FINAL REPORT

DEVELOPMENT OF CHEMICAL MEASUREMENTS USING MICROELECTRODES

Principal Investigator:

Arland H. Johannes
School of Chemical Engineering
Oklahoma State University
Stillwater, OK 74078-0537

For the Period
July 1, 1986 - June 30, 1987

Submitted to:

University Center for Water Research
Oklahoma State University

July, 1987

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Measurement of chemical concentration in extremely thin biological films (membranes, slimes, etc.) is difficult and current methods are crude. One of the difficulties for direct investigations of these slimes is their thinness, which is on the order of microns. Also, the thickness of the boundary layer covering these slimes is found to be extremely thin even at high Reynolds number.

A delicate balance exists in biological systems between the biomass, required nutrients and oxygen concentrations. New biochemical and biomass applications require precise measurements of chemical concentrations on a microscale. Microelectrodes are highly desirable in these areas of applications since their extremely small tip size (0.5-5 microns) does not damage the cells in the bio-mass. Such minute amounts of biomass are required for sampling that it does not alter the state of the system. The other advantages of these microprobes include; freedom from effects due to flow, stirring or mechanical pressure. The microprobe is also capable of measuring very small concentrations. Past investigations involved the use of these microelectrodes to measure mass transfer coefficients

in biological slimes. Different types of microelectrodes, such as, pH, carbon-dioxide, ammonia, sodium and potassium microelectrodes have recently become commercially available. The main disadvantage of micro-electrodes is the extreme fragility due to the small tip size (generally 0.5 to 5 microns). Proper handling and careful storage of the probe are essential for extended probe life. Most microprobes are broken due to imprecise positioning in the sample.

The single greatest weakness of all the past studies utilizing microelectrodes has been, not knowing where the tip is with respect to the reference point, film surface and the underlying support surface. These studies were based on a very limited number of measurements and did not measure concentrations as a function of position in the slime due to extreme difficulty of repositioning the electrode for each measurement.

The objective of the present research work is to solve this problem by designing and building a real-time computer controlled micropositioning system. The system was controlled by a Texas Instrument professional microcomputer using state of the art micropositioners. This system has been built and tested. The programs are designed in C and Basic languages. The following contains a detailed description of the system, how it was developed and built, its important features and recommendations for future work.

CHAPTER II

LITERATURE REVIEW

Microelectrodes

The use of microelectrodes is steadily finding increasing areas of application in biological and medical research (1) and several areas of chemical engineering, such as study of mass transfer through gas-liquid interface. These devices measure the chemical activity instead of concentration of ionic species. This is important since biological phenomena are functions of ionic activity, and in a biological system there are significant ion-complexing and ion-association phenomena (2). This measurement is accomplished simply and quickly with few interferences from other species. There is also the unique potential for making measurements in living system in-vivo under representative conditions. If the tip diameter is sufficiently small, entry into a cell causes negligible injury thus the activity should closely approximate that of the undisturbed system.

The first ion selective electrode, a pH electrode, invented by Cremer in 1906 was a solid state unit based on antimony. In the 1920s glass electrodes filled with an

electrolyte for measuring conductivity were in common use. However these electrodes needed a high-input-impedance amplifier. In 1953, Dowben and Rose (3) invented a metal filled microelectrode that satisfied the need for an electrode of low impedance. A glass "wetting" metal, an alloy of gallium and indium, was drawn into a glass microcapillary. This microelectrode was sturdier than saline filled electrodes of comparable size. Based on this concept, Whalen et al. (4) developed the first oxygen microelectrode in 1967.

Different types of microelectrodes are now available from private and commercial sources. The polarographic type microelectrodes, which are described in the literature (3), are currently made by Nair (4) of the Cleveland Research Institute. Many modifications can be made in the probe, primarily dealing with the tip design and length of the recess.

Past Studies Using Microelectrodes And Its Applications

The microprobe is used in physiologic research for measuring intracellular oxygen (pO_2). If the diameter of the tip is sufficiently small, this can be done without damaging the cell, or interfering with the blood supply. Thus the activity measured closely approximates an undisturbed system. In-vivo measurement of pO_2 in brain, heart, and skeletal muscle have shown that pO_2 normally

fluctuates with time and the cell pO_2 is lower than that of the venous outflow (5). The oxygen microelectrode was also adapted as a hypodermic needle pO_2 electrode (6). It is advantageous in many situations to use a stainless steel needle as the anode in the system, rather than having a separate anode and a cathode. However, the needle causes tissue damage due to its large tip size. Another adaptation of the oxygen micro electrode has been as a part of flow through pO_2 sensor which is easy to use and has excellent long term stability and eliminates clotting problems (7).

A membrane-covered, platinum, polarographic microelectrode has been used for an amperometric assay of dissolved oxygen in marine sediments (8). The oxygen level of marine sediments may be a limiting factor for the Benthic community. This type of information allows a more complete characterization of the Benthic community. Oxygen profiles of the sediment were recorded during a light-dark cycle. These profiles were used to estimate the rate of oxygen production and consumption, and to calculate the apparent diffusion coefficients for oxygen in the sediment. In general, the depth of oxygen penetration was limited to 1 - 3.5 mm. The depth of oxygen production was related to the rate at which oxygen is consumed in the sediment (9). Oxygen transport into the deeper layer (5-10cm) was mainly a function of macrofaunal activity such as by burrowing animals(ie., clams, flounders, etc).

Microbial slimes are found in natural waters affecting

private, recreational, municipal and potable water use. Fixed film reactors utilize microbial slimes for biological waste treatment of municipal and industrial waste. Microbial slime growth is often undesirable and has adverse effects on systems, such as, industrial cooling towers or water distribution systems. Therefore, a better understanding of the mass transfer of oxygen into the slime system can lead to improved design and control of microbial systems.

The oxygen probe was first applied to environmental engineering applications in 1968 when Whalen et al. (10) measured dissolved oxygen profiles in laboratory grown slimes. Additional work was performed with slimes to determine the respiration rate and diffusivity of oxygen (11, 12). Oxygen profiles for slimes under natural stream conditions as well as profiles for trickling filter slimes were determined (13, 14, 15). Oxygen profiles were produced by changing the illumination of the slimes, thereby examining the effect of photosynthesis on the oxygen production.

The microprobe is also used to characterize oxygen transfer into activated sludge flow (16) and into mycelial pellets (17). A study was also done to test the feasibility of using the microprobe as an assaying tool for trace quantities of toxic chemicals (18). The chemicals used for this study were phenol, potassium cyanide and copper sulphate. It was indicated by the results that the

respiratory and the photosynthetic functions of Chlorella Vulgaris (green algae) were not discernably effected by trace concentrations of these toxic chemical during short term exposure of these substances. However, chronic effects of long term exposure to trace concentrations was not studied.

The survey of literature clearly indicates that:

1. The techniques used in the past are not sophisticated enough for measuring slime concentrations at very low depths. Whalen et al. (10) tried to locate the slime surface and the underlying base within a distance of two microns from the base by a trial and error procedure, since the resolution of the movement of the probe was much higher (25 microns).
2. Previous studies involved simulating a system and then proposing a mathematical model to suit the system. No real-time measurements were made. Precision was generally poor. According to Bungay and Harold (19) greater accuracy can be obtained by specifying small error tolerances and by taking more layers and closer slices of slimes which is only possible through a computer controlled system.
3. There is a need for a more sophisticated computer controlled system to make accurate real-time measurements. Bicher and Knisley (20) measured brain tissue reoxygenation time manually with a micromanipulator (in 10 microns steps) using a

ultramicro oxygen electrode. The type of accuracy needed in such measurements which is in the order of 1 - 2 microns can be obtained through a computer controlled system.

CHAPTER III

CONSTRUCTION OF MICROELECTRODES

The type of microelectrode shown in Figure 1 is made by filling a glass capillary tube with Wood's metal and pulling it in a pipette puller.

About 5g of Wood's metal, with a melting range of 73 - 75°C, is melted on a hot plate and mixed with 0.3 -0.5g of precipitated gold powder. An initial heating of 300°C forms an alloy. This alloy remains bright and exposed to air for several days, and requires aqua regia to dissolve it. Yet, it adheres to glass as well as or better than, the Wood's metal by itself. There are three stages involved in preparation of the microelectrodes. These are illustrated in Figure 2. In stage one, a glass capillary tube (0.9 mm OD - 0.4 mm ID) about 4 inches long is filled by suction to about half its length with the molten alloy. The suction is applied by using the syringe fitted with vinyl tubing which fits snugly over the capillary tube. In stage two, the glass capillary tube is placed in a pipette puller with the top edge of the metal at the top of the heating element. The glass capillary is pulled out to a tip of 1 -2 microns. Usually, the metal does not extend all the way to the tip. In stage three, the base end of the capillary is heated, and

then a wire (usually iridium 0.2 mm OD, crimped in a Burndy contact (Burndy corp., Norwalk, Connecticut), is inserted into the metal alloy. The tip is warmed again over the hot plate to force the metal towards the tip, resulting in an electrode having a recess of 10-30 microns at the tip. The electrode is placed in a gold plating solution for several minutes and the metal in the recess is electroplated with a layer of gold, using 0.1 to 0.5 volts. For an electrode of 2 to 3 microns, the plating time is 30 to 45 seconds. The plating solution is removed by allowing the electrode to stand in distilled water for several hours, then for two hours in 95% ethanol. Finally, the electrode tip is placed in collodion for 5-10 minutes. Collodion improves the performance of recessed oxygen electrodes.

The advantages of using such microelectrodes are listed below:

1. They have a rapid response time.
2. Similar calibration curves and current-voltage plateau relationships are obtained in different media.
3. They show no effect of stirring. This is attributed to a stagnant layer, a few microns thick, around the electrode tip. Since the electrodes (cathodes) are small they cannot see beyond this stagnant layer.
4. They show little ageing or poisoning by the microbial system.

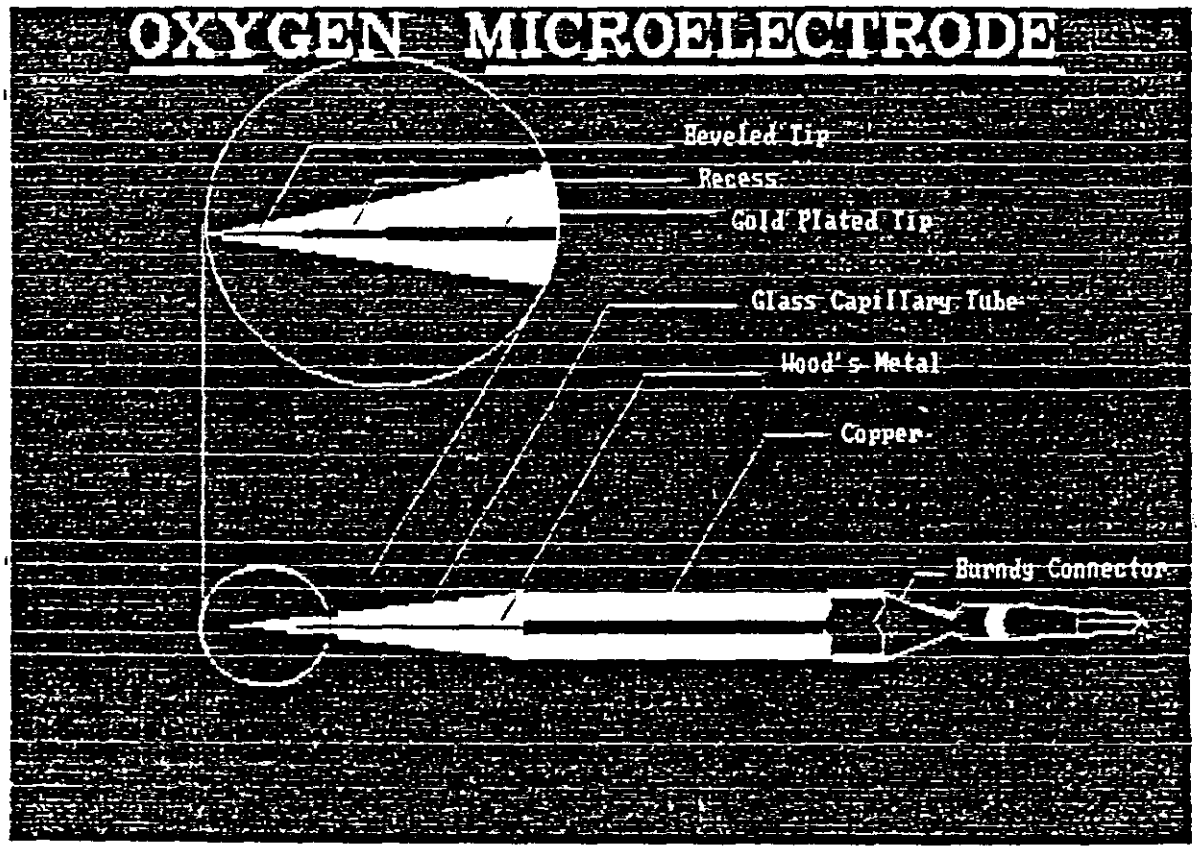
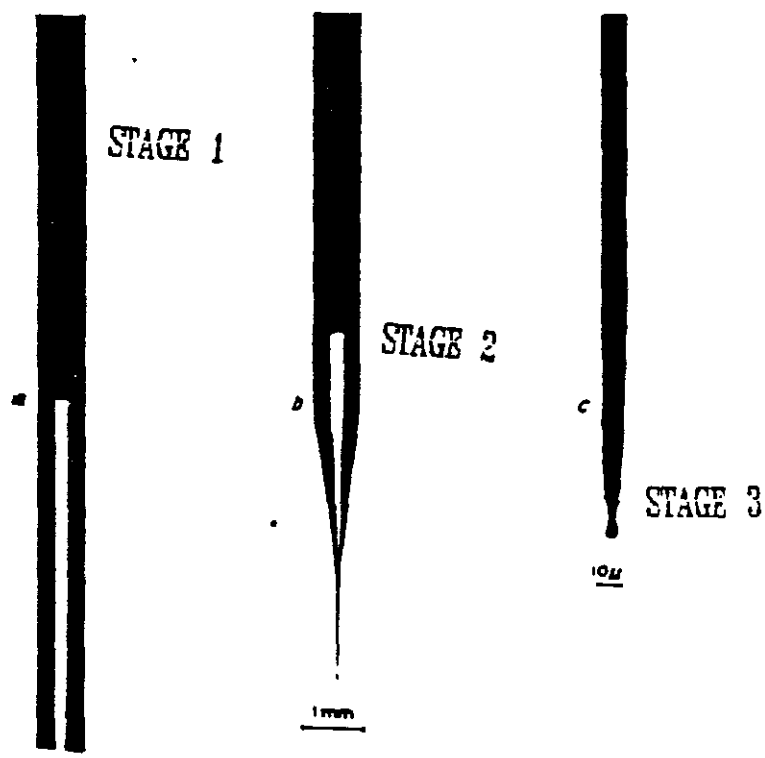


Figure 1. The Oxygen Microelectrode

THREE STAGES IN PREPARATION OF THE MICROELECTRODE



Source: Dowben., R. M. and J. E. Rose, "A Metal-Filled Microelectrode," Science (1953).

Figure 2. Three Stages in the Preparation of Microelectrodes.

CHAPTER IV

COMPUTERIZED MEASUREMENT SYSTEM

A computerized measurement system can collect information and process it efficiently with the least effort on the part of the operator. Careful planning is the key to get maximum utilization out of a computerized measurement system. First, the following must be determined:

1. How often to sample the data signal and with what resolution.
2. Choose what computations the computer will perform on-line,
3. Decide what information to process in real-time and what measurements to save for later processing.

Real Time Definition

The term applies to the use of a computer in conjuction with some external "process". The object of this interconnection is to obtain information from the process by monitoring its operation through measurement of important variables. It is also used to operate in some desired fashion and to control the way in which it operates based on the information previously acquired. For the computer to accomplish these objectives, its operations have to be

carefully sequenced in time. This is called "Real-Time". It implies that the computer has the ability to respond to stimuli from the process in a timely fashion, i.e., sufficiently fast to accommodate the needs of the process. For example, if some emergency conditions arise in the process and is signalled to the computer, the computer must be capable of reacting to the process requirements fast enough to handle the emergency. The idea of real-time response requires careful attention in the selection of the computer and in designing the total real-time system.

A common feature of the computer system involves the physical means of connecting the process to the computer. The measurement equipment used to connect the physical process to a computer is called a "Computer Interface". Every real-time system is a unique creation. Hence attention needs to be paid to the structure of the computer system, to the interface, and to the programming. The other term used commonly in place of "Real-Time" computing is "on-Line" computing.

Real Time vs Batch Signal Processing

Any result used for feedback during a measurement session must be calculated in real-time. These results might be used in experimental control or perhaps provide a visual display as the data is being collected. Alternatively if the data is not needed for feedback, it must be saved for later analysis.

In real-time the data processing that occurs, reduces the information content in signal and thus precludes some type of later analysis. Saving raw data for batch processing can therefore allow greater flexibility, and can possibly save rewriting a real-time data collection program, whenever a new type of data analysis is desired.

The second major advantage of batch processing is the ability to perform computations that require more time than available in real-time.

The advantages of real-time data processing are also two-fold. First, the amount of data that needs to be printed or saved on mass storage devices is greatly reduced. The second advantage is that it does not have to be done later.

Normally, any measurement situation with more than 1,000,000 data values per hour must be considered for real-time computing.

Structure of Real Time Systems

Any digital computer is composed of 4 sub units:

1. Arithmetic unit: It contains all the hardware necessary to carry out arithmetic and logic commands. All the components in the computer are constantly under the supervision of the control units.
2. Control units: This part of the computer is responsible for reading a program from the memory, interpreting it and causing appropriate action to take

place.

3. Memory unit: This is used for the storage of the data and the computer program itself. Normally the control unit causes a sequence of program statements stored in a consecutive memory to be executed.

4. Input/Output Interface: The I/O interface is necessary for the computer to communicate with all of its peripheral equipment. The interface generally consists of set of bi-directional data lines and control lines, usually referred to as buses, and the logic necessary to detect and respond to external "events". These events usually take the form of a request for some kind of action on the part of the computer which than would have to interrupt its normal processing. The ability to respond to the external "interrupt" is a requirement for the computer. It is this capability that allows the real-time computer to keep track of time, independent of its normal operations, and to watch multiple processes, each with a different set of commands which must be serviced by the computer. As with the other three computer units, the I/O interface operations are coordinated by the control unit.

Selection And Justification of Computer Language

FORTRAN and BASIC are two major types of high level

computer languages. FORTRAN is a computer based language, meaning that a FORTRAN source program must be translated into a series of steps involving the computer itself before execution can take place. These steps are:

1. The FORTRAN program must be compiled, i.e., read into the computer where the compiler converts each high level statement into a correct sequence of assembly language statement,
2. The assembly language program is then converted to machine language in the computer using the machine assembler, and
3. The machine language program is loaded and executed under the supervision of the operating system.

BASIC, by contrast, usually is an interpreter based language. Compiler versions of BASIC also exist. This means that the sequence of statements constituting a BASIC program is read into the computer along with the BASIC interpreter program and the operating system. The interpreter treats the basic program source statements as a set of data. In executing the program, it proceeds to look at each statement, interpret it as to specific functions, and call subroutines to carry out the functions. Hence a BASIC program does not become a executing program. Each time a statement is executed, it must be interpreted as if it were the first time. The operating characteristics of a high level language depend significantly on whether it is compiler or interpreter based. BASIC language programs run

much slower than the FORTRAN because of the extra time required to interpret. On the other hand the basic program can be modified on line, simply by typing in any desired changes and rerunning the program; whereas the FORTRAN program must have the changes edited in, then recompiled, reassembled, and reloaded before rerunning. BASIC has advantages for situations where the programs are developed continuously. FORTRAN has advantages where a fixed program once developed will be used for long periods of time.

In general, both programming languages require more memory for execution than does an equivalent program written in assembly language. Assembly language requires four times less memory than the programs generated by FORTRAN compiler, and will also execute faster. Inspite of such advantages of assembly language, high level languages are preferred because they simplify the task of programing, and make programing documentation and restructuring easier.

In the present work two high level languages are used - C and BASIC. Though C does not generate code as fast or memory efficient as assembly language, it is more elegant and powerful than other high level languages like FORTRAN or PASCAL. Another advantage of using C language is, it has powerful Input/Output functions which make real-time computing small and maneauverable.

CHAPTER V

INTERFACING FOR DATA ACQUISITION: SELECTION AND JUSTIFICATION

Generally the data may be acquired using either the main frame computer, minicomputer or microcomputer. The experiences that the people had in the past using minicomputers and mainframe were generally very discouraging. The basic reasons why microcomputers are important for data acquisition are as follows:

1. Minicomputers and Mainframes must be shared by more than one person. But in data acquisition it is crucial to have the computer's attention when the data is ready.
2. The main frame is not located in the laboratory. Thus in data acquisition contexts, there is a communication bottle neck created by the data transmission.
3. There is no common standard for interfacing laboratory instruments on large computers.

Interfacing for data acquisition may be achieved in two ways:

1. Using Analog to Digital converters.
2. Using Digital/Digital converters.

The important features and limitations of each are discussed briefly.

Analog/Digital Converters

The least expensive way to automate a laboratory is with an analog to digital converter, which converts analog signals to digital signals and vice versa. However, it has the following limitations:

1. An A/D converter samples only one voltage source at a time. A/D converter may be acceptable, but often the time lag is sufficient to make the data hopelessly imprecise.
2. A/D converters are slow (the maximum sampling rate on most "high speed" A/D is 100 kHz). This means that we cannot track a transient of greater than approximately 20 kHz. But the scientific data acquisition requires at least a few megahertz.
3. In A/D converters the boards are expensive. The linearity is not very good. A 12 bit board may have a resolution of only 7 or 8 bits.
4. The most important is that the A/D converters are very susceptible to noise in the laboratory. This may not give the level of noise immunity required in a laboratory environment.

Digital/Digital Converters

A D/D converter can communicate directly with a computer because both are digital devices. The digital to analog convertion step is not required in this type of

communication. This makes the D/D converter faster than the A/D converter.

The speed of data transfer is important because it determines how quickly the instrument can repeat an analysis.

D/D convertors are available in two types: 1) Serial Port, which transfers information one bit at a time, and 2) Parallel Port, which transfers one word at a time.

RS-232 Serial Port

The most common serial port is an RS-232C interface. Its disadvantages are as follows,

1. It is not a standard interface.
2. There are two ends to an RS-232 interface: The data terminal equipment end and the data communication end. Often the two instruments hooked together are configured as DTEs (data terminal equipments).
3. The handshaking provided is on the level of whole messages only. The interface does not verify that the data has been received before proceeding.
4. It is very noisy.
5. It can connect only two devices together.
6. RS-232 is slow since it sends only one bit at a time.
7. For multiple data sources, more than one RS-232C port is required on the computer making it very difficult to write the software.

IEEE-488 Parallel Port

1. The IEEE-488 is a byte serial, bit parallel that overcomes the problems of the interface outlines above.
2. The interface is incredibly resistant to interferences.
3. It provides excellent noisy immunity.
4. A very important feature of IEEE-488 is that the interface has a bus-structure, and up to 15 devices can be interfaced at a time using the same board. This structure simplifies the process control and allows true simultaneous data acquisition.
5. The interface is as fast as the microcomputer. Data can be transferred up to one million bytes per second (using special tri state drivers on the lines) and without any special care will support transmission rates of about 250-300K bytes per second using direct memory access (DMA).
6. The interface is standard and is widely available. All IEEE-488 instruments are plug compatible.
7. The primary limitation on the standard is that it cannot exceed 20 meters in cable length without expensive repeaters. And given long cabling slows transmission rates and is more susceptible to noise. In the present work a 4 meter cable is used.

Perhaps the greatest advantage of the IEEE-488 interface is that it is a standard interface. The use of

IEEE-488 began as a general purpose interface bus (GPIB) of the Hewlett Packard Corp. In 1975, the IEEE adopted the GPIB as its standard. Some minor modifications were made to the standard in 1978. But IEEE-488 still goes by the name of GPIB on HP products. Devices on the interface may perform three types of functions.

1. They may be talkers i.e., they may transmit data to other devices on the interface. There can be only one active talker at a given time.
2. Alternatively, a device may be a listener. It may receive data and instructions. There may be more than one active listener at a time.
3. Finally, a device may do nothing but standby. At different times may assume any of the above functions. The interface has two modes of operations - Command and Data. Command mode is for process control. In the Data mode, data is transferred from talker to listener(s).

The interface has 24 lines, out of which 8 lines are ground lines. The other 16 lines are divided into three groups. 8 bi-directional data lines, 3 data byte control lines (hand shake lines), and 5 general interface lines.

The three line handshake protocol functions as follows:

When the information is transferred over the bus the listeners must be ready to receive the data. If they are not they signal NRFD (not ready for data) by pulling the NRFD line low ("low" is defined as true by the IEEE-488

standard). The NRFD line has an open collector design, so that if one listener is not ready, the line is kept low. When all the listeners are ready, the NRFD block goes high. If the talker is ready to transmit data, it sets the DAV (data valid) line low. The transmission of the DAV triggers the resetting of the NRFD line, and the listeners pick up the last byte of data. When each listener receives the data, it releases NDAC (not data accepted) line, which is also an open collector. When all listeners have received the data the NDAC line goes high, causing the reset of the DAV line, which in turn triggers the resetting of the NDAC line. This information is repeated for each byte in transmission.

The description of the IEEE controller drivers, in both C and Basic languages, is provided in Appendix A. A program to test these IEEE drivers provided by Ziatech corporation, is located in Appendix E along with the listing of other programs.

CHAPTER VI

EXPERIMENTAL SYSTEM

Polarographic measurement of dissolved gases, especially of oxygen is most frequently done in the field of medicine and physiology. Micro sized electrodes with tip diameters of 1 - 25mm are used for measuring oxygen concentration in blood and tissues. Micro organisms are generally found attached to solid surfaces in bodies of water and in other natural environments. The critical role of these micro organisms, is the removal and degradation of organic materials in water and waste water systems. This sparked a national concern for water pollution and water pollution control. Fortunately, the introduction of electronics in chemical engineering has opened promising avenues of research to provide a greater insight into the study of microbial systems.

A system using microelectrodes has been developed and designed to effectively make chemical measurements. Figure 3 illustrates the diagram of the system which is divided into two control loops-the main control loop and the inner loop. The main control loop consists of a Texas Instrument Professional microcomputer designed to function in real-time with Oriel Corporation stepper motor

controllers. In turn, these controllers are used in conjunction with the micromanipulators for x, y and z directions micropositioning. The circular motion of the stepper motor drives is then translated into the precision linear motion with the help of translators. Also included in this main loop are the A/D converter and IEEE-488 interface.

The inner loop contains a microelectrode, an auto ranging picoammeter, a constant voltage source, and a voltmeter. The main control loop and the inner loop are interfaced in real-time through the IEEE-488 interface and the A/D converter.

The key to the system is that micromanipulators and controllers can under computer control be positioned within 1 micron. Position changes using this system can be operated in half and full step mode, corresponding to 1 or 2 microns per step. Maximum speed under full step operation is 500 steps per second i.e., 1000 microns per second. A ten pin collector located on the rear of the controller module allows for external control by the computer. Two remote control inputs drive the motor in either the forward or backward direction one single step per 5 volts TTL (transistor transistor logic) pulse received. Two additional inputs from the computer drive the motor in the forward or reverse direction at the speed control for the length of time the TTL (transistor transistor logic) signal is applied. Two out pins indicate motion in the forward or

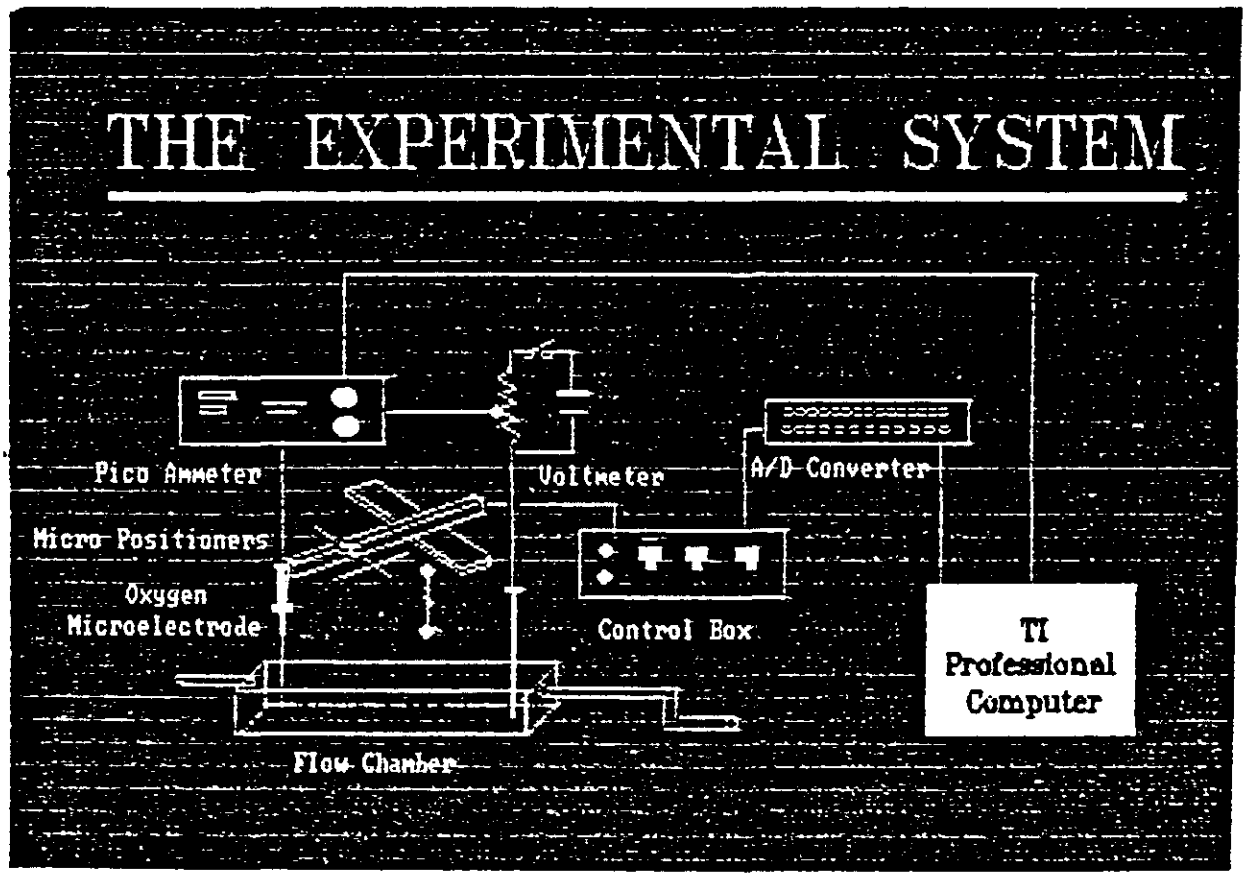


Figure 3. The Experimental System

reverse direction by a +5volt (TTL) pulse per step.

At this point it is necessary to explain why a stepper motor is used for precise positioning instead of a conventional motor. The stepper motor is a device which translates electrical pulses into mechanical movements. The shaft rotates through a specific angular rotation for each pulse, and this is repeated precisely with each succeeding pulse. The result of this precise, fixed and repeatable movement is the ability to accurately position the probe. A conventional motor has a free running shaft, while the stepper motor does not. The stepper motor shaft rotation is in fixed repeatable, known increments. The shaft rotation for this particular stepper motor for each complete step is 15 degrees.

Computer operation is accomplished by applying TTL logic"0" and "1" using the Data Translation's A/D converter model 2805. The digital I/O lines on this converter are used for this purpose. The digital input/output (also called digital I/O, or Dio) permits the Texas Instruments Professional computer to be used with stepper motor drives controllers which accept and supply parallel digital data. Parallel data requires a separate electrical connection for each bit.

CHAPTER VII

OPERATION OF THE SYSTEM

EQUIPMENT USED

- Texas Instrument Professional computer.
- Kiethly's Auto-ranging Picoammeter. Model 485
- Kiethly's IEEE-488 Interface. Model 4853
- Oriel Corporation's Micropositioners. Model 18503
- Oriel Corporation's Stepper Motor Controllers. Model 18548
- Data Translation's A/D Converter. Model DT 285
- Data Translation's Screw Terminal Panel. Model DT 707
- Ziatech's IEEE-488 Interface Card. Model ZT 1446
- Ziatech's Controller Device Drivers. Model C and BASIC Languages Software
- Diamond Electro-tech Incorporated's Oxygen Microelectrodes. Model 723(Po_2)
- Reference Electrodes Ag/AgCl.
- Plexiglass Chamber.

As mentioned before, the system consists of two loops: 1) A main loop run by the TI professional computer and 2) An inner loop to carry out the actual measurements. Each is discussed in detail.

MAIN CONTROL LOOP

The DT 2805 of the Data Translation is a complete single board data acquisition system for personal computers (IBM and compatible systems). This board has an on board microprocessor with a power supply. The Data Translation board is capable of performing :

1. A/D conversion.
2. D/A conversion.
3. Digital I/O transfers.

It consists of 16 channels of 12 bit A/D conversion, and two channels of D/A conversion. It also consists of 16 lines of digital I/O. This feature of digital I/O is split into two 8-line digital I/O ports which can be used separately to read or write 8-bit transfers, or simultaneously a 16-bit transfer.

All the channels for analog to digital and digital to analog conversion are easily accessible through a screw terminal shown in Figure 4. The terminal board also shows all the digital I/O lines used for I/O signals from the computer to the stepper. They are divided as port 0 and port 1, each port consisting of 8 lines of digital I/O.

The board is also capable of performing the following:

1. Reporting errors in the operation of the board while running the micropositioners.
2. Setting the period of the on board clock.
3. Stopping board operations in process and thus,

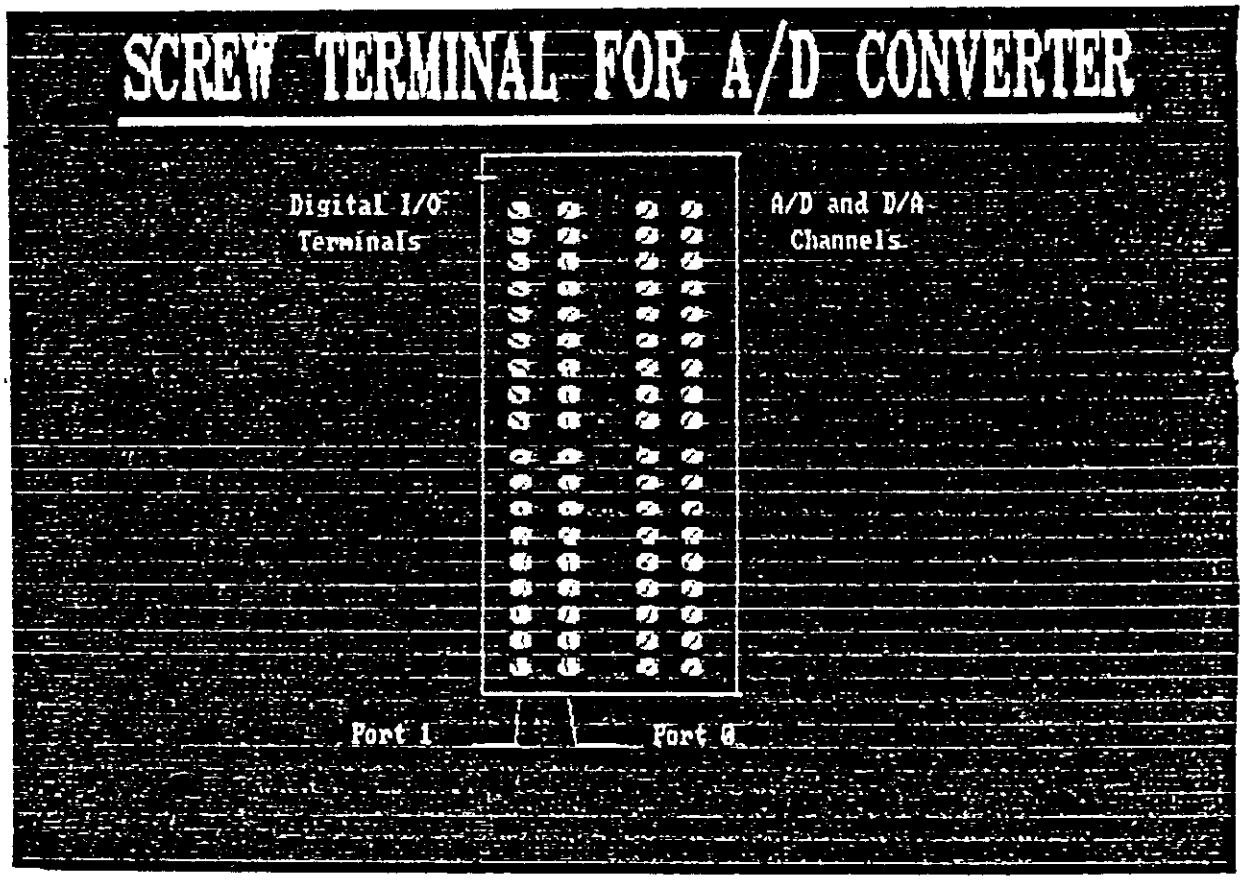


Figure 4. Screw Terminal for A/D Converter

stopping the stepper motors.

4. Resetting some of the board's programmable parameters.
5. Performing simple tests on the board such as clearing up the set bits.

There are basically four registers which control all the functions on the DT 2805 board. All these registers are 8-bit registers.

1. Command Register (write only).
2. Status Register (read only).
3. Data-In Register (write only).
4. Data-Out Register (read only).

1. Command Register: This is located at the base address + 1 of the DT 2805 board. Base address is the lowest I/O address at which the board can be accessed over the TI professional computer bus i.e., it is computer's I/O space where the board will be addressed. Among the 8-bits of the command register, the first four bits of 0-3 (lower byte) are called operation code bits. There are sixteen pre-defined functions on the board. These operation code bits are used to specify what functions the board should perform. This can be understood a little better with the help of Figure 5.

The upper four bits are called command modifiers. Depending on the first 4 bits the operation can be performed in DMA (Direct Memory Access) mode, continuous mode, and with an external trigger or external clock.

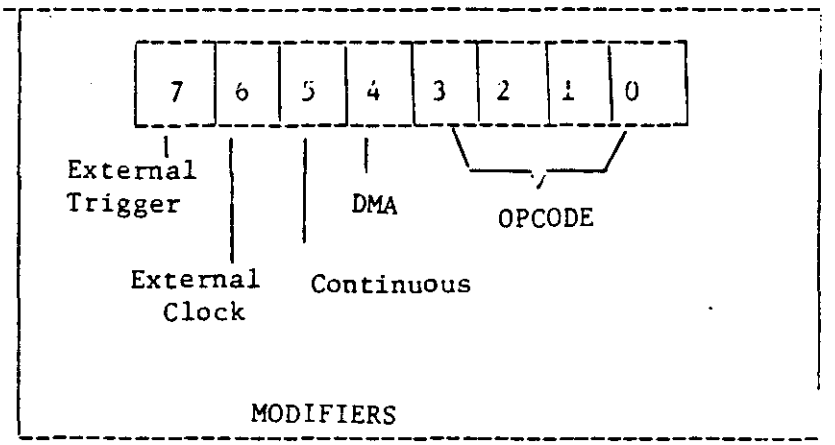


Figure 5. Command Register Bit Functions

The motors can run in different modes depending on which command modifier bit is set low or high(low or high is indicated by a 0 or 1). Figure 6 shows as to which command modifier can be used with which command. In Figure 6, "X" is a legal modifier. "0" is a illegal modifier. This bit-value must always be a "0". For instance, if the binary code "0010" is used in the OPCODE bits of the Command Register, the DT 2805 series board will "Read the Error Register".

2. Status Register: This is Read only register located at the base address + 1 of the board. It contains status byte from the board. Using the register bits as Status Flags, the current status of the board is reflected by indicating:

COMMAND	LEGAL MODIFIERS Ext Trig	Ext Clk	Cont	DMA	OPCODE (Binary)
RESET	0	0	0	0	0000
CLEAR ERROR	0	0	0	0	0001
READ ERROR REG.	0	0	0	0	0010
SET INTERNAL CLOCK PERIOD	0	0	0	0	0011
STOP	0	0	0	0	1111
TEST	X	0	0	0	1011
SET DIG. PORT FOR INPUT	X	0	0	0	0100
SET DIG. PORT FOR OUTPUT	X	0	0	0	0101
READ DIG. INPUT IMMEDIATE	X	0	0	0	0110
WRITE DIG. OUTPUT IMMEDIATE	X	0	0	0	0111
WRITE D/A IMMEDIATE	X	0	0	0	1000
SET D/A PARAMETERS	0	0	0	0	1001
WRITE D/A	X	X	X	X	1010
READ A/D IMMEDIATE	X	0	0	0	1100
SET A/D PARAMETERS	0	0	0	0	1101
READ A/D	X	X	X	X	1110

Figure 6. Command and Legal Modifiers

i) If an error has occurred,

ii) Whether a command is completed or not,

iii) Whether the last byte written to the DT 2805 board was written to the Data-In Register or to the Command Register,

iv) Whether a write to the Data-In Register can occur, and

v) Whether a read from the Data-Out Register can occur.

The Status Register bit functions are indicated below in Figure 7.

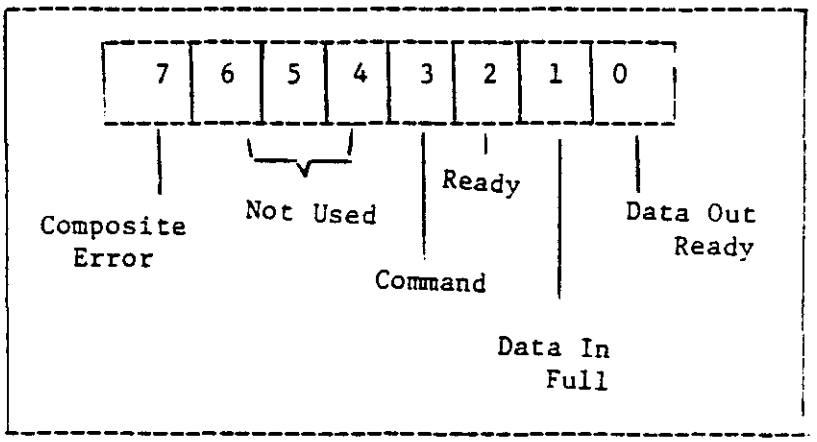


Figure 7. Status Register Bit Functions

Bit 0 is a Data-Out ready bit. when is set, it indicates that new A/D data, digital data, or error register information is present in the Data-Out register and has not

been read. Bits are set to 1 if they are 0 and setting themselves to 0 if they are 1. After the Data-Out Register is read, bit 0 clears automatically. The Data-Out ready bit always needs to be checked before reading the Data-Out Register.

Bit 1, a Data-In full bit, when set, indicates that D/A data, digital output data, or a command parameter is present in the Data-In register, or that a command byte is present in the command register, and has not been read by the boards internal circuitry.

Bit 2, a Ready bit, when set, indicates that the board has completed the previous command, and is ready to begin execution of the next command. When clear, bit 0 indicates that the board is busy executing a command. Writing to the command register while bit 2 is clear will result in a command overwrite error.

Bit 3, a command bit, when set indicates that the last byte written to the DT board was written to the command register. When clear it indicates that the last byte was written to the Data-In register.

Bit 4, 5, 6 are not used.

Bit 7 is a composite error bit, when set this indicates that an error has occurred on the board. The error bit remains set until cleared by a reset or clear error command.

3. Data-In Register: This is a write only register located at the base address of the board. It receives data written from the TI professional personal computer to the

board to perform a D/A conversion, or a digital output operation. The Data-In register also receives command parameters as a part of the operating sequence of a number of commands.

4. Data-Out Register: This is a read only register located at the board's base address. It contains data which is read from the DT 2805 board by the TI professional computer as a result of an A/D conversion, or a digital

Finally, when a command is written from the personal computer to the board , its execution is divided into three sequential events.

1. The set period, during which the various subsystems of the boards are prepared to perform the command,
2. The issuing of an internal software trigger, which starts the operating sequence opted from the list in Figure 6. In this case digital I/O commands are used.
3. The actual performance of the command. In this board the operations can be either single operation commands or block commands. All the commands in this board are single operation commands except for Read A/D and Write D/A (digital to analog) which are block commands. Single operation commands accomplish a single event when they run, and block commands accomplish multiple events when they run. Single operation commands are listed in Appendix B.

The above register functions and addresses are

indicated in Figure 8.

Now with a clear understanding of all the aspects involved in real-time control of micropositioners, Chapter VIII will highlight the sequential steps involved in writing the program for the process. As mentioned earlier, only the digital I/O lines, shown in Figure 4 are used to control

REGISTER NAME	REGISTER FUNCTION	REGISTER ADDRESS
Data-In	Receives data, command parameters from the TI professional computer	Base (Write)
Data-Out	Contains data, error information from the board	Base (Read)
Command	Receives command byte from the TI PC	Base + 1 (Write)
Status	Contains status byte from the board	Base + 1 (Read)

Figure 8. Register Functions and Addresses

the stepper motors. There are basically four commands for the digital I/O.

1. Set digital port for input.

2. Set digital port for output.
3. Read digital input immediate.
4. Write digital output immediate.

Only 'Set Digital Port for Output' and 'Write Digital Output Immediate' are used.

Inner Control Loop

Polarographic Oxygen Measurement

Oxygen microelectrodes are the most commonly used, and since it is monolayer electrode, it requires the use of an external reference electrode. Two electrodes are polarized with a potential of slightly less than 1.0 volt in a electrolytic solution containing dissolved oxygen. Current flows as a result of the reduction of oxygen at the cathodic (negatively polarized) surface. At the cathode the reaction is expressed as :

$$O_2 + 2H_2O + 4e^- \longrightarrow 4OH^-$$

At the other electrode(Ag/AgCl, reference electrode), the reaction is,

$$4Ag + 4Cl^- \longrightarrow 4AgCl + 4e^-$$

Theoretically, the voltage-current relationship for a polarographic oxygen is represented by the characteristic curve shown in Figure 9. In the region below approximately -0.5 volt, there is a reasonably linear voltage-current

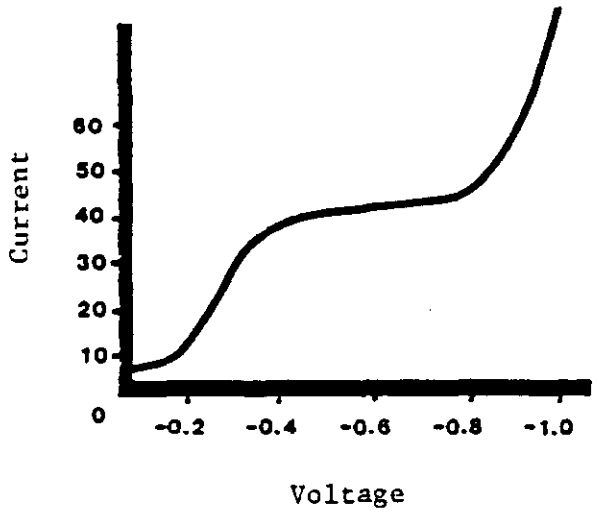


Figure 9. Characteristic Curve

relationship. As the polarization voltage is increased beyond -0.5 volt, the current tends to reach a plateau where changes in voltage have little effect on current. In this plateau region, the current is limited by the rate at which oxygen diffuses to the cathode. As the voltage is increased above -1.0 volt, the current increases again with the voltage, due to reduction of other elements in addition to oxygen.

The electrode is best operated with polarization voltage set to the mid-point of the plateau region, in which case the current is diffusion limited. In a diffusion limited condition, virtually all of the oxygen molecules which reach the cathode are immediately reduced, resulting

in a zero oxygen concentration at the cathode surface, and a current which is limited by the rate at which oxygen can diffuse to this zero concentration region. The diffusion rate is a function of the oxygen diffusion coefficient of the substance (membrane and media) surrounding the cathode and the dissolved oxygen concentration. This, in turn, is proportional to the oxygen partial pressure. For constant temperature, current flow through the electrode is directly proportional to the partial pressure of oxygen.

A plot of the relationship between the current and the partial pressure of oxygen (at a fixed polarization voltage) is called the standard curve. This is shown in figure(10). The curve is linear and it does not intersect the origin, but rather, indicates a small current at the zero partial

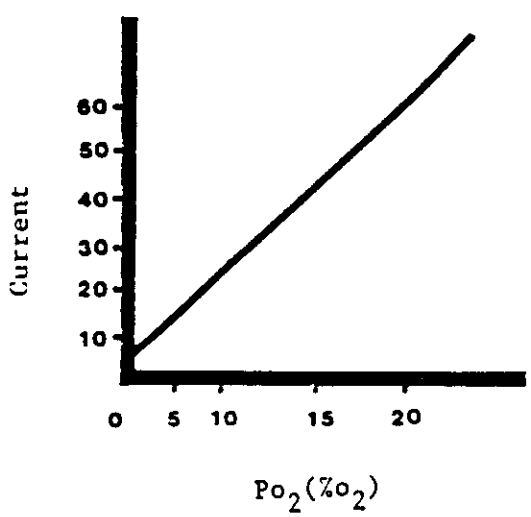


Figure 10. Standard Curve

pressure. This is the residual current and results from the factors such as electrical leakage through insulating materials in the system and reduction of oxygen which is absorbed into the electrode material.

Characteristics of Oxygen Electrodes

Since the electrode is very small, it is difficult to maintain identical characteristics from one electrode to another. For micro-sized electrodes, the characteristics are sometimes less than ideal. Wide variety of applications of these electrodes makes it difficult to optimize the system for a particular application. However a proper understanding of the electrode characteristics is important for accurate oxygen measurements. The slope of the plateau in the characteristics curve varies from electrode to electrode. It generally covers a span of 0.1 to 0.4 volt in width, with the mid-point occurring anywhere between 0.5 and 0.95 volt. A plateau is generally defined as the region of the characteristics curve which has the minimum slope and operation of the electrode is at a voltage occurring near the mid-point of this region. Since oxygen electrodes function well when polarized with a potential of 0.75 volts, it is not necessary to produce a characteristics curve. The electrode is stabilized at the operating voltage (normally 0.75) at a temperature at which measurements are made for at least two hours. It is important to take extreme care in handling the electrodes. Electrodes maintenance techniques

are located in Appendix C.

Calibration

The probes are soaked in saline solution or in distilled water before use. It is also possible to store the probes in the saline solution. Storage techniques are nighlighted by Cully (19).

Calibration is carried out at the same temperature as the measurement media. A reference electrode is used in the calibration media as well as in measurement media. Since oxygen partial pressure and current have a linear relationship, a two-point calibration suffices. However, a three-point calibration should be done to ensure the linearity.

The first step in probe calibration is to warm up the picoammeter for 20 minutes. It is also necessary to calibrate the probe in saline solution and not in distilled water since distilled water contains very few ions, and it does not conduct electricity well. The calibration is done using 21% oxygen (ambient room air) and 0% oxygen (100% nitrogen) to produce two calibration points. This is done by bubbling higher level oxygen gas through the saline solution containing the electrode for a period of 15 minutes. This will allow the solution to equilibrate with the gas. Next step is to bubble 100% nitrogen until the solution is saturated. This displaces the oxygen gas. The picoammeter still displays a small residual current called

"dark current". This small current is subtracted electronically at the amplifier later.

The reference electrode and microprobe are now placed into the aerated saline solution and a voltage slowly turned to 0.75 volts. A probe working properly will display a reading of 10^{-10} amps - 10^{-12} amps on the picoammmeter. It is important not to remove the reference electrode from the solution before the microprobe. This may send a sudden surge of current through the probe, separating the gold layer from the woods metal, thus ruining the probe. Regardless of the calibration methods chosen, electrodes do drift. So, it is important to do additional calibration checks during the experiment.

Electrodes should also be transferred as quickly as possible from the calibration media to the measurement media. If there are any temperature changes while transferring, it will take several minutes to restabilize. Polarization is temporarily stopped when electrode is removed from the solution.

Laboratory Microprobing

After the calibration is done, the microprobe is carefully lowered in the flow chamber, shown in Figure 11 until the electrical contact is made with the slime. This can be viewed on the picoammeter. The probe is moved up and down in cycles to confirm this location.

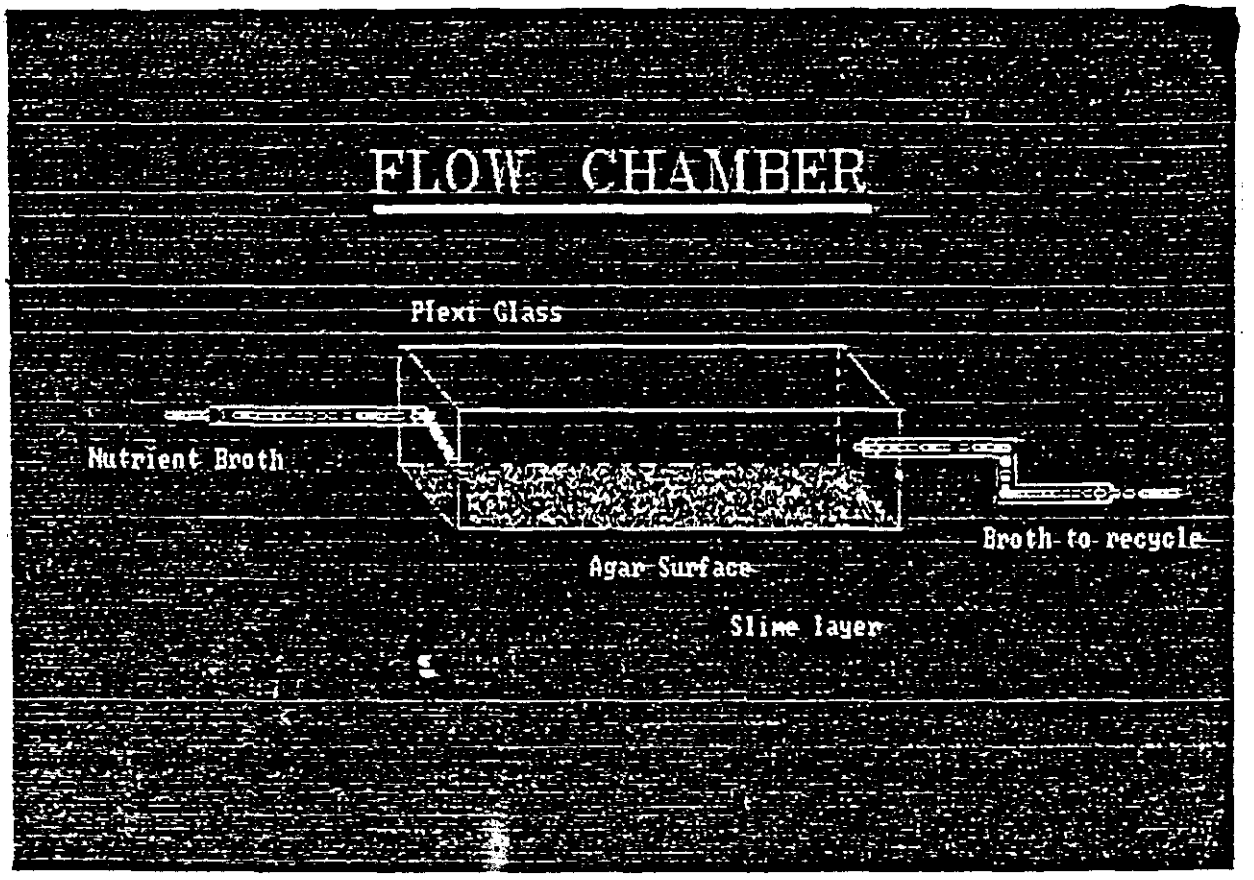


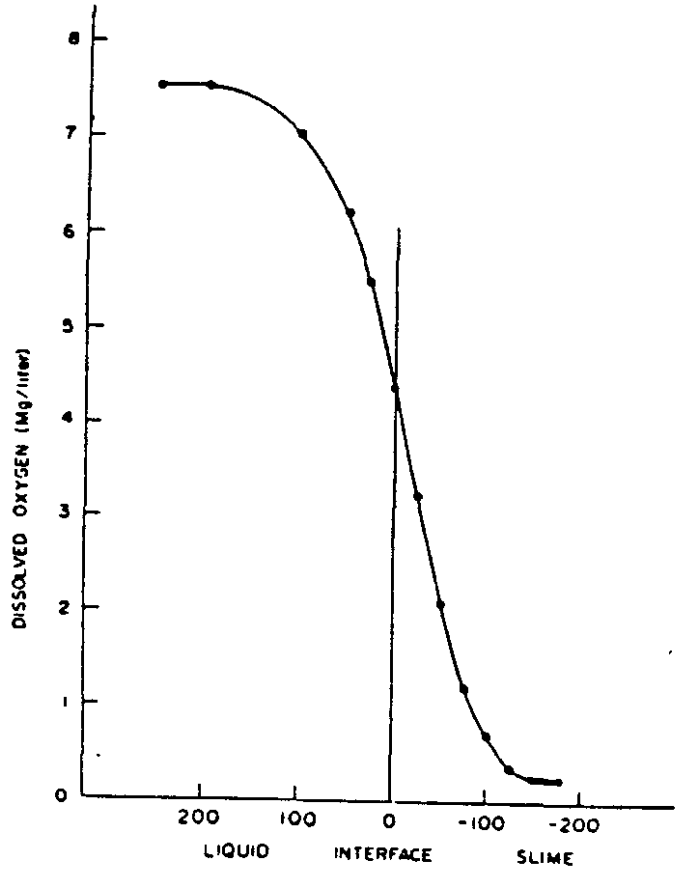
Figure 11. Flow Chamber

The flow chamber is made out of plexiglass. Slimes (Algal layers) are grown on concentrated agar surface in the flow chamber, since this kind of surface minimizes the chance of microprobe breakage. Nutrient broth or Algal-gro concentrate is used to grow the algae. Since this is a continuous media, it is necessary to maintain a low velocity of the flowing media to reduce the shear on the algae surface, and efficiently utilize the nutrient within the medium. The chamber is raised slightly at the inlet end to reduce pooling of the liquid and to maintain flow across the surface.

The microprobe and silver/silver chloride reference are connected to a voltage source, picoammeter and micropositioners, which in turn are connected to the TI personal computer for data acquisition and control. This is illustrated in Figure 3. As in the calibration stage, a voltage of 0.75 volts is applied across the microelectrode and the reference electrode. At this stage, the molecular oxygen breaks into ions which move in the electrical field created. A picoammeter used in conjunction with the microcomputer records the changes in the current. Oxygen activity is linear with the current flow. The microprobe is positioned with a micromanipulator which allows 3-axis movement with a precision of 1 micron. The positioners have a limit switch which enables them to stop after a maximum movement of 3cm. This very important function of the manipulators is incorporated in the computer program. It

prevents the microprobe to reach the bottom of the studying media and thus, avoid its breakage.

The Ag/Agcl reference electrode is approximately 7cm in length. It is very important that the microelectrode set be well shielded and grounded to minimize any electrical interference. Co-axial low noise cable is used and is taped down to minimize vibrations that create stray signals.



Source: Whalen et al, "Microelectrode Determination of Oxygen Profiles in Microbial Systems", Environmental Science and Technology (1969)

Figure 12. Microprobe Measurements of Dissolved Oxygen Concentrations.

A typical profile of oxygen concentration with respect to the distance in the slime using air saturated substrate is shown in Figure 12. This study done by Bungay, Sanders and Whalen (10) involved lowering of the probe at 25 microns increments at 30 second interval each. Such studies demonstrate the applicability of microelectrodes in direct determination of oxygen concentration, and also the oxygen and nutrient diffusivity coeffecients for the microbial system. Mass transfer cosiderations are provided in Appendix D.

CHAPTER VIII

OVERVIEW OF THE MICROPOSITIONERS PROGRAM

The program is written both in C and BASIC languages. The program is listed in Appendix E.

The micropositioners are run and controlled by writing command bytes to the boards Command Register, and by writing data bytes and command parameters bytes to the boards Data-In Register.

The following points are important for programing the board.

Before each read or write of the Command or Data Registers, the Status Register must be checked. The Status Register can be read at any time.

Before starting a command, the Status Register must indicate that the board is ready for a new command and a check must be made to see that the Ready bit, bit 2 of the Status Register, is set.

Data must not at any time be written to the Data-In Register unless the Data-In full bit is clear, indicating that the board is ready for data input. The Stop command is a special case, and can be written to the Command Register any time, without checking the Status Register and regardless of the state of the board.

Valid data cannot be read from the Data-Out Register unless the data out ready bit is set. The board will not return to the ready state if data remains in the Data-Out Register.

Thus considering all the points mentioned above, the actual program is written in two parts. The first part sets up the digital ports for output, and the second part writes digital output immediate commands to the board. Each part involves a sequence of operation. These operating sequences are detailed below:

SET DIGITAL PORT FOR OUTPUT

1. Check Status Register and write command to Command Register.
2. Check Status Register and write Digital Port Select to Data-In Register.
3. If no external trigger, the board issues a software trigger.
4. If external trigger:
   A. Wait at least 1 ms.
   B. Issue external trigger.
5. If Digital Port Select equals 0 or 1:
   A. DIO Port 0 or 1 is set to provide digital outputs.
6. If digital Port Select equals 2:
   A. DIO ports 0 and 1 are set to provide digital outputs.

7. Perform a WRITE DIGITAL OUTPUT IMMEDIATE operating sequence, specifying a data byte value of 0 to clear the newly enabled output port.

WRITE DIGITAL OUTPUT IMMEDIATE

1. Check Status Register and perform a SET DIGITAL PORT FOR OUTPUT command to set the port or ports used for output.
2. Check Status Register and write command to Command Register.
3. Check Status Register and write Digital Port Select to the Data-In Register.
4. Check Status Register and data for DIO port 0 or 1 to Data-In Register.
5. If Digital Port Select equals 2:
   A. Write data for DIO port 1 to Data-In Register.
6. If no external trigger, the board issues a software trigger.
7. If external trigger:
   A. Wait at least 1 ms.
   B. Issue external trigger.
8. If Digital Port Select equals 0 or 1:
   A. Check Status Register and write data to Data-In Register.
9. If Digital Port Select equals 2:
   A. Check Status Register and write data for Port 0 to Data-In register.

B. Check Status Register and write data for Port 1 to Data-In Register.

CHAPTER IX

CONCLUSIONS AND RECOMMENDATIONS

CONCLUSIONS

1. A real-time computer controlled positioning system has been designed and built, using the state of the art micropositioners. The system controlled by Texas Instrument's professional computer has been tested and works satisfactorily with respect to the micro movement of the probe.

2. A software package has been developed in C and BASIC languages. The package is easy to use and may be conveniently combined with other available packages to expand the current capabilities of the system.

3. C language is sophisticated as well as convenient in data acquisition and control aspects of the system. Basic's Input/Output functions have proved to be very useful.

RECOMMENDATIONS

1. The developed computer package, when used, prompts the user to provide an external trigger to move the position of the microprobes. This trigger may be provided through another software package. Such a feature is desirable to

make micropositioning of the probe a function of the trend analysis of the measurements. For instance, the position of the microprobe can be made a function of the oxygen concentration with time. This would be the basis for automatic feedback to control the amount of oxygen supplied to a biological or a chemical system.

2. All the equipment used in building the system is designed for IBM personal computers only. However, they are controlled by a TI personal computer, which is not completely compatible with parts designed for IBM machines. This was accomplished by emulating the TI computer to work like an IBM computer. Most software packages available in the market are IBM compatible and may not work even if TI is emulated as an IBM machine. Consequently, controlling the whole system with an IBM computer would be more advantageous in the long run.

3. If an IBM computer is eventually installed in place of the TI, care should be taken to ensure that I/O addresses of the measuring and the sensing systems do not interfere with the I/O mapping of the IBM personal computer.

CHAPTER X

SUGGESTED APPLICATIONS

1. For delicate and accurate positioning of any sensing device for a particular measurement.

2. Study of the fundamental mechanism of the growth of micro-organisms. This will better define the kinetics of growth and metabolism of slime organisms in water bodies and waste water systems.

3. Study of corrosion mechanisms and developing rapid methods for identifying microbiologically innfluenced corrosion. On-line detection and trend analysis of the microbiological and chemical systems will help.

4. Measureing the effect of fouling in Heat Exchangers.

5. Measureing the cell activity as a function of their immediate environment.

6. Precise measurements of the interfacial mass transfer can be made with ultra microprobes. This information would be valuable in improving equipment design and performance.

7. Considerable insight into mass tranfer to and within microbial slime films can be obtained from steady state and from dynamic measurements of dissolved oxygen.

8. The dynamic method presented here, for measuring oxygen transfer coefficients, has a potential to provide more consistent results.

9. This technique adds one more dimension to the study of turbulent mass transfer in addition to conventional methods such as hot film or wire anemometry, interferometry, and holography.

10. This technique can also be applied to investigate the surface region of the liquid, which has been difficult so far.

11. Many different models can be checked in this way and, above all, an interfacial mass transfer mechanism can be elucidated which can be directly used in improving the performance of existing two-phase contactors or designing new mass transfer devices.

12. Unsteady state measurements are real difficult to make and a real time operation has a potential to make point concentration measurements in real time.

A SELECTED BIBLIOGRAPHY

1. Khuri, R. N. Ion-Selective electrodes in biomedical research In: Ion-Selective electrodes, R. A. Durst (editor) National Bureau of Standards Special Publication 314 November (1969).

2. Kim, N. K. and D. W. Stone. Organic Chemicals and Drinking Water. New York State Department of Health (undated).

3. Dowben, R. M. and J. E. Rose. A Metal Filled Microelectrode, Science 118: 22-24 (1953).

4. Whalen, W. J., J. Riley and P. Nair. A Microelectrode for Measuring Intracellular pO_2, Journal Applied Physiology, 23, 798 (1967).

5. Whalen, W. J., P. Nair and R. A. Ganfield. Measurements of Oxygen Tension in Tissues with a Micro Oxygen Electrode, Microvascular Research 5: 254-262 (1973).

6. Berman, H. and M. Herbert (editors). Ion-Selective Microelectrode, Proceeding of a Workshop on Ion Selective Microelectrodes, Plenum Press, New York (1974).

7. Spande, J. I., W. J. Whalen and D. Buerk. Flow Through pO_2 Sensor, JEPT, vol. 7, No. 1: 4-9 (1980).

8. Revsbech, N., J. Sorensen, T. Blackburn and J. Lomholt. Distribution of Oxygen in Marine Sediments Measured with Microelectrodes, Limnology and Oceanography 25(3): 403-411 (1980).

9. Revsbech, N., B. Jorgensen and T. Blackburn. Oxygen in the Sea Bottom Measured with a Microelectrode, Science 207: 1355-1356 (1980).

10. Whalen, W. J., H. R. Bungay III and W. M. Sanders III. Microelectrode Determination of Oxygen Profiles in Microbial Slime Systems, Environmental Science and Technology 3: 1297 (1969).

11. Bungay III, H. R., W. J. Whalen and W. M. Sanders III. Microprobe Techniques for determining Diffusivities and Respiration Rates in Microbial Slime Systems,

Biotechnology and Bioengineering 11: 765 (1965).

12. Bungay III, H. R., W. M. Sanders III and W. J. Whalen. Oxygen Transfer at the Microscopic level. 160th National A.C.S Meeting, Division of Microbial Chemistry and Technology, Chicago (1970).

13. Chen, Y. S. Microelectrode Studies of Oxygen Transfer in Microbial Slime, PH. D Thesis, R.P.I. (1979).

14. Chen, Y. S. and H. R. Bungay. Microelectrode Studies of Oxygen Transfer in Trickling Filter Slimes Manuscript for American Chemical Society Meeting, Washington, D.C. (1979).

15. Bungay III, H. R. and Y. S. Chen. Oxygen Transfer in Photosynthetic Slimes. Manuscript for American Chemical Society National Meeting, Miami (1978).

16. Drislane, A. M. and H. R. Bungay. Microelectrode Measurements of Oxygen Profiles in Activated Sludge Flocs, not published yet (1982).

17. Huang, M. Y. and H. R. Bungay III. Microprobe Measurement of Oxygen Concentration in Mycelail Pellets. Biotechnology and Bioengineering 15: 1183 (1973).

18. Cully, D. T. Masters Thesis Work, R.P.I. New York, (1982).

19. Bungay III, H. R. and D. M. Harold, Jr.. Simulation of Oxygen Transfer in Microbial Slimes. Biotechnology and Bioengineering. 13: 569(1971).

20. Bicher, H. I. and M. H. Kinsley. Brain Tissue Reoxygenation Time, Demonstrated With a New Ultramicro Oxygen Electrode. Journal of Applied Physiology 28: 387(1970).

APPENDIXES

APPENDIX A

IEEE 488 CONTROLLER DRIVERS

IEEE 488 controller drivers and files contained in Ziatech's disk for ZT 1444 GPIB and multifunction I/O board are listed as follows. Also a program called XXTEST.exe interactively gives control of the bus to the computer. All the function modules listed below are incorporated in this program.

ZT 1444 GPIB Basic and C Language Drivers

Driver Name	Function
bustat	Get bus status.
cmd	Send command.
devclr	Device clear.
doc	Get software revision.
eoi	Supress EOI output.
init	Initialize the bus.
ioc1	Set local state.
iokout	Lock out state.
ppoll	Perform a parallel poll.
ppold	Parallel poll disable.
ppollu	Parallel poll unconfigure.

recvdm	Receive data with DMA.
recvst	Receive data.
remote	Remote the bus.
senddm	Send data with DMA.
sendst	Send data.
setaddr	Set I/O port address.
setpri	Select primary port.
setsec	Select secondary port.
spoll	Serial poll.
srqsta	Get SRQ status.
term	Set terminating char.
timedy	Set timeout delay.
trig	Execute trigger.
xfer	Transfer between devices.
example	Example program.
alarm	Set alarm.
instat	Get clock status.
loadc	Load counter.
readc	Read counter.
readl	Read latch.
synch	Synchronize.
systic	System tic.

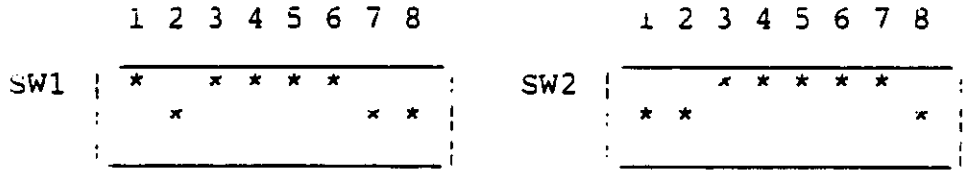
The Following highlights the installed IEEE board's capabilities.

- IEEE bus is a listener, talker and controller.
- Can control upto 15 other IEEE-488 compatible

devices.

- Each controller occupies one Input/Output slot in the TI PC.
- Fully compatible with IEEE-488 1978 standards.
- DMA channel user selectable.
- Eight Input/Output port addresses.
- Interrupt enabling and disabling.
- System controller enabling and disabling.
- No jumper changes need be made when used.

The board has been installed with the following Input/Output address dip switch configuration.



APPENDIX B

SINGLE OPERATION COMMANDS

1. Reset.
2. Clear Error.
3. Read Error Register.
4. Set Internal Clock.
5. Stop.
6. Test.
7. Set Digital Port for Input.
8. Set digital port for Output.
9. Read digital Input Immediate.
10. Write Digital Output Immediate.
11. Write D/A Immediate.
12. Set D/A Parameters.
13. Read A/D Immediate.
14. Set A/D Parameters.

APPENDIX C

MICROPROBE MAINTENANCE

A. Storage: Microelectrodes should either be stored in saline solution or in a dust free container. Care should be taken to see that the tips are not touching any surface. Contact with any surface can result in microprobe breakage.

B. Cleaning: 723 oxygen microelectrodes can not be used indefinitely without some biomass build-up or contamination in the tip. The probe can be used a number of times if proper care is taken. After use, the tip of the electrode is submerged in de-ionized water for 15-30 minutes. Following this, there are three more steps for cleaning the probes.

1. Put a drop of water on a kimwipe.
2. Hold the wet spot of the tissue folded between the thumb and the index finger.
3. Pull the electrode through these fingers, being very careful not to snap the tip by bending the probe.

Fragility of the Probes

The main disadvantage of using the microprobe is the fragility of the instrument. Extreme care is necessary in

handling the electrodes, as mechanical contact with materials can easily break the tip. When bubbling gases through test solutions to calibrate the electrode, it is advantageous to make certain that the tip is not directly in the flow of bubbles, as this can create breakage. Other factors that can cause breakage are,

- continuous use over extended periods of time
- sneezing or sudden movement of the probe in hand
- shipping hazards
- probing too far in the test medium and hitting the hard bottom surface.

APPENDIX D

THE DIFFUSION EQUATIONS

The diffusion equations are presented here. Consider an element of slime parallel to the slime-medium interface. The mass transfer equations for the system can be written for rectangular coordinates as follows,

$$\frac{\delta c}{\delta t} + U_x \frac{\delta c}{\delta x} + U_y \frac{\delta c}{\delta y} + U_z \frac{\delta c}{\delta z} = D \left[\frac{\delta^2 c}{\delta x^2} + \frac{\delta^2 c}{\delta y^2} + \frac{\delta^2 c}{\delta z^2} \right] + R \qquad (D-1)$$

neglecting the influence of velocity terms, Equation (D-1) becomes,

$$\frac{\delta c}{\delta t} = D \left[\frac{\delta^2 c}{\delta x^2} + \frac{\delta^2 c}{\delta y^2} + \frac{\delta^2 c}{\delta z^2} \right] + R \qquad (D-2)$$

where, R = rate of generation.

But in the present system, oxygen is consumed in the system, hence R is negative (-R).

$$\frac{\delta c}{\delta t} = D \left[\frac{\delta^2 c}{\delta x^2} + \frac{\delta^2 c}{\delta y^2} + \frac{\delta^2 c}{\delta z^2} \right] - R \qquad (D-3)$$

Neglecting the diffusion in Y and Z directions, Equation (D-3) can be written as,

$$\frac{\delta c}{\delta t} = D \left[\frac{\delta^2 c}{\delta x^2} \right] - R \qquad (D-4)$$

Since inside film is in steady state,

$$\frac{\delta c}{\delta t} = 0 \qquad \text{(D-5)}$$

therefore,

$$D \left[\frac{\delta^2 c}{\delta x^2} \right] = R \qquad \text{(D-6)}$$

Now diffusivity D, can be assumed to be constant. Then, partial equation becomes total differential equation and Equation (D-6) becomes,

$$\frac{d^2 c}{dx^2} = \frac{R}{D} \qquad \text{(D-7)}$$

Integrating Equation (D-7)

$$\frac{dc}{dx} = \frac{R}{D} x + K \qquad \text{(D-8)}$$

where K = constant of integration.

Now D can be found by plotting dc/dx versus X. It will be a straight line as suggested by Equation (D-8). Hence, the slope of the straight line will be R/D. Knowing R, the rate of oxygen uptake, D can be found from the slope.

The local mass transfer coefficient can be obtained from the following equation,

$$(K_L)_{local} = \left[- D \left(\frac{\delta c}{\delta x}\right)_{x=0} \right]_{local} / \left(C_i - C_b \right) \qquad \text{(D-9)}$$

where C_i is the concentration at the interface and C_b is the bulk concentration.

APPENDIX E

LISTING OF PROGRAMS

SET DIO FOR OUTPUT

```
10    '' A Program to set DIO for output.
20    '' ----------------------------------
30    ''
40    ''  This is a program to run the micropositioners. As this program
50    ''  sets the DIO for output, it needs to be run before the
60    ''  WRITE DIGITAL PORT IMMEDIATE program which follows.
70    ''
80    ''       This program is written in BASIC language. It can be used
90    ''  to move the micropositioners with a precision of 1 micron in
100   ''  any of the three directions--X, Y or Z. Each single step
110   ''  corresponds to 1 micron.
120   ''
130   ''       The program assumes that there is a Data Translation's
140   ''  2801 series board installed at the Base address of &H2EC.
150   ''  A different Base address can be chosen, but since this is a
160   ''  location in the TI professional computer's I/O space where the
170   ''  board is addressed, care should be taken to see that this address
180   ''  does not interfere with the I/O mapping of the TI computer.
190   ''
200   ''        The user is asked whether DIO port 0, DIO port 1 or DIO
210   ''  port 2 is to be written (note that port 2 is a way to specify
220   ''  both port 1 and port 0).
230   ''
240   ''         Programming principles are commented preceding each step.
250   ''  This serves the purpose of documentation for any further changes
260   ''
270     CLS : PRINT
280     PRINT" A PROGRAM TO SET DIO FOR OUTPUT
290     PRINT" ------------------------------
300   ''
310     PRINT" THIS PROGRAM ASSUMES THAT A DT 2801 SERIES BOARD IS INSTALLED.
320     PRINT"  ------------------------------------------------------
330   ''
340     PRINT" Please make sure to run this program before you run the
350     PRINT" WRITE DIGITAL OUTPUT IMMEDIATE PROGRAM.
360     PRINT : PRINT
370   ''
380   '' Define constants and variables: Definitions are used to specify
```

```
390 '' the addresses of the Command Register, the Status Register, the
400 '' Data_In Register and the Data-Out Register. Lines 510 530
410 '' define Hex values which when used with a 'WAIT" command,
420 '' indicate whether a particular bit in the Status Register is set or
430 '' cleared. Cstop is defined as the Hex value F. This is the Command
440 '' byte value for stop. Similarly, other Hex values are used to define
450 ''
460   DEFINT A-Z
470   BASE.ADDRESS      = &H2EC
480   COMMAND.REGISTER  = BASE.ADDRESS + 1
490   STATUS.REGISTER   = BASE.ADDRESS + 1
500   DATA.REGISTER     = BASE.ADDRESS
510   COMMAND.WAIT      = &H4
520   WRITE.WAIT        = &H2
530   READ.WAIT         = &H5
540 ''
550   CCLEAR            = &H1
560   CERROR            = &H2
570   CSOUT             = &H5
580   CSTOP             = &HF
590   EXT.TRIGGER       = &H80
600   MENU$             = "EP00.BAS"
610 ''
620 '' Check for legal Status Register.
630 ''  ------------------------------
640 ''
650 ''
660   STATUS = INP(STATUS.REGISTER)
670   IF NOT((STATUS AND &H70) = 0) THEN GOTO 1650
680 ''
690 '' Stop and clear the DT2801.
700 ''  ------------------------
710 ''
720   OUT  COMMAND.REGISTER, CSTOP
730   TEMP = INP(DATA.REGISTER)
740   WAIT STATUS.REGISTER,  COMMAND.WAIT
750   OUT  COMMAND.REGISTER, CCLEAR
760 ''
770   INPUT "         Set DIO Port 0,1 or 2 for OUTPUT";DIOPORT
780   IF DIOPORT >= 0 AND DIOPORT =< 2 THEN GOTO 820
790 ''
800   PRINT : PRINT "         Please respond with 0, 1 or 2 only."
810   GOTO 760
820 ''
830   PRINT "         Are you sure you want to set DIO Port ";DIOPORT;
840   INPUT " for OUTPUT";Y$
850   IF Y$ = "Y" OR Y$ = "y" THEN GOTO 900
860   IF Y$ = "N" OR Y$ = "n" THEN GOTO 1740
870 ''
880   PRINT : PRINT "         Please respond with 'Y' or 'N' only."
890   GOTO 820
900 ''
910   PRINT
```

```
920    INPUT "         Wait for External Trigger (Y/N)";Y$
930    IF Y$ = "Y" OR Y$ = "y" THEN GOTO 970
940    IF Y$ = "N" OR Y$ = "n" THEN GOTO 1170
950    PRINT : PRINT "         Please respond with 'Y' or 'N'."
960    GOTO 900
970  ''
980  '' Write SET DIGITAL PORT FOR OUTPUT WITH TRIG command.
990  '' ------------------------------------------------
1000 ''
1010   WAIT STATUS.REGISTER,  COMMAND.WAIT
1020   OUT  COMMAND.REGISTER, CSOUT + EXT.TRIGGER
1030 ''
1040 '' Write DIGITAL PORT SELECT byte.
1050 '' --------------------------------
1060 ''
1070   WAIT STATUS.REGISTER,  WRITE.WAIT, WRITE.WAIT
1080   OUT  DATA.REGISTER,    DIOPORT
1090 ''
1100 '' Wait for EXTERNAL TRIGGER.
1110 '' --------------------------
1120 ''
1130   PRINT : PRINT "         Waiting for EXTERNAL TRIGGER."
1140   WAIT STATUS.REGISTER,  COMMAND.WAIT
1150   PRINT "         EXTERNAL TRIGGER Received."
1160   PRINT : GOTO 1290
1170 ''
1180 '' Write SET DIGITAL PORT FOR OUTPUT command.
1190 '' ------------------------------------
1200 ''
1210   WAIT STATUS.REGISTER,  COMMAND.WAIT
1220   OUT  COMMAND.REGISTER, CSOUT
1230 ''
1240 '' Write DIGITAL PORT SELECT byte.
1250 '' -------------------------------
1260 ''
1270   WAIT STATUS.REGISTER,  WRITE.WAIT, WRITE.WAIT
1280   OUT  DATA.REGISTER,    DIOPORT
1290 ''
1300 '' Check for ERROR.
1310 '' ---------------
1320 ''
1330   WAIT STATUS.REGISTER,  COMMAND.WAIT
1340   STATUS = INP(STATUS.REGISTER)
1350   IF (STATUS AND &H80) THEN GOTO 1400
1360 ''
1370   PRINT
1380   PRINT "         SET DIO FOR OUTPUT Operation Complete"
1390   GOTO 1740
1400 ''
1410 '' Fatal board error.
1420 ''
1430   PRINT
1440   PRINT "FATAL BOARD ERROR"
```

```
1450    PRINT "STATUS REGISTER VALUE IS ";HEX$(STATUS);" HEXIDECIMAL"
1460    PRINT : BEEP : BEEP : GOSUB 1510
1470    PRINT "ERROR REGISTER VALUES ARE:"
1480    PRINT "       BYTE 1 -   ";HEX$(ERROR1);"  HEXIDECIMAL"
1490    PRINT "       BYTE 2 -   ";HEX$(ERROR2);"  HEXIDECIMAL"
1500    PRINT : GOTO 1740
1510 ''
1520 '' Read the Error Register.
1530 '' --------------------------
1540 ''
1550    OUT  COMMAND.REGISTER, CSTOP : TEMP = INP(DATA.REGISTER)
1560 ''
1570    WAIT STATUS.REGISTER, COMMAND.WAIT
1580    OUT COMMAND.REGISTER, CERROR
1590 ''
1600    WAIT STATUS.REGISTER, READ.WAIT
1610    ERROR1 = INP(DATA.REGISTER)
1620    WAIT STATUS.REGISTER, READ.WAIT
1630    ERROR2 = INP(DATA.REGISTER)
1640    RETURN
1650 ''
1660 '' Illegal Status Register.
1670 '' ------------------------
1680 ''
1690    PRINT
1700    PRINT "FATAL ERROR - ILLEGAL STATUS REGISTER VALUE"
1710    PRINT "STATUS REGISTER VALUE IS ";HEX$(STATUS);" HEXIDECIMAL"
1720    BEEP : BEEP
1730 ''
1740    PRINT : PRINT
1750 ''
1760   INPUT "          Run program again (Y/N)";Y$
1770   IF Y$ = "Y" OR Y$ = "y" THEN RUN
1780   IF Y$ = "N" OR Y$ = "n" THEN GOTO 1820
1790 ''
1800   PRINT : PRINT "          Please respond with 'Y' or 'N'."
1810   GOTO 1750
1820 ''
1830   INPUT "          Return to MENU (Y/N)";Y$
1840   IF Y$ = "Y" OR Y$ = "y" THEN RUN MENU$
1850   IF Y$ = "N" OR Y$ = "n" THEN GOTO 1890
1860 ''
1870   PRINT : PRINT "          Please respond with 'Y' or 'N'."
1880   GOTO 1820
1890   END
```

WRITE DIGITAL OUTPUT IMMEDIATE

```
10    '' WRITE DIGITAL OUTPUT IMMEDIATE program
20    '' ------------------------------------
30    ''       Initial documentation of this program is same as that for
40    '' the SET DIO for OUTPUT program. Same definitions are used.
50    '' After setting the DIO port for output, this program is run to
60    '' actually move the micropositioners with a precision of 1 micron.
70    ''  The program prompts for the number of steps to be moved in the desired
80    '' direction. Delay is the time between each step. The smaller this
90    '' number is, the faster the positioners move. Please make note that
100   '' a very short delay can cause the stepper motors to skip a couple
110   '' steps and lose the precision. The recommended delay for the
120   '' compiler version of the program is 85.
130  ''
140     CLS
150     PRINT
160    PRINT "THIS PROGRAM MOVES THE MICROPOSITIONERS WITH THE PRECISION
170    PRINT " OF ONE MICRON.
180    PRINT " ------------------------------------------------------
190    PRINT " ------------------------------------------------------
200    PRINT "BEFORE YOU RUN THIS PROGRAM PLEASE MAKE SURE TO SET
210    PRINT " THE DIGITAL PORT FOR OUTPUT
220    PRINT "------------------------------------------------------
230     PRINT
240 ''
250    DEFINT A-Z
260    BASE.ADDRESS      = &H2EC
270    COMMAND.REGISTER  = BASE.ADDRESS + 1
280    STATUS.REGISTER   = BASE.ADDRESS + 1
290    DATA.REGISTER     = BASE.ADDRESS
300    COMMAND.WAIT      = &H4
310    WRITE.WAIT        = &H2
320    READ.WAIT         = &H5
330  ''
340    CCLEAR            = &H1
350    CERROR            = &H2
360    CD100UT           = &H7
370    CSTOP             = &HF
380    EXT.TRIGGER       = &H80
390    MENU$             = "EP00.BAS"
400 ''
410 '' Check for legal Status Register.
420 ''  ------------------------------
430 ''
440    STATUS = INP(STATUS.REGISTER)
```

```
450    IF NOT((STATUS AND &H70) = 0) THEN GOTO 2850
460 ''
470 '' Stop and clear the DT2801.
480 ''
490    OUT  COMMAND.REGISTER, CSTOP
500    TEMP = INP(DATA.REGISTER)
510    WAIT STATUS.REGISTER,  COMMAND.WAIT
520    OUT  COMMAND.REGISTER, CCLEAR
530 DIOPORT  = 0
540    DIO.DATA0 = 255
550 GOSUB 1610    ' write value to I/O port
560 ''
570 ''-------------------------------------------------------------
580 ''    S T A R T    T H E    M A I N    P R O G R A M
590 ''-------------------------------------------------------------
600 INPUT " # of steps " ,NUMSTEPS
610 INPUT "delay ",IDELAY
620 INPUT " foward or reverse " , FRV$
630 IF FRV$ = "F" OR FRV$ = "f" THEN  GOSUB 3100
640 IF FRV$ = "R" OR FRV$= "r" THEN  GOSUB 3220
650 GOTO 580
660 ''
670    INPUT "         Write DIO Port 0,1 or 2";DIOPORT
680    IF DIOPORT >= 0 AND DIOPORT =< 2 THEN GOTO 720
690 ''
700    PRINT : PRINT "         Please respond with 0, 1 or 2 only."
710    GOTO 660
720 ''
730    PRINT
740    PRINT "         Legal data values are in decimal, 0 through ";
750 ''
760    IF DIOPORT < 2 THEN PRINT "255."
770    IF DIOPORT = 2 THEN PRINT "65535."
780    PRINT "         Data value to write to DIO Port ";
790    PRINT DIOPORT;
800    INPUT DATA.VALUE#
810    DATA.VALUE# = INT(DATA.VALUE#)
820 ''
830    IF DATA.VALUE# <    0 THEN GOTO 870
840    IF (DATA.VALUE# > 255 AND DIOPORT < 2) THEN GOTO 870
850    IF DATA.VALUE# > 65535! THEN GOTO 870
860    GOTO 900
870 ''
880    PRINT : PRINT "         Please use legal value."
890    GOTO 720
900 ''
910 '' Decide which port values to print out for user check.
920 ''
930    IF DIOPORT = 0 THEN GOTO 960
940    IF DIOPORT = 1 THEN GOTO 1050
950    IF DIOPORT = 2 THEN GOTO 1140
960 ''
970 '' Print out data value, port 0.
```

```
980 ''
990    PRINT
1000    DIO.DATA0 = DATA.VALUE# : GOSUB 2020
1010    PRINT "         Port 0 value = ";DIO.DATA0;" Decimal, ";
1020    PRINT HEX$(DIO.DATA0);" Hexidecimal, ";
1030    PRINT BINARY0$;" Binary."
1040    PRINT : GOTO 1330
1050 ''
1060 '' Print out data value, port 1.
1070 ''
1080    PRINT : DIO.DATA1 = DATA.VALUE#
1090    DIO.DATA0 = DATA.VALUE# : GOSUB 2080
1100    PRINT "         Port 1 value = ";DIO.DATA1;" Decimal, ";
1110    PRINT HEX$(DIO.DATA1);" Hexidecimal, ";
1120    PRINT BINARY1$;" Binary."
1130    PRINT : GOTO 1330
1140 ''
1150 '' Print out data value, port 2.
1160 ''
1170    PRINT : DIO.DATA1 = INT(DATA.VALUE#/256)
1180    DIO.DATA0 = DATA.VALUE# - DIO.DATA1 * 256
1190    GOSUB 2020 : GOSUB 2080
1200 ''
1210    PRINT "         Port 0 value = ";DIO.DATA0;" Decimal, ";
1220    PRINT HEX$(DIO.DATA0);" Hexidecimal, ";
1230    PRINT BINARY0$;" Binary."
1240 ''
1250    PRINT "         Port 1 value = ";DIO.DATA1;" Decimal, ";
1260    PRINT HEX$(DIO.DATA1);" Hexidecimal, ";
1270    PRINT BINARY1$;" Binary."
1280 ''
1290    DIO.DATA2# = DIO.DATA1 * 256 + DIO.DATA0
1300    PRINT "         Port 2 value = ";DIO.DATA2#;" Decimal, ";
1310    PRINT HEX$(DIO.DATA2#);" Hex, ";
1320    PRINT BINARY1$;" - ";BINARY0$;" Binary."
1330 ''
1340 '' Check these values with user.
1350 ''
1360    PRINT "         Are these the correct values to write to ";
1370    PRINT "digital port ";DIOPORT;" (Y/N)";
1380    INPUT Y$
1390 ''
1400    IF Y$ = "Y" OR Y$ = "y" THEN GOTO 1450
1410    IF Y$ = "N" OR Y$ = "n" THEN GOTO 2930
1420 ''
1430    PRINT : PRINT "         Please respond with 'Y' or 'N' only."
1440    GOTO 900
1450 ''
1460 '' Ask user to make external trigger decision.
1470 ''
1480    PRINT
1490    INPUT "         Wait for External Trigger (Y/N)";Y$
1500    IF Y$ = "Y" OR Y$ = "y" THEN GOTO 1540
```

```
1510    IF Y$ = "N" OR Y$ = "n" THEN GOTO 1540
1520    PRINT : PRINT "         Please respond with 'Y' or 'N'."
1530    GOTO 1490
1540 ''
1550 '' Set up command for EXTERNAL TRIGGER.
1560 ''  ---------------------------------
1570 ''
1580    IF Y$ = "Y" OR Y$ = "y" THEN COMMAND = EXT.TRIGGER
1590    IF Y$ = "N" OR Y$ = "n" THEN COMMAND = 0
1600 ''-------------------------------------------------------------------
1610 '' R O U T I N E   F O R   O U T P U T   I M M E D I A T E
1620 ''-------------------------------------------------------------------
1630 '' Write WRITE DIGITAL OUTPUT IMMEDIATE.
1640 ''
1650    WAIT STATUS.REGISTER,  COMMAND.WAIT
1660    OUT  COMMAND.REGISTER, CDIOOUT + COMMAND
1670 ''
1680 '' Write DIGITAL PORT SELECT byte.
1690 ''
1700    WAIT STATUS.REGISTER,  WRITE.WAIT, WRITE.WAIT
1710    OUT  DATA.REGISTER,    DIOPORT
1720 ''
1730 '' Write the first data byte.
1740 ''
1750    WAIT STATUS.REGISTER,  WRITE.WAIT, WRITE.WAIT
1760    OUT  DATA.REGISTER,    DIO.DATA0
1770    RETURN : END
1780 ''------------------------ ROUTINE ENDS-----------------------
1790 '' If Port 2, write second data byte.
1800 ''
1810    IF NOT(DIOPORT = 2) THEN GOTO 1840
1820    WAIT STATUS.REGISTER,  WRITE.WAIT, WRITE.WAIT
1830    OUT  DATA.REGISTER,    DIO.DATA1
1840 ''
1850 '' Wait for EXTERNAL TRIGGER.
1860 ''
1870    IF Y$ = "N" OR Y$ = "n" THEN GOTO 1920
1880 ''
1890    PRINT : PRINT "         Waiting for EXTERNAL TRIGGER."
1900    WAIT STATUS.REGISTER,  READ.WAIT
1910    PRINT "         EXTERNAL TRIGGER Received." : PRINT
1920 ''
1930 '' Check for ERROR.
1940 ''
1950    WAIT STATUS.REGISTER,  COMMAND.WAIT
1960    STATUS = INP(STATUS.REGISTER)
1970    IF (STATUS AND &H80) THEN GOTO 2590
1980 ''
1990    PRINT
2000    PRINT "         WRITE DIGITAL INPUT Operation Complete"
2010    GOTO 2930
2020 ''
2030 '' Decode DIO value, port 0.
```

```
2040 ''
2050    HEADER$ = "         DIO PORT 0, BIT "
2060    TEST = DIO.DATA0 : GOSUB 2140
2070    BINARY0$ = BINARY$ : RETURN
2080 ''
2090 '' Decode DIO value, port 1
2100 ''
2110    HEADER$ = "         DIO PORT 1, BIT "
2120    TEST = DIO.DATA1 : GOSUB 2140
2130    BINARY1$ = BINARY$ : RETURN
2140 ''
2150 '' Decode set and clear bits of TEST.
2160 '' ---------------------------------
2170 ''
2180    IF (TEST AND &H1) THEN PRINT HEADER$;"0 SET",
2190    IF (TEST AND &H1) = 0 THEN PRINT HEADER$;"0 CLEAR",
2200    IF (TEST AND &H1) THEN BINARY$ = "1"
2210    IF (TEST AND &H1) = 0 THEN BINARY$ = "0"
2220 ''
2230    IF (TEST AND &H2) THEN PRINT HEADER$;"1 SET"
2240    IF (TEST AND &H2) = 0 THEN PRINT HEADER$;"1 CLEAR"
2250    IF (TEST AND &H2) THEN BINARY$ = "1" + BINARY$
2260    IF (TEST AND &H2) = 0 THEN BINARY$ = "0" + BINARY$
2270 ''
2280    IF (TEST AND &H4) THEN PRINT HEADER$;"2 SET",
2290    IF (TEST AND &H4) = 0 THEN PRINT HEADER$;"2 CLEAR",
2300    IF (TEST AND &H4) THEN BINARY$ = "1" + BINARY$
2310    IF (TEST AND &H4) = 0 THEN BINARY$ = "0" + BINARY$
2320 ''
2330    IF (TEST AND &H8) THEN PRINT HEADER$;"3 SET"
2340    IF (TEST AND &H8) = 0 THEN PRINT HEADER$;"3 CLEAR"
2350    IF (TEST AND &H8) THEN BINARY$ = "1" + BINARY$
2360    IF (TEST AND &H8) = 0 THEN BINARY$ = "0" + BINARY$
2370 ''
2380    IF (TEST AND &H10) THEN PRINT HEADER$;"4 SET",
2390    IF (TEST AND &H10) = 0 THEN PRINT HEADER$;"4 CLEAR",
2400    IF (TEST AND &H10) THEN BINARY$ = "1" + BINARY$
2410    IF (TEST AND &H10) = 0 THEN BINARY$ = "0" + BINARY$
2420 ''
2430    IF (TEST AND &H20) THEN PRINT HEADER$;"5 SET"
2440    IF (TEST AND &H20) = 0 THEN PRINT HEADER$;"5 CLEAR"
2450    IF (TEST AND &H20) THEN BINARY$ = "1" + BINARY$
2460    IF (TEST AND &H20) = 0 THEN BINARY$ = "0" + BINARY$
2470 ''
2480    IF (TEST AND &H40) THEN PRINT HEADER$;"6 SET",
2490    IF (TEST AND &H40) = 0 THEN PRINT HEADER$;"6 CLEAR",
2500    IF (TEST AND &H40) THEN BINARY$ = "1" + BINARY$
2510    IF (TEST AND &H40) = 0 THEN BINARY$ = "0" + BINARY$
2520 ''
2530    IF (TEST AND &H80) THEN PRINT HEADER$;"7 SET"
2540    IF (TEST AND &H80) = 0 THEN PRINT HEADER$;"7 CLEAR"
2550    IF (TEST AND &H80) THEN BINARY$ = "1" + BINARY$
2560    IF (TEST AND &H80) = 0 THEN BINARY$ = "0" + BINARY$
```

```
2570 ''
2580    PRINT : RETURN
2590 ''
2600 '' Fatal board error.
2610 ''
2620    PRINT
2630    PRINT "FATAL BOARD ERROR"
2640    PRINT "STATUS REGISTER VALUE IS ";HEX$(STATUS);" HEXIDECIMAL"
2650    PRINT : BEEP : BEEP : GOSUB 2700
2660    PRINT "ERROR REGISTER VALUES ARE:"
2670    PRINT "      BYTE 1 -   ";HEX$(ERROR1);"  HEXIDECIMAL"
2680    PRINT "      BYTE 2 -   ";HEX$(ERROR2);"  HEXIDECIMAL"
2690    PRINT : GOTO 2930
2700 ''
2710 '' Read the Error Register.
2720 '' ------------------------
2730 ''
2740    OUT  COMMAND.REGISTER, CSTOP : TEMP = INP(DATA.REGISTER)
2750 ''
2760    WAIT STATUS.REGISTER, COMMAND.WAIT
2770    OUT COMMAND.REGISTER, CERROR
2780 ''
2790    WAIT STATUS.REGISTER, READ.WAIT
2800    ERROR1 = INP(DATA.REGISTER)
2810    WAIT STATUS.REGISTER, READ.WAIT
2820    ERROR2 = INP(DATA.REGISTER)
2830    RETURN
2840 '' ------------------------
2850 ''
2860 '' Illegal Status Register.
2870 ''
2880    PRINT
2890    PRINT "FATAL ERROR - ILLEGAL STATUS REGISTER VALUE"
2900    PRINT "STATUS REGISTER VALUE IS ";HEX$(STATUS);" HEXIDECIMAL"
2910    BEEP : BEEP
2920 ''
2930    PRINT : PRINT
2940 ''
2950   INPUT "         Run program again (Y/N)";Y$
2960   IF Y$ = "Y" OR Y$ = "y" THEN RUN
2970   IF Y$ = "N" OR Y$ = "n" THEN GOTO 3010
2980 ''
2990   PRINT : PRINT "         Please respond with 'Y' or 'N'."
3000   GOTO 2940
3010 ''
3020   INPUT "         Return to MENU (Y/N)";Y$
3030   IF Y$ = "Y" OR Y$ = "y" THEN RUN MENU$
3040   IF Y$ = "N" OR Y$ = "n" THEN GOTO 3080
3050 ''
3060   PRINT : PRINT "         Please respond with 'Y' or 'N'."
3070   GOTO 3010
3080   END
3090 ''-------------------------------------------------------------
```

```
3100 '' R O U T I N E   F O R   F O R W A R D   M O V E M E N T
3110 ''-------------------------------------------------------------
3120 FOR I = 1 TO NUMSTEPS
3130   DIO.DATA0 = 21
3140   GOSUB 1610
3150   GOSUB 3340
3160   DIO.DATA0 = 255
3170   GOSUB 1610
3180   GOSUB 3340
3190 NEXT I
3200 RETURN : END
3210 '-------------------------------------------------------------------
3220 '' R O U T I N E   F O R   B A C K W A R D   M O V E M E N T
3230 '-------------------------------------------------------------------
3240 FOR I = 1 TO NUMSTEPS
3250   DIO.DATA0 = 42
3260   GOSUB 1610
3270   GOSUB 3340
3280   DIO.DATA0 = 255
3290   GOSUB 1610
3300   GOSUB 3340
3310 NEXT I
3320 RETURN : END
3330 '-------------------------------------------------------------------
3340 ''  R O U T I N E   F O R   D E L A Y
3350 '-------------------------------------------------------------------
3360 FOR J = 1 TO IDELAY
3370   X = X+1 : X = X-1
3380 NEXT J
3390 RETURN : END
```

A PROGRAM TO TEST THE IEEE-488 DRIVERS

```
/*
This program is used to test the Lattice c IEEE 488
drivers for the Ziatech ZT 1488 board. Each driver can
be tested by inputing the statement to be tested.  Any
other required inputs are prompted for.  To exit type
"end". In this version this program has been modified to change
arg form for term and strlen from pointer to value.

To verify operation of each of the routines a ZT 488
analyzer or IEEE 488 instrument is used.

*/
/*  FOR WRITING ANY MODULES TO "LISTEN", "TALK" OR CONTROL
     THE IEE-488 BUS, IT IS BETTER TO BUY THE ZT 488 LOGIC
     ANLYZER . This is required to debug the program.  */

#include "stdio.h"
#include "ctype.h"

#define NREG 25  /* number of chars in time string */
#define N 8097  /* size of data buffers */

#define GETS(str) gets(str)
#define GETI(i) scanf("%d", &i);getchar()
#define GETH(j) scanf("%x", &j);getchar ()
#define LF '\012'  /* line feed terminator string */
#define LINE_MODE 30
#define COL_MODE   1
#define TRUE 1
#define FALSE 0

extern ercode;     /* set by 488 */
extern rcvlen;     /* likewise */
extern primaryadr;   /* GPIB I/O addresss */
extern clkadr;   /* Clock I/O address */
extern secondaryadr;   /* zSBX I/O address */
extern year;     /* clock year */
extern delayconst;   /* const for software delay */

char devlst[80] = {0};
int  line = {1};  /* default is column mode */
int  b_len;
int  files = FALSE;  /* flag for files command */
char ans[3];
```

```
int  buffer[N];  /* space for both types of buffers */
char devdata[N];
int  i1 = {3000};  /* initial timeout value 3 seconds*/
int  i2;   /* temp location */
int  i0 = {0};  /* location containing a zero */
int  di5180 = {0};  /* 5180 flag set false */
int   found;

main ( k, argv ) int k; char *argv[];
{
char command[80];
char tim[NREG];
int  i;

     /* Now initialize various parameters within the driver.
           Note, the I/O addresses can be changed.       */

  year = 0x3538;   /* ascii code for '85' */
 delayconst = 42;  /* const for IBM 4.7 MHz clock */
 clkadr = 0x240;   /* clock I/O address */
 primaryadr = 0x210;  /* GPIB I/O address */
 secondaryadr = 0x220;  /* zSBX I/O address */

 /* If primaryadr or clkadr are changed form the defaults
          then place a call to setaddr here. This will change
          the factory default I/O addresses.              */

timedy(i1);  /* set initial time delay in case of init error */

i = 1;  /* for init w/commands */
init(i);  /* always perform an init on entry */
if(ercode == 5) {
 printf("\nTime out occurred during initialization.  Turn on a device");
 exit(1); /* exit the program */
};
timedy(i0);  /* restore time delay to forever */
term(LF);  /* and set line feed as default terminator */
printf("\nFirst test routine must be 'devlst' to set a device address");
printf("\n'help' will provide a brief list of available commands.\n");

do {
 printf ( "\n enter routine to test - " );
 do {
 GETS ( command );
 } while (strlen(command) == 0);
 strlower(command);  /* everything lower case */
 decode1 ( command );
 if (found == 0) {
        decode2 ( command );
        }
} while ( strcmp ( command, "end" ) != 0 );
}      /* main */
```

```
decode1 ( command ) char *command;
{

int datalen;

found = 1;

if (strcmp(command, "help") == 0) {
 printf("\n\t\t\tCommands are:");
 printf("\nIEEE-488 Routines.");
 printf("\n\tdevlst\tset device address");
 printf("\n\tinit\tinitialize (done automatically at start)");
 printf("\n\tline\tset recvst output to line or col mode");
 printf("\n\tsendst\tsend string to device");
 printf("\n\trecvst\tretrieve data from device");
     printf("\n\tremote\tenable remote programming");
 printf("\n\tdevclr\tclear device");
 printf("\n\tlocl\tset device to local");
 printf("\n\tspoll\tperform serial poll (after srqsta)");
 printf("\n\ttimedy\tsets timeout delay per handshake");
 printf("\n\tsenddm\t\tsend string to device using dma");
 printf("\n\trecvdm\tretrieve data from device using dma");
     printf("\n\tsendbn\tsend binary data to device");
     printf("\n\tsnddbn\tsend binary data to device using dma");
 printf("\n\tterm\tchange default string terminator character");
     printf("\n\tsetpri\tuse primary (default) IBM i/o port address(0210h)");
 printf("\n\tsetsec\tuse secondary ZT 1488 IBM i/o port address (0220h)");
 printf("\n\tsrqsta\treturns current srq status");
     printf("\n\tlokout\tlocal lock out");
     printf("\n\teoi\tsuppress eoi");
     printf("\n\tbustat\tget bus status");
 printf("\n\tfiles\tpermits storage of data in a disk file");
 printf("\n\tppoll\treturn parallel poll status");
 printf("\n\tppollu\tunconfigure for parallel poll");
 printf("\n\tppolle\tenable parallel poll");
 printf("\n\tppolld\tdisable parallel poll");
     printf("\n\ttrig\tgroup execute trigger");
 printf("\n\tsetaddr\tchange I/O addresses");
     printf("\n\tdoc\tget version number");
 printf("\n\tcmd\toutput IEEE 488 command");
 printf("\n\txfer\ttransfer between devices");
 printf("\nReal Time Clock Routines 'ZT 1488'.");
     printf("\n\talarm\tset clock alarm");
     printf("\n\tloadl\tload clock latches");
     printf("\n\treadl\tread clock latches");
 printf("\n\tloadc\tload clock counters");
 printf("\n\treadc\tread clock counters");
 printf("\n\tsystic\tset periodic tics");
 printf("\n\tinstat\tget status\n");

 return;
};
```

```
if (strcmp(command, "devlst") == 0) {
  getlist(devlst);
  return;
};

if (strlen(devlst) == 0 ) {
 printf("\nDevice list must be set up before any other command.");
 return;
};

if ( strcmp ( command, "init" ) == 0 )
 {
 int t;

 printf ("\ninit data - ");
 GETI ( t );
 init ( t );
 }

else if ( strcmp ( command, "sendst" ) == 0 )
 {
 getdata( devdata );
 sendst( devlst, devdata );
 puts (devlst );
 puts (devdata );
 }

else if (strcmp (command, "sendbn") == 0)
 {
 getdata( devdata );
 b_len = strlen(devdata);
 printf("\nbuffer length is %d.  Do you wish to change it (y/n)? ");
 GETS(ans);
 if (tolower(ans[0]) == 'y') {
  printf("\nEnter new length - ");
  GETI ( b_len );
 }
 sendbn( devlst, devdata, b_len );
 puts (devlst );
 puts (devdata );
 }

else if ( strcmp ( command, "recvst" ) == 0 )
 {
 int i ;
 int actualen ;

 printf( "\ninput number of bytes - ");
 GETI ( datalen );

 setstr ( devdata, datalen < N ? datalen : N );
 recvst ( devlst, devdata );
 printf ("\nnumber bytes requested = %d\n", datalen);
```

```
 printf ("number bytes untransmitted %d\n", rcvlen);
 actualen = datalen - rcvlen;
 printf ("number bytes received  = %d\n", actualen);

 prntdata(devdata,actualen,line);

 }

else if ( strcmp ( command, "cmd" ) == 0 )
 {
 getdata ( devdata );
 cmd ( devdata );
 }

else if ( strcmp ( command, "xfer" ) == 0 )
 {
 xfer ( devlst );
 }

else if ( strcmp ( command, "srqsta" ) == 0 )
 {
 int stat;

 srqsta ( &stat );
 printf ( "\nstatus = %x\n", stat );
 }

else if ( strcmp ( command, "remote" ) == 0 )
 {
 remote ( devlst );
 }

else if ( strcmp ( command, "locl" ) == 0 )
 {
 locl ( devlst );
 }

else if ( strcmp ( command, "lokout" ) == 0 )
 {
 lokout();
 }

else if ( strcmp ( command, "devclr" ) == 0 )
 {
 devclr ( devlst );
 }

else if ( strcmp ( command, "trig" ) == 0 )
 {
 trig (devlst);
 }

else if ( strcmp ( command, "doc" ) == 0 )
```

```
     {
     char  ver[35];

     setstr (ver, 35);
     doc (ver);
     printf ("%s\n", ver);
     }

else if ( strcmp ( command, "eoi" ) == 0 )
     {
     int     t;

     printf ("\nenter 1 for EOI, 0 to supress EOI - ");
     GETI ( t );
     eoi ( &t );
     }

else if ( strcmp ( command, "bustat" ) == 0 )
     {
     int t;

     bustat( &t );
     printf ( "\n status = %x\n", t );
     }

else if ( strcmp ( command, "term" ) == 0 )
 {
 printf("\n Enter terminating character - ");
 ans[0] = getche();
 ans[1] = ' ';
 ans[2] = 0;
 i2 = term (ans[0]);
 printf("\nprevious terminating character was %02x (hex)", i2);
 }

else if ( strcmp ( command, "timedy" ) == 0 )
 {
 int msecs;

 printf("\nmilliseconds timeout - ");
 GETI ( msecs );
 i2 = timedy ( msecs );
 printf("\nprevious value of timeout was %d millisec.", i2);
 }

else if ( strcmp ( command, "ppoll" ) == 0 )
 {
 int response;

 ppoll (&response);
 printf ("ppoll response = %x\n",response);
 }
```

```
else
 {
 found = 0;
 }

if (ercode != 0)
 {
 printf ( "\nerror code for %s is %d\n", command, ercode );
 }

 }  /* decode1 */

decode2 ( command ) char *command;
{

int datalen;

found = 1;

if ( strcmp ( command, "ppolld" ) == 0 )
 {
 ppolld (devlst);
 }

else if ( strcmp ( command, "ppolle" ) == 0)
 {
 int conf;

 printf ("\nconfiguration data - ");
 GETI ( conf );
 ppolle (devlst,conf);
 }

else if (strcmp ( command, "ppollu" ) == 0 )
 {
 ppollu ();
 }

else if ( strcmp ( command, "spoll" ) == 0 )
 {
 int status[36];
 char device[9] ;
 int i;
 int n = 10;

 for (i = 0; i < 36; i++)
  status[i] = 0;   /* zero status array */
 setstr ( device, 8 );
 spoll ( devlst, &status[1], device );
/*
 * don't bother displaying first srq if none were detected
 */
 if(device[0] == '9' && device[1] == '9')
```

```
 ;
 else
     printf ( "\nfirst device responding with SRQ is %s\n", device);
 printf("\ndev#\thex\toctal\n");
 for ( i=1; i < n; i++)
  printf("\n%d\t%x\t%o", i, status[i], status[i]);

 }

else if ( strcmp ( command, "setpri" ) == 0 )
 {
 setpri ();
 }

else if ( strcmp ( command, "setsec" ) == 0 )
 {
 setsec ();
 }

else if ( strcmp ( command, "setaddr" ) == 0 )
     {
     int t;

 printf ("\ncurrent GPIB I/O address is %x (hex), new hex value - ", primaryadr );
 GETH ( t );
 primaryadr = t;
 printf ("\n current clock I/O address is %x (hex), new hex value - ", clkadr );
 GETH ( t );
 clkadr = t;

 setaddr ();
 }

else if ( strcmp ( command, "senddm" ) == 0 )
 {
 getdata (devdata);
 senddm (devlst,devdata);
 puts (devlst);
 puts (devdata);
 }

else if (strcmp (command, "snddbn") == 0)
 {
 getdata( devdata );
 b_len = strlen(devdata);
 printf("\nbuffer length is %d.  Do you wish to change it (y/n)? ");
 GETS(ans);
 if (tolower(ans[0]) == 'y') {
  printf("\nEnter new length - ");
  GETI ( b_len );
 }
 snddbn( devlst, devdata, b_len );
 puts (devlst );
```

```
 puts (devdata );
 }

else if ( strcmp ( command, "recvdm" ) == 0 )

 {
 int i,j;
 int actualen;
 union {
  char rdgc[2];
  int  rdgi;
 } fix;
 char *idx;   /*  temp pointer */
 int  *idi;
 printf ("\ninput number of data bytes - ");
 GETI ( datalen ) ;
 setstr (buffer, datalen < N ? datalen : N );
 recvdm ( devlst, buffer );
 printf ( "\nnumber bytes requested = %d\n", datalen);
 printf ( "number bytes untransmitted %d\n", rcvlen);
 actualen = datalen - rcvlen;
 printf ( "number bytes received = %d\n", actualen);
 prntdata(buffer,actualen,line); /* normal data output */
 }

else if ( strcmp ( command, "loadc" ) == 0 )
 {
 char string[NREG];

 getdate (string);
 loadc (string);
 }

else if ( strcmp ( command, "loadl" ) == 0 )
 {
 char string[NREG];

 getdate (string);
 loadl (string);
}

else if (strcmp ( command, "readc" ) == 0 )
 {
 char tim[NREG];

 setstr ( tim, NREG-1);
 readc ( tim );
 pdate ( tim );
 }

else if ( strcmp ( command, "readl" ) == 0 )
 {
 char tim[NREG];
```

```
 setstr ( tim, NREG-1 );
 readl ( tim );
 pdate ( tim );
 }

else if ( strcmp ( command, "synch" ) == 0 )
 {
 synch ();
 }

else if ( strcmp ( command, "systic" ) == 0 )
 {
 int stat;

 printf ("\nperiodic value - ");
 GETI ( stat );
 systic (stat);
 }

else if ( strcmp ( command, "alarm" ) == 0 )
 {
 char tim[NREG];

 setstr ( tim, NREG-1 );
 getdate ( tim );
 alarm ( tim );
 }

else if ( strcmp ( command, "instat" ) == 0 )
 {
 int stat;

 instat ( &stat );
 printf ("\nstatus = %02x (hex)", stat);
 }

else if ( strcmp ( command, "end" ) == 0 )
    ;

else if (strcmp(command, "line") == 0 ) {

   printf("\n recvst data may be output in either a horizontal line (best");
   printf("\n for ascii data) or in column format in octal, hex and ascii");
   printf("\n (better for binary or mixed ascii/binary data)");
   printf("\n Enter 'l' to output data in line form, else 'c' for column - ");
   ans[0] = getche();
   line = ((tolower(ans[0]) == 'l') ? LINE_MODE : COL_MODE);
}

else if (strcmp(command, "files") == 0 )
 {
 printf("\n Turn data file recording flag ON or OFF - ");
```

```
 gets(ans);
 strlower(ans);
 files = ((strcmp(ans,"on") == 0) ? TRUE : FALSE );
 }

else
 {
 puts ( "illegal command\n\n" );
 }

if (ercode != 0)
 {
 printf ( "\nerror code for %s is %d\n", command, ercode );
 }

}    /* decode2 */

getlist ( dl ) char *dl;
{
 printf("\nEnter device list ");
 GETS ( dl );
 strcat ( dl, " " );
}      /* getlist */

getdata ( dl ) char *dl;
{
 printf ( "\nenter data string " );
 GETS ( dl );
}      /* getdata */


setstr ( string, n ) char *string; int n;
{
int i;

for ( i = 0; i < n; i++)
 *string++ = ' ';
*string = 0;
}      /* setstr */

getdate ( dl ) char *dl;
{
 int len;

 printf ( "\nInput time and date:\n" );
 if ((len = strlen (dl))  < (NREG - 1))
    {
    printf ("\nstring must be longer then %d chars \n", len);
    }
 printf ( "\nHT SS MM HH DW DM MO YR\n");
 GETS ( dl );
 strcat (dl, " ");
}      /* getdate */
```

```
pdate (dl) char *dl;
{
 printf ("\nHT SS MM HH DW DM MO YR\n");
 printf ( "%s\n",  dl );
}      /* pdate */

strlower(strng)  /* convert string to lower case */
char *strng;
{
 while (*strng) {
  *strng = tolower(*strng);
  *strng++;
 }
}

prntdata(dbuf,actualen,line)
char *dbuf;
int  actualen,line;
{
 /* this routine just prints out the received data in column or
  * line format depending on value of line
  */

 int i;
 char ans[4], filename[12], c;
 FILE *fl;

 for ( i = 0; i < actualen; i++ ) {
  if (i % (20 * line) == 0) {
   printf ("\n press any key to continue (e to exit) ");
   ans[0] = getch();
   printf("\n");
   if (ans[0] == 'e')
   break;
   }
  if(line != COL_MODE) {
    printf("%c",dbuf[i]);
   if (i != 0 && i % 60 == 0)
    printf ("\n");
  }
  else {
     if (i % (20 * line) == 0) {
   printf ("\nbyte#\toctal\thex\tcharacter\n");
   }
     printf ("\n%d\t%03o\t%03xh\t%c", i, dbuf[i],dbuf[i],dbuf[i] < ' ' ? '!' : dbuf[i]);
  }
 };
 if (files == TRUE) {
  printf("\nwrite data to file (y/n)? ");
  c = getche();
   if (tolower(c) == 'y') {
```

```
    printf("\nenter file name: ");
    gets(filename);
    if ((fl = fopen(filename,"w")) == NULL) {
     printf("\ncannot open output file");
     exit(1);
    }
    fputs("\n");
    fclose(fl);
   }
 }
}
```

DEVELOPMENT OF CHEMICAL MEASUREMENTS USING ELECTRODES

Arland H. Johannes
School of Chemical Engineering
Oklahoma State University

University Center for Water Research
Oklahoma State University
Stillwater, Oklahoma 74078

July 1987