular positional parameters were specified in the Automation window. The thread opened a file specified by the user in order to store the collected data. The X and Z-axes motors were initialized to their zero positions. The probe was moved to the specified starting point. After the measurement was completed the VP 9000 moved the probe over the radial and angular increments specified. Two parameters, *runn* and *pause*, were used to check user intervention. If these variables became zero or one the thread was stopped. The flowchart presented in Figure 5.2, page 4, illustrates the steps involved.

5.1.3 The New-Automation Thread

The *newautomthread* was developed after the automthread. In the automthread the parameters required for a run were specified in the automation window. This limited the number of runs to one. If a new set of readings was to be taken the parameter values had to be changed and the thread started again. This problem was overcome in the newautomthread. This thread read the variable values from a file and completed the run. The file was simply a text file containing the position, angular increment, radial increment, depth, temperature and humidity. Values for one run were specified without a line break. The flowchart presented in Figure 5.3, page 5, illustrates the flow for the newautomthread.

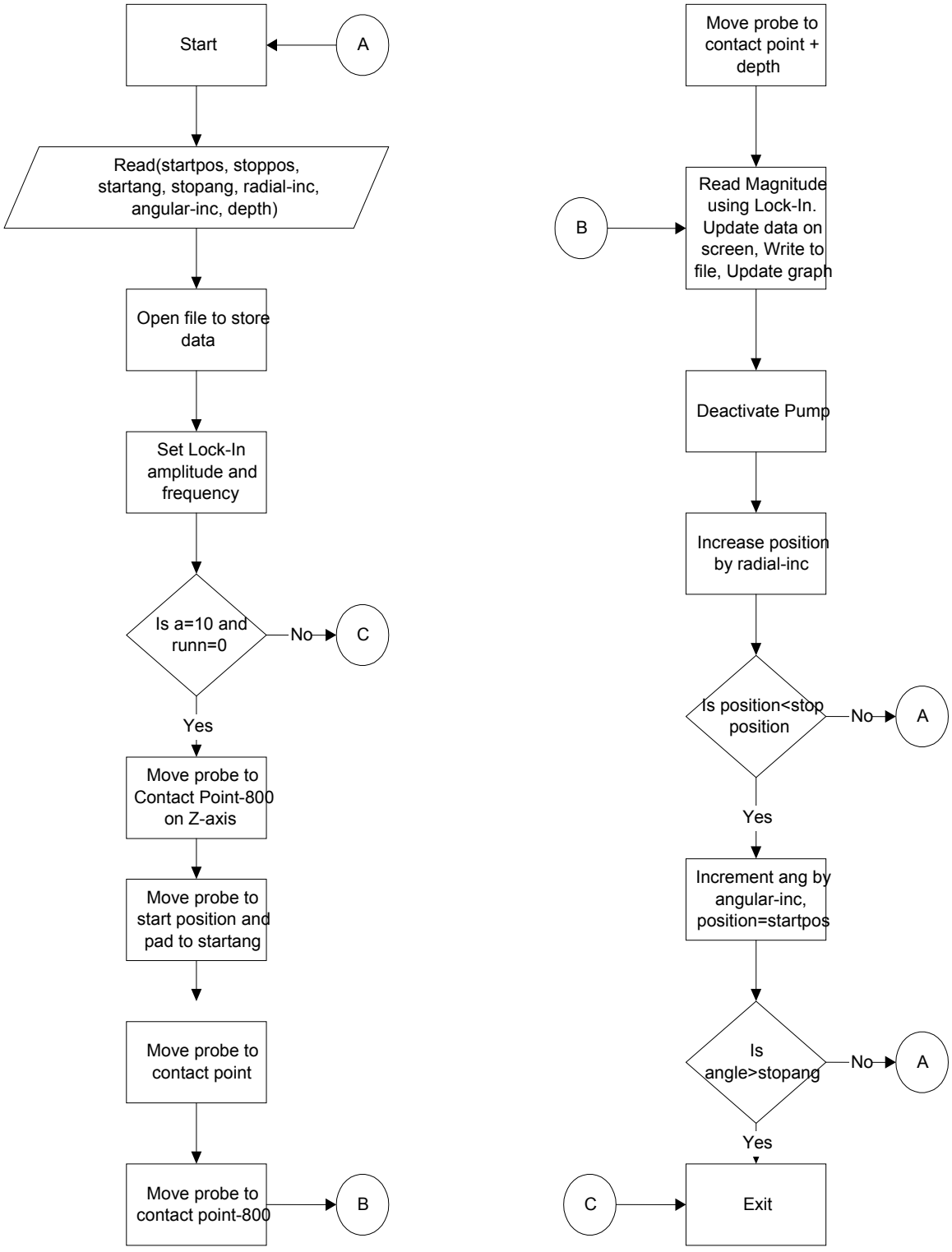

Figure 5.2: Flowchart For Automthread

Temperature and humidity control were implemented by the newautomthread. When the thread was started it spawned two processes to control these physical factors. As long as the temperature and humidity remained below a required level the thread was paused. The thread was only restarted when the processes notified the thread that the required levels of temperature and humidity were reached. The remainder of the thread is similar to automthread. When a run was completed the motors were re-initialized, the next set of parameters was read from the parameter file and the process continued.

No capability existed, in the system, to implement cooling. Therefore, the parameter file always specified increasing values of temperature.

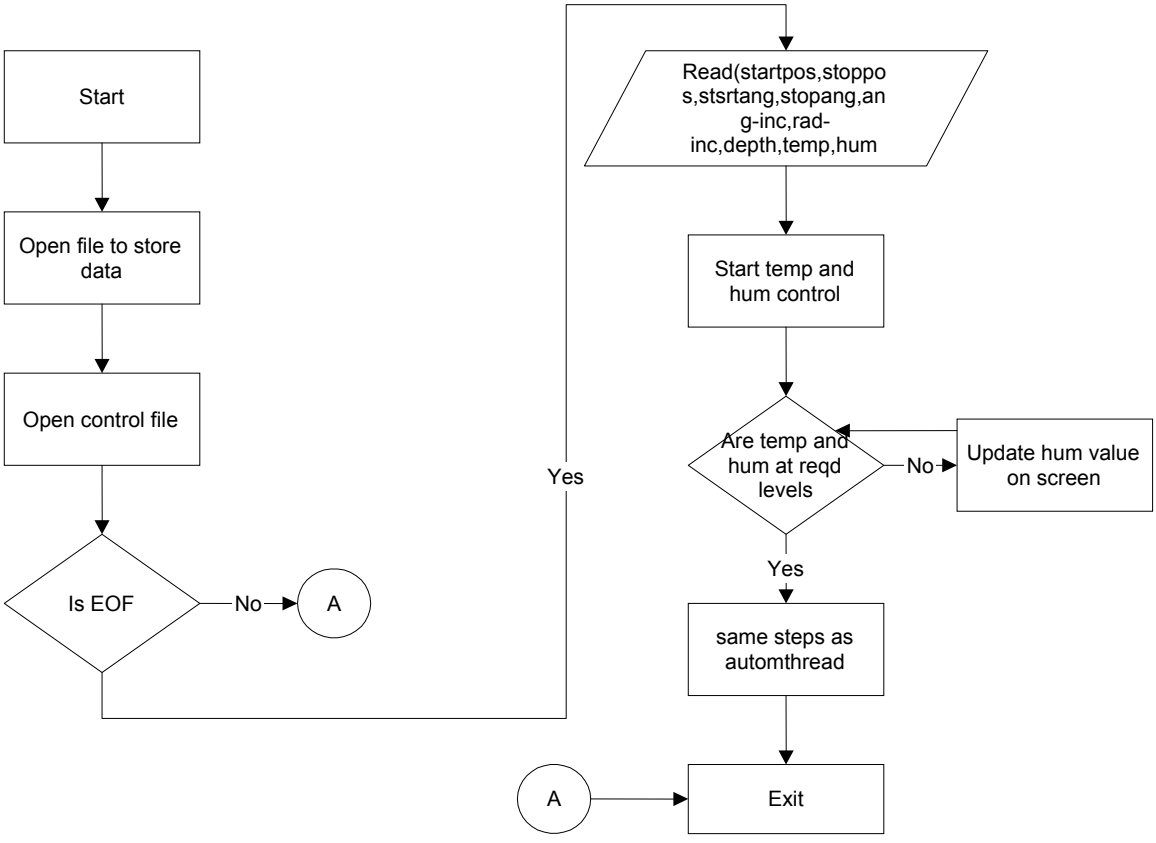

Figure 5.3: Flowchart For Newautomthread

5.1.4 The Temperature Control Thread

An object of the temperature control class was created when the Automation class was initialized. The CNi16D was queried and the current temperature was displayed in the dialog box. However, the dialog box was kept hidden by the system. There were two situations that required starting temperature control.

- If the temperature needed to be monitored for chamber characterization purposes the thread would be started by using a button on the temperature control dialog box.
- If monitoring was required during a run the automation thread could spawn a temperature control thread from within itself.

Activating the Temp Control button on the automation window spawned a thread called *tmpctlthread*. The only action performed by this thread was to make the dialog box visible to the operator. The operator could then specify the required temperature level and actuate the control. There were text boxes that displayed the current and desired temperatures as the process continued.

The implementation of tmpctlthread was simple. A flowchart illustrating the implementation of the thread is presented in Figure 5.4. Once a threshold level was fixed, on the CNi16D, no further decisions about switching the heater on or off were required. Upon starting, the thread checked to see whether the required temperature was specified in the dialog box. If the required temperature was not specified and the automation class had spawned the thread during a run, the value of the threshold level was read from the automation class. The thread

converted the required temperature value to its equivalent hexadecimal number and passed the value to the CNi16D. The algorithm is presented in Figure 5.5.

At this point the actual control of the temperature was complete. However, when the threshold was reached the automation thread had to be notified so the run could continue. This was accomplished by calling a function called *comptemps.* This function read the current temperature level and compared it to the threshold value once a second. If the two were equal a variable in the class was set. The automation thread monitored this variable in order to determine if the experiment could proceed.

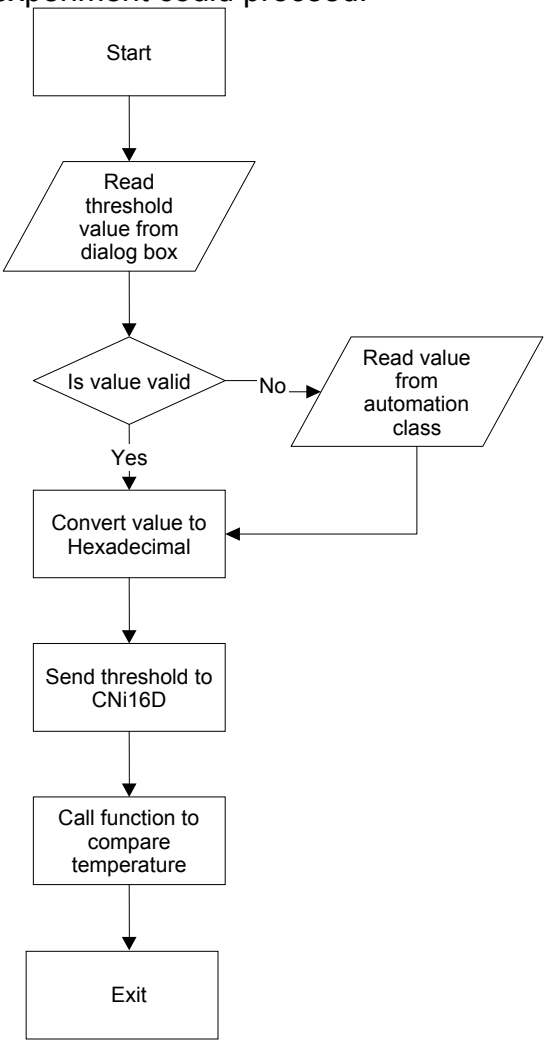

Figure 5.4: Flowchart For Temperature Control Thread

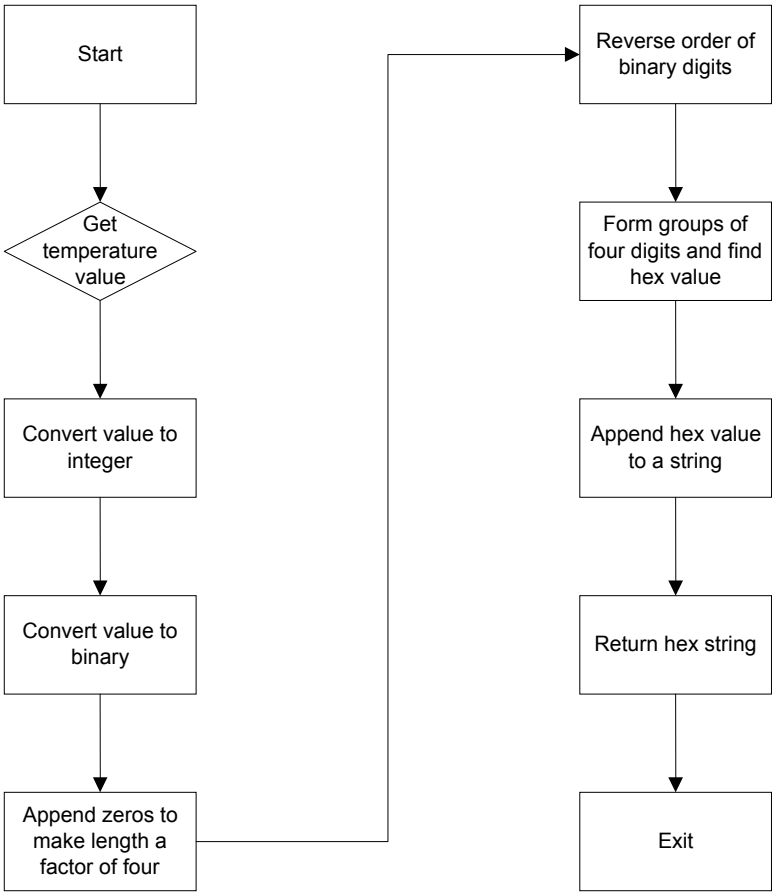

Figure 5.5: Function To Convert A Decimal Value To A Hexadecimal String

5.1.5 The Humidity Control Thread

The Humidity Control dialog box was created during the creation of the Automation class. However, like the temperature control dialog box, it was kept hidden by the system until its use was required. Pressing of a button on the Automation window spawned a thread called humctlthread that displayed the window and exits. The value of the required humidity could be specified in this window. The Start button on the window activated the humidity control. The humidity control window could be used during chamber characterization or for other purposes when neither of the automation threads was running. Another

way to control humidity was to use the *inhumthread.* The newautomthread started this thread, along with the temperature control thread, at the beginning of a run.

The manner is which humidity control was implemented was markedly different from the way temperature control was implemented. Temperature control simply required setting the threshold value on the CNi16D and notifying the automation class when the desired temperature was reached. This was possible because the CNi16D was specifically designed to monitor and control the temperature at a user-defined value. In the case of humidity control, there was no single instrument that could implement control. Humidity control used a combination of a DAQ board, a humidity sensor and a relay board.

When execution began, the thread first read the values of the required and current humidity levels. If the humidity control dialog box was not open the required humidity value was read directly from the automation thread. The current value was read from the humidity control dialog box. Although this dialog box might not be open, it existed, which made it possible for the required value to be obtained. All the relays were put in the disabled state so that neither the humidification nor the dehumidification apparatus was active.

The actual part of the thread that implemented the control was in the form of a while loop. The loop checked the humidity level every second and made decisions based on the current value. The thread made use of *if* statements to make decisions. There was a dead band of 3 percentage points around the required humidity level. When the actual humidity was in the dead band region

the thread switched off the control apparatus. Specific commands and data types were used to control the operation of the relays. These commands and data types were part of the libraries supplied by National Instruments. They were incorporated into the software. The flowchart presented in Figure 5.6 illustrates the humidity control technique.

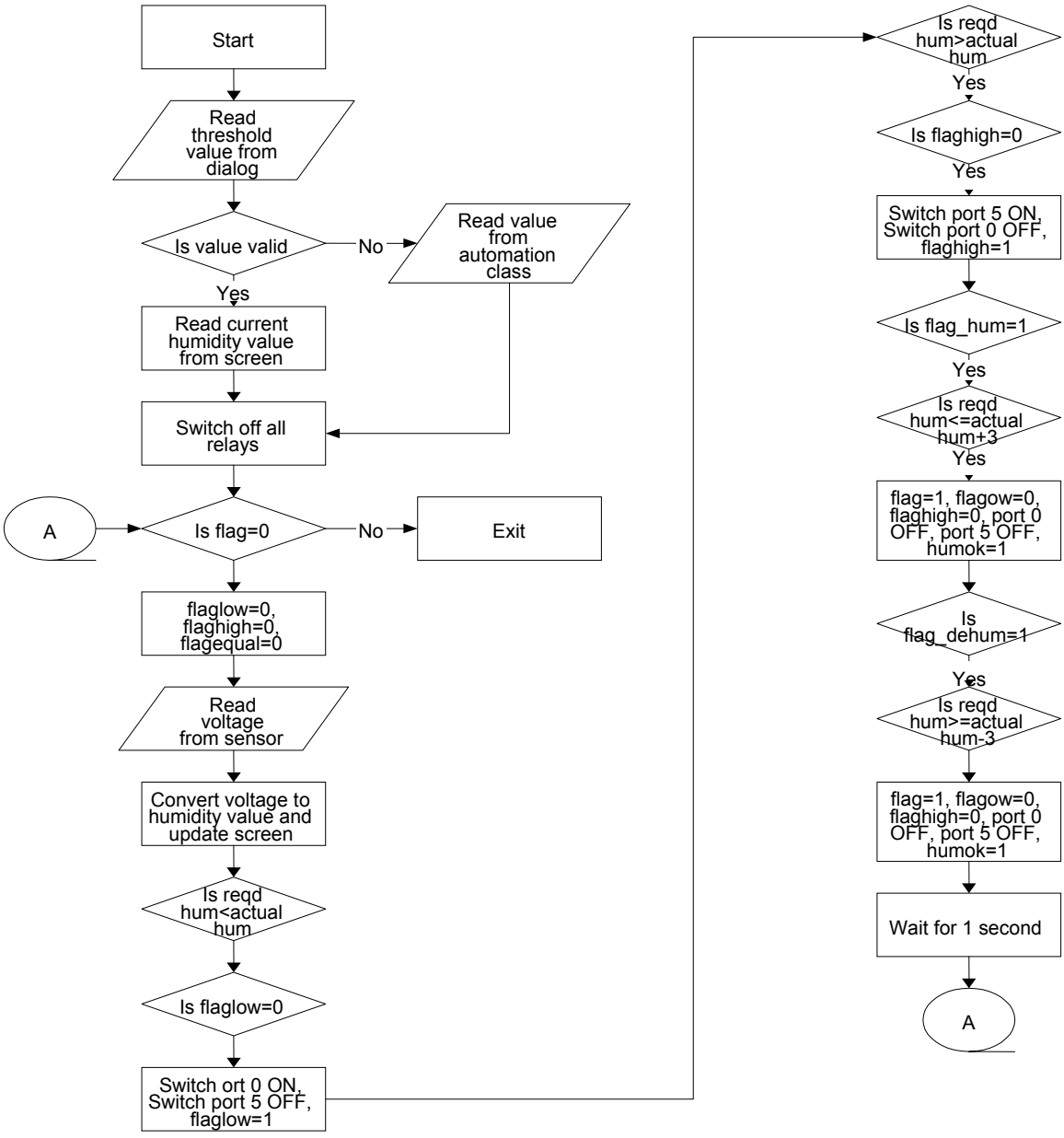

Figure 5.6: Flowchart For Humidity Control

6 Results And Further Work

The previous chapters provide an account of the development process for an application that integrated several instruments and brought them under the control of a single software program. This chapter provides some results obtained by using the various capabilities of the software program.

The main purpose of designing and building this software was to analyze polyurethane pads and study their response to varying physical conditions. The results can be used to find ways of making the pads more resistant to such changes, thereby ensuring a longer working life.

Pads of various dimensions and geometry were tested using the system. The following sections present results from tests performed on a pad using statistical analysis.

6.1 Surface Uniformity Testing

There are various types of scans that can be used to collect statistical data from the pads. The system was designed to enable operators to perform various scans such as random, linear, full and sector-wise scans. This section deals with sector scans.

The circular pad under study had a diameter of thirty-two centimeters and a thickness of four millimeters. For the purpose of analysis the pad was divided into three 120° sectors. A zero position was marked on the pad as a reference

point for the radial motor. The three sectors ranged from 0°-60°, 120°-180° and 240°-300°. These three sectors were identified as Sectors 0, 1 and 2 respectively.

There were four parameters associated with every run. The values of these parameters determined the number of observations made and the density of points in every sector. The four parameters were as follows:

- Position: Each sector on the pad was identified by an integer. The value of position specified the sector over which the measurements were made. A value of “1” specified sector 1 (120°-240°).
- Radial Increment: This value specified the separation between consecutive points on a line of the sector. A value of “8” would leave eight millimeters between consecutive points.
- Angular Increment: Similar to Radial Increment, Angular Increment was the angular separation between two neighboring lines on a sector.
- Depth: This parameter determined the depth to which the sensor pushed into the pad to determine the response at a point.

The results obtained from the runs were stored in text files. These values were analyzed using the statistical F-test to determine the uniformity of the pad over a given sector. The tabulated results are presented in Table 6.1.

Sectors 1 and 2 were observed to be uniform as the probe moved along the sector in a circular fashion. However, for these two sectors the radial variations were significant. The observed readings varied as the probe was

moved along the pad from the periphery to the center. Sector 3 was non-uniform along both directions.

Table 6.1: Surface Uniformity Test Results

Comparison	F-test result
Sector 0 (0° to 60°): Line-wise	Insignificant
Radial	Significant
Sector 1 (120° to 180°): Line-wise	Insignificant
Radial	Significant

Calculating the correlation between the observed readings in two sectors provides an idea about their physical similarity. A correlation close to 100% signifies that the sectors have a similar surface whereas a low value of correlation is an indication of non-uniformity between sectors. The particular pad tested proved to have significant variations between sectors.

6.2 Effects Of Ambient Conditions

The pad response to changing conditions of temperature and humidity was the focus of this testing. As in the previous tests the pad was divided into three sectors of equal size, see Figure 6.1. Two more parameters, temperature and humidity, were introduced. The parameter values are presented in Table 6.2

Table 6.2: Test Parameters

Position	Radial Increment (millimeters)	Angular Increment (degrees)	Temperature (° Fahrenheit)	Humidity (%)
0	18	12	80, 90, 100	50, 60, 70
1	18	12	80, 90, 100	50, 60, 70
2	18	12	80, 90, 100	50, 60, 70

Experiments using the parameter values in Table 6.2 yielded nine sets of readings for each sector.

Figure 6.1: 60° Sectors On Pad

The analysis in section 6.1 was carried out using the Single Factor Analysis method. In this case the only factor that caused variation in the pad's response was the location of the point on the pad where the reading was taken. In order to check for the effects of temperature and humidity on the pad's response a Multifactor Analysis was used. The results from this test are presented in Table 6.3.

Table 6.3: Multifactor Analysis Results

Factor	F-test Significance Sector 0	F-test Significance Sector 1	F-test Significance Sector 2
Temperature	Yes	Yes	Yes
Humidity	Yes	Yes	Yes
Temperature*Humidity	Yes	Yes	Yes

The results presented in Table 6.3 show that Temperature and Humidity affected the pad's response in every sector. The third column signifies that these two factors acting together also have an effect on the pad. Therefore, if these

two factors change together the pad's response would be different than the response obtained when one factor remains constant and the value of the other factor is allowed to change.

6.3 Future Work

The statistical experiments conducted demonstrated that the system is capable of testing the pads for their surface uniformity. A high value of uniformity is desired in pads since this would lead to better polished IC wafers. The experiments also demonstrated that the data acquired from the system can be used to analyze the behavior of the pads under changing ambient conditions. This analysis is very important since it helps in studying and improving pad properties, which will ultimately lead to better yields in IC manufacturing.

Further statistical tests that use regression analysis are to be carried out on the data taken by this system. These tests will determine the way in which the changes in temperature and humidity affect the pad's response. These tests can determine whether the pad's response will increase or decrease with a rise or fall in the values temperature and humidity. These results are invaluable in characterizing the pad's behavior.

This system was designed to enable researchers to test pads using ultrasound testing. This system can be modified to run tests on pads using laser beams. In laser testing the response of the pad at a point is measured in terms of the extent to which a laser-beam aimed at that point is scattered. The modular nature of the software makes it very easy to integrate this type of testing into the system. All that will be required is the inclusion of the device drivers for the laser

interferometer and the development of a new class that controls the laser. Coding for the laser experiments should follow the same philosophy as that for ultrasound testing. A thread spawned by the main program can move the laser over the pad and record readings as required.

References

[1] Test Automation and Design of Experiments for Microelectronics CMP Pads, Franklyn A. Diaz, December 2000

[2] Introduction to Object-Oriented Programming Using C++, Peter Müller**,** Globewide Network Academy

[3] Learning the Java Language, 1995-2000 Sun Microsystems, Inc.

[4] J.P. Mueller, (1998) “Visual C++ 6 from the ground up”, Osborne/McGraw-Hill

[5] http://www.taltech.com

[6] http://www.signalrecovery.com

[7] VELMEX, INC., VP 9000 Motor Controller, Command Manual

[8] Tektronix PS2520G Programmer Manual

[9] EG&G instruments, Inc. (1997), Model 7260 DSP Lock-In amplifier instruction manual, Princeton Applied Research

[10] SC-205X Series User Manual

[11] SC-206X Series User Manual

[12] OMEGA, iSeries Controller Manual

[13] HIH-3610-001 Data Sheet, http://www.honeywell.com

Bibliography

[1] OMEGA, iSeries Communication Manual

[2] Visual C++6: The Complete Reference, Chris H. Pappas, William H. Murray, III, Osborne/McGraw-Hill

[3] C++: Effective Object-Oriented Construction: concepts, principles, industrial strategies, and practices, Kayshav Dattatri, Prentice Hall PTR, c1999

Appendices

Appendix A: The RS-232 Standard

RS-232 Cables are commonly available with 4, 9 or 25-pin wiring. The 25-pin cable connects every pin. The 9-pin cables do not include many of the uncommonly used connections. The 4-pin cables provide the bare minimum of connections and have jumpers to provide handshaking for those devices that require it. These jumpers connect pins 4, 5 and 8 and also pins 6 and 20.

In the IBM PC AT and many new expansion boards a 9-pin serial port replaces the 25-pin connector. A 9-pin to 25-pin adapter cable can be used to connect this port to a standard 25-pin port.

Table A.1 and Figure A.1 present the transmitted and received signal voltage levels and the protocol data format respectively.

Table A.1: RS-232 Signal Levels

Voltage Levels	Transmitted Signal	Received Signal
Binary 0	+5 to +15 Vdc	+3 to +13 Vdc
Binary 1	-5 to -15Vdc	-3 to -13 Vdc

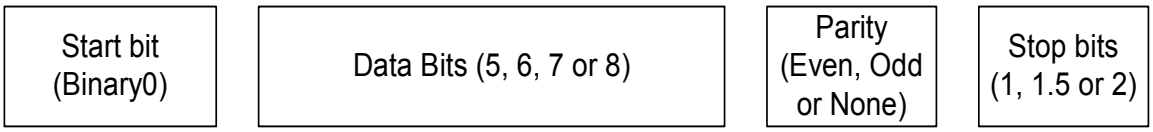

Figure A.1: RS-232 Data Format

Appendix B: Programming For DAQ

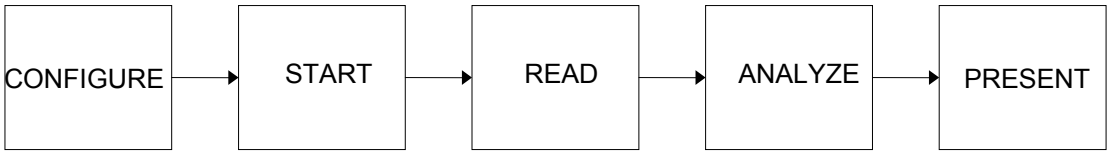

Figure B.1: DAQ Programming Flow

The programming flow for data-acquisition devices is presented in Figure B.1. These devices are programmed through register access to the various chips and components on the hardware and by low-level calls to the operating system to control the bus, map hardware and allocate PC memory. Nearly all data-acquisition devices include high-level driver software, which isolates users from any low-level programming. The driver software allows users to configure the hardware so that the operating system recognizes it and provides a basic programming interface.

The user selects channels, input ranges and buffer sizes during the configuration step. During the start step the user instructs the device to start acquiring data when it receives a trigger. In the read step the user directs the program to transfer data across the bus to the PC memory. Once in memory, the data is available for analysis and manipulation. The device can present the results in a numeric, graphical or tabular format.

Appendix C: Programming For Instruments

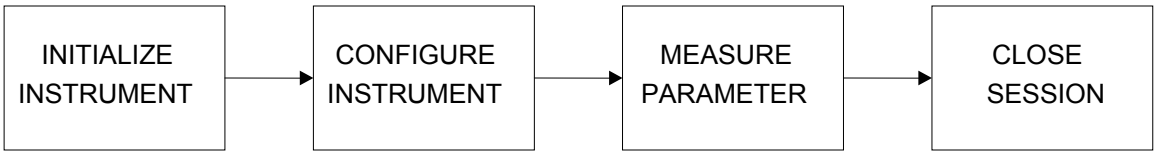

Figure C.1: Instrument Programming Flow

The programming flow for instruments is presented in Figure C-1. Instruments are programmed via text commands or messages sent via I/O interfaces. Interfaces have an associated driver that allows the user the send commands to the device over the bus. Such drivers are incorporated into industry standard libraries that hide details from the users of the specific bus. Therefore, the programs written are indifferent to the type of interface being used.

Instrument programming also follows a standard progression of calls as shown in the Figure C.1. Upon initialization the instrument is prepared to receive commands. Then the instrument is configured to make a specific measurement. After the measurement is completed communication with the instrument is closed in order to provide access to other applications.

Appendix D: Software Documentation

This document is intended to serve as a manual for operating the system by using the software program interface. Interfaces presented to the user during execution of the program are explained in the same order as they appear.

D.1: Login

The Login screen is a security feature that prevents unauthorized use of the system. Legitimate users can gain access to the system by providing their login ID and password in the textboxes provided. The login-password combination is validated from a database in the software.

D.2: Main

Prior to displaying the Main screen the system initializes all the required hardware and communication channels. The text boxes in this screen were used to control axes positions and signal parameters but are now redundant. These variables are now controlled by other interfaces, which are explained later.

The *Device* menu-tab on this interface displays the screens to control various instruments used by the system. The *Settings* menu is used to open the *automation* window. The automation window controls various types of experiments offered by the system.

D.3: Lock-In

The Lock-In screen is used to find the optimum oscillator frequency at which the pad yields the maximum response. The required amplitude and the

initial frequency are specified. The frequency sweep is started and the graphs are updated as readings are taken at the specified amplitude. The optimum amplitude and frequency were found to be 0.5 Vrms and 26 KHz respectively.

D.4: Motion

The Motion screen can be opened from the main as well as automation screens. It is used to control and display the position as well as the speed of motion of the sensor. The text boxes titled *Big* are used to specify the distance through which the sensors are to be moved. The value in millimeters is multiplied by a factor of 10 and written into these boxes. There are two boxes, one for linear and the other for angular motion. The speed of motion is varied by checking the *small* or *fast* radio buttons. Small provides a speed range of 1500 to 2500 steps/second and fast provides a range of 2500 to 3000 steps/second.

The *home* buttons for the respective axes move the sensor to the home or zero position of that axis. The movement is started by pressing the arrow buttons once for moving the axes through one step.

D.5: Contact Point

Altering the depth parameter significantly changes the value of the measured signal. It is therefore necessary to accurately determine the depth at which the measurement is made. The zero position of the Z-axis is fixed and therefore the distance from this position to the surface of the pad is required to specify the depth.

Appendix D: (Continued)

The *Find Contact Point* function is invoked from the automation screen to measure this distance. This module does not require any input from the user. However, before starting this program it is necessary to use the motion menu and move the sensor to a point over the pad. The function uses an algorithm that differentiates between successive readings to detect contact between the sensor and the pad. The jump in the response signal is used to determine the location of the contact point. The contact point and the intermediate readings are displayed on the screen.

In order to get an average value for this parameter the function can be run at various points on the pad and the readings can be averaged.

D.6: Automation

The automation window specifies parameters for runs and starts and monitors experiments. Every run has four basic parameters associated with it; sector, radial increment, angular increment and depth. Temperature and humidity levels are specified if the effect of these factors on the pad is under study.

The four parameters, sector, radial increment, angular increment and depth, can be specified on the screen in the dialog boxes provided. The start position, end position and increment are specified for the X and Radial axes. Pressing the *linear start* button starts the experiment.

Appendix D: (Continued)

The file to which the readings are to be saved is specified in the *File settings* area. This file can later be used to provide a graphical representation of the pad's surface based on the observed readings.

It is also possible to use the system in such a way that successive experiments are carried out with various combinations of the parameters. The parameters for every run are written line-by-line in a text file.

The parameters are specified in sequence of sector, angular increment, radial increment, depth, temperature and humidity. At the end of every run the system resets itself and begins a new run with the new parameters on the next line in the control file. In this mode the start and stop angle and radius are not specified since these are fixed at 0°, 60°, 250 mm and 154 mm respectively. The radial values depend on the pad's size and have to be changed in the software to values that depend on the pad's radius.

The automatic mode is activated from the *Choose file* area by selecting the *Run from file* checkbox. The *Automation control* area starts and stops runs as well as displays the readings as they are taken. The *Data Graph* button opens the Graphical screen. The *Find C.P.* button is used to view the Contact Point window.

Appendix D: (Continued)

D.7: Graph Data

This screen allows the user to view a 3-D graph of the readings. The color displayed at a point on the graph depends on the value of the reading at that point. The user can also view x-y, x-z, and y-z axes projections of the graph by selecting the corresponding checkboxes on the right side of the image. The graph and color style is selected by checking the required radio button. The program offers nine graph styles and three color styles to facilitate analysis.

In order to graph a particular data set the *Graph Data* button is pressed. The required file is selected from the file browser that opens.

D.8: Temperature And Humidity Control

The programs to control these parameters can be started in two ways. During chamber characterization, pressing the *Temp. Control* or *Humidity Control* buttons on the automation screen displays the respective screen. The required value for the parameters can be specified in these screens. Upon activating the *Start* button the temperature and humidity control starts and the changing values of the parameters are updated on the screen.

In the second mode the automation class activates these programs at the start of each run. The user has to specify the value of the required temperature first and then humidity in the control file after the other parameters. In this mode the dialog boxes for the programs that control these two factors are not displayed. The value of the current temperature can be read from the front panel of the CNi16D and the humidity is displayed on the automation screen as it

changes. The experiment does not start till these parameters reach their required values.

D.9: Statistical Analysis

The statistical F-test is used to determine the uniformity of the pad. The statistical analysis toolbox does this test on a set of readings and determines whether the pad is uniform or not. The user has to browse and select the file in which the readings for a run are stored. Upon activating the *Go* button the program compares the calculated and tabulated F-values and provides the result in the textbox provided.