U. S. DEPARTMENT OF THE INTERIOR
U. S. GEOLOGICAL SURVEY

Data Acquisition System for Magnetotellurics

by

Thomas P. Grover[1]

Open-File Report 92-569
1992

---

[1] U.S. Geological Survey, Branch of Geophysics, Golden, Colorado 80401

Abstract

This report describes a data acquisition system suitable for magnetotellurics. It converts signal voltages from electrode or coil amplifiers to digital data and stores the values in local memory. The data is later transferred via a serial cable to a personal computer under software control. The personal computer can set measurement parameters for the data acquisition system thru the same serial cable. The report includes a technical discussion, circuit schematics and some code for the software. The system comprises a single circuit board and is compact, lightweight, and uses very little power. It can convert 8 channels to 12 bit accuracy at a 2 kHz scan rate. The circuit uses a Motorola MC68HC811 microcontroller and is programmable from a personal computer.

The following abbreviations are used in this report:

- A/D - Analog to digital converter
- BIOS - Built In Operating System
- DAS - Data acquisition system
- FIFO - First In, First Out memory
- IC - Integrated Circuit
- MC - microcontroller integrated circuit
- PC - Personal Computer
- RAM - Random access memory
- ROM - Read Only Memory
- RS232- Serial data communications link

Introduction

Magnetotelluric surveys of deep formations measure low frequency electric and magnetic fields by using electrodes and coils to convert the fields to voltages (Kaufman and Keller, 1981). The voltages are amplified with frequency selective circuits to one volt levels. These waveforms are digitized, stored in a computer and analyzed. Typical frequencies range from 0.001 Hz to 1000 Hz. The waveforms are represented by a discrete sequence of samples, usually a power of 2, typically 1024. Analysis includes computation of discrete-frequency power-spectral and cross spectral analysis to derive an earth transfer function, generally in two dimensions.

We have developed a data acquisition system (DAS) to provide the interface between the high level signal voltages and a personal computer (PC)(Swiger and Glover, 1991). The devices communicate via a serial (RS232) link thereby providing compatibility with many types of computers (Brey, 1988). The signals are converted to digital data at a rate dependant on the desired frequency range and sent to the computer via the serial link. The sampled data is buffered by local memory in the DAS. When measuring low frequency signals, the buffer memory is used for only one scan so that digitized data is sent to the computer and displayed as soon as all channels are converted. For higher frequencies the serial link transfer rate prevents real time display of digitized data.

Based on the needs of magnetotellurics, we have chosen the following set of system design criteria:

1- Controlled by a personal computer.
2- 4 to 8 signal channels with matched analog responses.
3- 0.5 Hz to 2 kHz scan rate.
4- 256 to 2048 samples per waveform.
5- Weight less than 2 lb (16 hour battery pack).

We use a 12 bit analog to digital converter (A/D) but a 16 bit device can be accomodated. There is a single A/D with a multiplexer to sequentially connect its input to the individual analog channel signal voltages. This forces us to have an interchannel time skew; the signal samples are not acquired at the same time. This skew is acceptable because the analysis uses frequency spectra, which change very slowly compared with the waveform acquisition time, but the computer cross-spectra are generally deskewed in the frequency domain. Our basic skew is 40 microseconds, independant of the intersample time. The input range is +10 volts to -10 volts and can be varied by changing resistors. The serial link operates at 9600 baud and can be programmed to any common value below 38 kilobaud.

System Description

A block diagram of the DAS is shown in figure 1. To elucidate the block functions we will describe an acquisition sequence. The central microcontroller integrated circuit (MC), an MC68HC811 from Motorola, contains sequencing code in its read only memory (ROM). The control program begins when power is applied and various portions of the code are executed under PC direction by the receipt of code bytes at the serial port of the MC.

The MC has four digital ports, A thru D, which are one byte wide. Port A outputs the multiplexer channel number on 3 bits. Port B outputs control signals for the A/D, random access memory (RAM) and address counter. Port C is digital input from either the A/D or from the RAM. Port D uses 2 bits for the serial link, an input and an output. Level shifting circuitry converts the 5 volt internal logic levels to RS232 levels (+5 to -5 volts).

DAS local memory consists of a 32 kilobyte RAM integrated circuit and a 15 bit up counter, labeled the 'address counter'. The memory size is determined by our choice of 8 channels X 2 byte per channel X 2048 scans = 32768 bytes. Together these chips constitute a first in, first out memory (FIFO). The use of a counter to generate the address for the RAM is a tradeoff between speed and circuit complexity. The MC68HC811 could compute and output the 15 bit addresses required by our application, but this would use up half of the available digital outputs and increase the intersample time skew. The MC is an 8 bit device and address generation would take several software instructions. With the address counter, addresses are 'computed' by clocking the counter, which takes one bit and two software instructions. The RAM is always loaded and unloaded sequentially, starting at the first address.

An A/D conversion sequence loads 2 bytes into RAM. An 8 channel sample or "scan" loads up to 16 bytes into RAM, i.e. all channels are converted. The scan results may be transmitted to the PC, if time permits, or scans may be repeated until the desired number is acquired and then all of the scans are transmitted to the PC. DAS activity always begins upon receipt of a code byte from the PC which causes the address counter to be set to zero and conversion sequences to begin. Activity always ends with the transmission of results to the PC, ranging from 8 bytes to 32 kilobytes.

This DAS design uses only part of the MC68HC811's capability. We feel that overdesign is appropriate for microcontrollers because the learning curve is rather steep and a "one size fits all" approach will make the best use of design time for low volume, cost insensitive systems. The 68HC811 can be run at very low power, has extensive capabilities and lots of software. Motorola maintains a bulletin board with assemblers, "C" compilers, floating point libraries and simulators (Dumas, 1991). No additional development equipment is needed; the chip can be programmed from a PC's RS232 port(Motorola Inc., 1988). The documentation is superb and there are many textbooks explaining real time microcontroller assembly language programming (Brey, 1988).

The PC uses two ancillary programs to create and enter new code in the 6811's read only memory. A Motorola assembler, AS11.EXE, converts ASCII text source code files to hexadecimal strings of control code and address specifications. A bootloader program is built into the MC68HC811. When the chip is powered up with the boot-line low, the bootloader program takes control and reads 256 bytes from the serial port into random access memory within the chip and transfers control to the first address in the random access memory. The 256 byte program reads the serial port, converts the hex strings to binary bytes and stores them in read only memory. The program to create and download the 256 byte program to the microcontroller memory and to read and transmit the object files produced by the assembler is written in BASIC and is available as EELOAD.BAS from Motorola.

Microcontroller Software

The main loop of the microcontroller software is shown in the flow diagram of Figure 2. Some of the blocks shown are discussed in greater detail using the timing diagram of figure 3. The main loop receives an RS232 byte from the PC and checks for the characters "A" or "P". If "P" is received, the next 5 bytes from the PC are loaded and interpreted as acquisition parameters: interscan time, number of scans and number of channels. The first two items require 2 bytes each. If "A" is received, the current value of the interscan time is checked. If it is greater than 50 milliseconds, the "slow scan" routine, which allows real time display, is executed; otherwise the "fast scan" routine is done.

The 50 millisecond (20 Hz) interscan time allows us to transmit 16 bytes with software handshaking at a 9600 baud RS232 rate. This rate needs about 1 millisecond per byte because each byte uses 11 baud periods when start, stop and intercharacter bits are included. The software handshake adds another 16 bytes from PC to DAS with full time offset, for a total of 32 milliseconds. The PC has about three milliseconds to process each byte and store it to memory. Our current routines do not use the software handshake but synchronization might be necessary for some data processing programs to avoid loss of data.

The "slow scan" routine zeros the address counter, sets up the interscan timer interrupt routine and the RS232 interrupt routine. One scan (all channels) is converted and stored in memory. The address counter is set to zero again and the results are sent to the PC. The MC then idles, waiting for an interrupt. If the timer interrupt occurs before the RS232 interrupt, the routine quits to the main loop. If the character "C" is received at the serial port before a timeout occurs the MC waits for the timer interrupt and then processes another scan.

The "fast scan" routine zeros the address counter, sets up the interscan timer interrupt routine and the number of scans. A scan is converted and stored in memory. The scan count is decremented and if the count is nonzero, the MC idles until a timer interrupt occurs, indicating that another sample should be converted. If the scan count is zero, the routine repeats the initialization of the address counter and sets the data count (# channels * # scans * bytes per channel). The memory contents are then transmitted to the PC, decrementing the data count at each byte to check when the memory is empty. When transmission is completed, the routine returns to the main loop.

Transmission from DAS to PC is currently done as a continuous stream of bytes to minimize the required time, which can be as great as 1/2 minute. The MC transfers a byte from RAM to Port D, waits for parallel to serial conversion, clocks the address counter, decrements the data count and checks it for zero and then repeats the process. This requires the full attention of the PC or an interrupt driven communication routine to avoid lost data. Note that there is no header or synchronizing information. If one byte is missed then all the succeeding bytes are worthless. The PC has about one millisecond to process each byte.

From the PC operators view, the "fast scan" routine is less convenient than the slow routine because it imposes a long waiting time before any results are available and the results are presented in a single block. The "slow scan" routine provides results instantly and in an incremental fashion. In fact, the slow routine can be run continuously in real time using appropriate PC software. A second convenient aspect of the slow routine is that it can be stopped at any time. This is helpful when debugging an instrument setup.

The conversion loop is the time sensitive portion of the microcontroller software. It is generally necessary to combine as many control actions as possible into each instruction to maximize throughput. This may obscure the program logic in favor of performance. In general, the software instruction sequence consists of loading code bytes to the MC accumulator and storing them to Port B. This requires 3 microseconds for an MC68HC811 with a 8 Mhz clock. There are 8 load and store instructions in our conversion routine plus 16 microseconds for the A/D chip to perform the conversion yielding our interchannel skew time of 40 microseconds. For 8 channels we need 320 microseconds to acquire one scan. We chose a minimum interscan time of 500 microseconds (2 kHz) but could have gone as low as 330 microseconds (3 kHz). If less than 8 channels are used the minimum interscan time can be reduced. Note that these times are exact, to crystal oscillator tolerances. Unlike personal computers, the microcontroller does not have background operating system activity and uncontrolled timer interrupts. Its internal circuitry also operates certain processes in parallel with the central processor: RS232 communication and timers.

Refer to figure 3 for a timing diagram of the activity on the control lines (Port B) during the conversion of one channel and the storage of 2 bytes to memory. The vertical lines, numbered 1 thru 8, refer to the instant when a new control byte appears at the port.

The sequence begins when the control line for the A/D is brought low to initiate conversion. The time from 1 to 2 is the conversion time for the A/D chip and depends on the type of analog to digital converter. It is not represented to scale in the diagram. If the A/D chip included a sample and hold feature then the time from 1 to 3 would be available for conversion. The time from 1 to 2 is set with a software delay loop and includes instructions for incrementing the channel number and testing for the end of a scan acquisition sequence. If all channels have been converted, this is the exit point for the routine.

At time 2, the multiplexer channel is changed at Port A. This allows ample time for the analog output of the multiplexer to settle to the value of the next channel. If the settling time is too slow, the results will appear to have channel to channel crosstalk.

At time 3 the memory write line is brought low in preparation for latching the data into RAM on the rising edge of the write waveform. Logically, this could have been combined with the multiplexer address change at time 2 but the MC has only one accumulator and it was occupied at time 2 with servicing port A. Data is latched at time 4 and you will note that the address and A/D control line are stable at this time as required by the RAM internal logic.

At time 5 the clock line is raised to advance to the odd numbered addresses used by the high order data bits. The HiByte line is also the low order address line. The A/D control line is brought high in preparation for placing the high order bits on the data bus. At time 6 the clock pulse is terminated and the falling edge of the A/D control line waveform causes the A/D to take control of the data bus. The RAM write line is brought low to prepare to latch the high order data bits with the rising edge at time 7.

At time 8 the single channel conversion sequence is completed by clocking the address counter and raising the A/D control line. This brings the logic levels back to those which were present before time 1. The only net change is that the address counter has been advanced by two counts.

The source code for the MC68HC811 is shown in Appendix A and has been heavily commented. It is written in 6811 assembly language and can be compiled with the AS11 program to Motorola hexadecimal code.

Personal Computer Software

This report describes a program which will download acquisition parameters to the DAS and can display, in graphical form, the measurement results. This program is intended for illustration only and does not save the measurements to disc or perform any analysis. The source code is in Appendix B and has been written in Borland's Turbo C version 2 and compiled using the small memory model. The graphics file EGAVGA.BGI must be in the current directory when running this program.

The PC software presents menus and decodes keystrokes to set acquisition parameters and initiate data acquisition, thereby eliminating the need for complex error checking. The PC code does some table lookups to create two byte hexadecimal data for downloading. We will discuss only the portions of the program which interact with the DAS and provide other details in the program source code in the appendix.

Calls to the Built In Operating System (BIOS) are used for RS232 setup and transmission. These are contained in subroutines which can be rewritten for direct reading and writing to the relevant hardware port addresses. It is assumed that a status byte is available which indicates that data is ready to be read or that transmission is complete, and that the serial port has some sort of local memory arranged in a first-in-first-out fashion and this FIFO is initialized before reading data from the DAS. Note that some dialects of BASIC append a newline character to every string transmitted from the serial port and will cause problems with our DAS software.

Experimentation disclosed that our BIOS demanded certain connections at the RS232 connector:

Identification	pin #	connect to
DCD	1	+5 volts
TX	2	DAS RS-232 input
RX	3	DAS RS-232 output
COMM	5	DAS circuit common (ground)
DSR	6	+5 volts
RTS	7	CTS (i.e. connect pin 7 to pin 8)
CTS	8	RTS

The pin numbers are for a 9 pin connector. Data Carrier Detect and Data Set Ready are held high to imitate a modem's response to a valid telephone link connection. Ready to Send connected to Clear to Send provides a hardware handshake commonly found in modems.

Figure 4 shows the flow chart for the PC software. The main loop restores the screen to text mode, transmits the current set of acquisition parameters as a 6 byte string whose first character is a "P", displays the main menu, clears the keyboard buffer and waits for a keypress. If the key is for parameter selection, a new menu is presented and a second key press is acquired to determine the operator's choice. The only error messages are for illegal (i.e. not on the menu) keypresses.

If data acquisition is selected, the serial port FIFO is cleared, the screen is set to a graphical display and the character "A" is transmitted. As each set of scan data arrives at the serial port it is converted to a pixel and displayed. For long interscan times the character "C" is transmitted after each scan is received.

The routine to capture and display signal waveforms is divided into two loops: the inner one counts channels and the outer loop counts scans. For short interscan times the first trip thru the inner loop takes a long time because the DAS must fill its RAM before sending data. The time to traverse the inner loop must be less than 1 millisecond to avoid losing data. When the outer loop is satisfied that all scans have been received it idles until a key is pressed in order to allow the operator to examine the waveforms.

The inner loop captures two bytes from the serial port, left shifts the second one and adds them together to create a binary offset representation of the data value, equal to the D/A output. The data is displayed as a pixel whose vertical position corresponds to the value. The pixel horizontal position corresponds to the scan number and its color corresponds to the channel number.

Data acquisition parameters are transmitted as a six byte string whose first character is a "P". The interscan time is calculated as the number of 8-microsecond MC clock ticks and sent as two bytes in hexadecimal. The number of scans is also sent as two hexadecimal bytes. The last byte is the number of channels per scan. The serial port status byte is used to determine if transmission is complete before sending the following character. If the PC has a transmission FIFO this step is not necessary.

Circuit Description

Refer to Figure 5 for a circuit schematic. This schematic has been simplified by the elimination of power supply and inactive connections. The signal path begins with the analog multiplexer, a DG508. Its channel selection pins are driven by port A of the MC. This part uses +15 volt and -15 volt supplies to accomodate an input range of +10 volts to -10 volts. If the input signal range is less then +5 volts to -5 volts the multiplexer power consumption can be reduced by using an all CMOS part, similar to the 4051, which does not have internal level shifting circuits. The multiplexer is followed by an operational amplifier to shift the voltage levels to the range of +5 volts to 0 volts used by the A/D and to eliminate loading errors due to the multiplexer internal switch resistance. The resistors connected to this operational amplifier can be changed to alter the acceptable input voltage range.

The A/D is a 12 bit unit with an 8 bit data bus output. It has an internal voltage reference which is used in the operational amplifier level shifting resistor network. Its clock is supplied by a 2 MHz square wave output by the MC. Conversion time is typically less than 12 microseconds. Many other devices are available which match the specifications for this chip. There are also several 16 bit devices available with higher power requirements and slower conversion times. Some chips require a separate line to enable the tristate output drivers.

The address counter comprises four 4029 chips with a common clock, whose 'carry in' inputs are connected to the 'carry out' output of the preceeding device. The ripple carry time is much less than the instruction cycle time of the MC. The other half of the memory FIFO is a RAM device, the MC60LHC256. The output enable signal for the RAM is developed by inverting the A/D control signal which eliminates bus contention between the two devices.

The microcontroller wiring is similar to examples described in the Motorola application literature (Motorola Inc. 1991). Unused pins are pulled up to +5 volts with 22 kohm resistors and are not shown on the schematic. An inverter to sharpen the reset signal and discrete circuits level shift the RS232 signals. The memory map is shown below (hexadecimal digits):

0000-00FF	variable and stack RAM
1000-103F	registers and input/output ports
F800-FFFF	Program storage in internal EEPROM (electrically erasable and programmable read only memory)

We did not make a special effort to build a low power system but the following changes will produce a milliwatt DAS:

1- Use a 4051 multiplexer and a +2.5volt to -2.5 volt input range.
2- Operate with +5 volt and -5 volt supplies only. Use a 7660 charge pump IC to generate the -5 volts.
3- use a lower power A/D similar to the Maxim MAX190.
4- Wire the RAM and A/D chip selects into port B and modify the software to shut down unused portions of the system as often as possible.
5- In the "slow scan" mode, use an external clock to restart the 68HC11 at the end of the interscan period. Place the MC in a stop state after sending each scan's data.

The circuits were constructed using wire wrap sockets on a perforated board. There is no ground plane but the clock wires for the 8 MHz crystal are kept as short as possible. The only controls on the DAS are a boot/run switch. A 9 pin D-subminature connector is used for the RS232 cable.

References

Kaufman, A.A., and Keller, G.V., 1981, The Magnetotelluric Sounding Method: Elsevier, New York, 734 p.

Swiger, F. and Glover, J., 1991, The FS-100 MC68HC11- based Single Board Computer: Circuit Cellar Ink, v.6, p. 52-59

Brey, B.B., 1988, Microprocessors and Peripherals: Merrill Publishing Co., Colombus OH, 567 p.

Technical Literature Department, Motorola Inc., Phoenix, AZ.

Application Note #AN1010 , 1988, MC68HC11 EEPROM programming from a Personal Computer: 13 p.

Dumas, J., 1991, How to Use the Freeware Bulletin Board Service: 72 p.

Technical Manual for the MC68HC811E2 Microcontroller: 109 p.

Appendix A: Microcontroller Software Source Code

This code is written in the IBM-PC hosted Motorola Freeware Assembly Language. It is converted to machine code by the as11.exe assembler.

```
*   mtdas6.asm  N sample load ram  & write to pc, no pacing
*           includes parameter setup and slow sampling
*   var time & num of chan  output mux chan # on porta
*               portb: 0=clk 1=zero/cs,en 2=/wr 3=rd/g
*        use toc4 as timer.
*  pc cmd: 'A' starts acquis and dump,  slow or fast depends
*        on stime
*          'P' gets parameters, 'C' continues slow sampling
*
*        equates for 68hc11 registers
porta    equ     $00     mux chan # on bits 3-6
portb    equ     $04     all other outputs
portc    equ     $03     input
tcnt     equ     $0e     real time clock ( int )
toc4     equ     $1c     output compare 4 time value ( int )
tmsk1    equ     $22     timer intr enables
tflg1    equ     $23     timer intr flags
tmsk2    equ     $24     port a enables, clock prescaler
baud     equ     $2b     rs232 baud rate
sccr1    equ     $2c     comm params
sccr2    equ     $2d     enable tx,rx, intr
scsr     equ     $2e     comm status byte
scdr     equ     $2f     comm data i/o
*
* initialization: jump vectors, data area, stack frame
         org     $fffe           reset vector loc
         fdb     $f800           goto bottom of eprom
         org     $ffd6           rs232 vector loc
         fdb     seri            rs232 intr routine
         org     $ffe2           output compare 4 vector
         fdb     timi            toc4 interrupt routine
*
         org     $0000           internal ram
inbyte   rmb     1               rec'd pc data byte
nchan    rmb     1               number of chan +1
stime    rmb     2               s time in 8uS units ( int )
nsamp    rmb     2               # samples +1 ( int )
nbyte    rmb     2               2*#samples*#chan
*
         org     $f800           beginning of eprom code
         lds     #$00ff          set stack ptr to top of ram
         ldx     #$1000          x reg=int reg block
*
         ldaa    #$03            set prescaler to 16
         staa    tmsk2,x           8 uS/count
         ldaa    #$30            9600 baud
         staa    baud,x           set baud rate
         ldaa    #$00            8 data, no parity, 1 stop
         staa    sccr1,x          set comm params
         ldaa    #$2c            tx, rx, rx intr
```

```
         staa    sccr2,x          enable comm & intr
*
* temporary setup for #chan, sample time, # samples
         ldaa    #$05             4 channels
         staa    nchan             nchan=#chan +1
         ldd     #$007d           1 mS=125*8uS
         std     stime             stime=$7d
         ldd     #$0201           512 samples
         std     nsamp             nsamp=513
         ldd     #$0800           2048 bytes to send
         std     nbyte             nbyte=2048
*
         ldaa    #$00             setup inbyte
         staa    inbyte
         cli                      enable maskable interrupts
*
*                main loop: wait for pc command
main     ldaa    inbyte           get pc code word
         cmpa    #$50             compare to 'P'
         bne     main1            not equal: stay in main
         jmp     param             else go to param load
main1    cmpa    #$41             compare to 'A'
         bne     main3            not equal: stay in main
         ldd     stime            get stime in acc d
         cmpd    #$1800           compare to $1800, 20 Hz
         bcs     main2              goto main2 if  d < $1800
         jmp     slld               else goto 1 slow sampling
main2    jmp     rld              goto fast sampling
main3    wai                      wait for pc xmit
         bra     main             loop forever
*
*                load ram with toc4 pacing
*                initialization
rld      ldaa    #$00             disable comm intr
         staa    sccr2,x            rx, tx, rx intr all off
*
         ldaa    #$10             set toc4 intr enable
         staa    tmsk1,x            toc4 intr is on
         ldd     tcnt,x           setup initial value for toc4
         addd    stime              add stime to tcnt
         std     toc4,x             and set output compare val
         bclr    tflg1,x $ef      clear toc4 intr flag
*
         ldy     nsamp            set reg y to # sample +1
         clra                     set mux to 1st chan
         staa    porta,x            via porta
         ldaa    #$0e             zero the addr counter
         staa    portb,x            a/d off, ram on
*
*                load ram loop, acc b=current mux chan #
rld1     dey                      next sample
         beq     rdmp             goto ram dump at last sample
         clrb                     init acc b=0, current chan
sld      incb                     next channel
         cmpb    nchan            compare acc b to # chan
         beq     rld2             wait for timer,no more chan
```

```
         ldaa   #$04               start conv, a/d on
         staa   portb,x              ram off, clr zero & clk
         nop
         nop                       conversion time delay
         nop                         3 uS
         nop
         nop
         nop
         tba                       acc a=current mux chan
         lsla                      !bad pin
         lsla                      move chan # to
         lsla                        bits 4-6
         lsla                      and place it on
         staa   porta,x              porta
         ldaa   #$00               write to ram, wr low
         staa   portb,x             a/d on, ram oe off
         ldaa   #$04               toggle wr hi
         staa   portb,x
         ldaa   #$0d               clock addr, ram on
         staa   portb,x              clear a/d read
         ldaa   #$00               2nd a/d read
         staa   portb,x              ram wr lo, ram oe hi
         ldaa   #$04               clear ram write hi
         staa   portb,x              a/d rd lo, ram oe hi
         ldaa   #$0d               clear a/d read hi
         staa   portb,x              clock addr, ram oe lo
         ldaa   #$0c               toggle clock lo
         staa   portb,x              & start mux settling
         bra    sld                next channel
*
*        end of one sample load to ram
rld2     clra                      set mux to 1st chan
         staa   porta,x              via porta
         wai                       wait for toc4
         bra    rld1               next sample
*
*        ram dump routine w/o pacing, count # bytes sent in
*        reg y
rdmp     ldaa   #$00               turn off the timer intr
         staa   tmsk1,x              toc4 is off
         ldaa   #$2c               enable comm intr
         staa   sccr2,x              rx,tx,rx intr enabled
         ldy    nbyte              get # bytes to xmit in reg y
         ldaa   #$0e               zero addr counter
         staa   portb,x            ram on, a/d off
sdmp1    ldaa   #$0c               toggle zero & clk lo
         staa   portb,x             ram oe lo, a/d rd hi
         ldaa   portc,x            get ram byte
chk1     brclr  scsr,x $80 chk1 chk if xmit done
         staa   scdr,x             send ram byte, clr flag
         dey                       decr count of bytes sent
         bne    sdmp2              goto sdmp2 if not done
         clra                      else done: set acc a=0
         staa   inbyte               clear inbyte
         jmp    main               goto main
sdmp2    ldaa   #$0d               clock addr hi
```

```
        staa    portb,x         ram on, a/d off
        bra     sdmp1           continue ram dump
*
*       get one sample with toc4 pacing and rs232 output
*     pc counts bytes rec'd and continues by receiving 'C'
*               initialization
s1ld    ldaa    #$10            set toc4 intr enable
        staa    tmsk1,x           toc4 intr is on
        ldd     tcnt,x          setup initial value for toc4
        addd    stime             add stime to tcnt
        std     toc4,x            and set output compare val
        bclr    tflg1,x $ef     clear toc4 intr flag
s1ld1   clra                    set mux to 1st chan
        staa    porta,x           via porta
        staa    inbyte          clear rec'd char
        ldaa    #$0e            zero the addr counter
        staa    portb,x           a/d off, ram on
*   load ram loop, acc b=current mux chan #
        clrb                    init acc b=0, current chan
s1ld2   incb                    next channel
        cmpb    nchan           compare acc b to # chan
        beq     s1ld3           dump one sample if no more
chan
        ldaa    #$04            start conv, a/d on
        staa    portb,x           ram off, clr zero & clk
        nop
        nop                     conversion time delay
        nop                       3 uS
        nop
        nop
        nop
        tba                     acc a=current mux chan
        lsla                    !bad pin
        lsla                    move chan # to
        lsla                      bits 4-6
        lsla                    and place it on
        staa    porta,x           porta
        ldaa    #$00            write to ram, wr low
        staa    portb,x          a/d on, ram oe off
        ldaa    #$04            toggle wr hi
        staa    portb,x
        ldaa    #$0d            clock addr, ram on
        staa    portb,x           clear a/d read
        ldaa    #$00            2nd a/d read
        staa    portb,x           ram wr lo, ram oe hi
        ldaa    #$04            clear ram write hi
        staa    portb,x           a/d rd lo, ram oe hi
        ldaa    #$0d            clear a/d read hi
        staa    portb,x           clock addr, ram oe lo
        ldaa    #$0c            toggle clock lo
        staa    portb,x           & start mux settling
        bra     s1ld2           next channel
*   one sample ram dump routine with pc continuation
s1ld3   ldab    nchan           acc b=(#ch+1)
        addb    nchan           acc b=(#ch+1)*2
        decb                    # bytes+1 to xmit
```

```
        ldaa    #$0e              zero addr counter
        staa    portb,x           ram on, a/d off
s1ld4   ldaa    #$0c              toggle zero & clk lo
        staa    portb,x            ram oe lo, a/d rd hi
        decb                      chk for 2*nchan bytes
        beq     s1ld6              goto to timeout
        ldaa    portc,x           get ram byte
chk2    brclr   scsr,x $80 chk2 chk if xmit done
        staa    scdr,x            send byte to pc, clr flag
        ldaa    #$0d              clock addr hi
        staa    portb,x           ram on, a/d off
        bra     s1ld4             continue ram dump
*  timeout check for one sample dump
s1ld6   wai                       wait for toc4 or comm
        ldaa    inbyte            check inbyte for 'C'
        cmpa    #$43                if timer, inbyte=0
        beq     timer             if 'C' goto timer
        bra     s1ld7              else goto quit routine
timer   wai                       wait for timer intr
        clra                        clear acc a
        staa    inbyte              and comm byte
        bra     s1ld1              and get next sample
*  turn off timer intr to quit slow sample load
s1ld7   ldaa    #$00              turn off timer intr
        staa    tmsk1,x             toc4 disabled
        staa    inbyte            clear rec'd byte
        jmp     main              goto main routine
*
*  routine loads parameters: # chan, sample time, # samples
param   wai                       wait for next byte
        ldaa    inbyte            get stime, lo byte
        staa    stime+1             lobyte to stime +1
        wai                       wait for next byte
        ldaa    inbyte            get stime, hi byte
        staa    stime               hibyte to stime
        wai                       wait for next byte
        ldaa    inbyte            get nsamp, lobyte
        staa    nsamp+1             lobyte to nsamp +1
        wai                       wait for next byte
        ldaa    inbyte            get nsamp, hibyte
        staa    nsamp               hibyte to nsamp
        ldd     nsamp             acc d = # samples
        addd    #$01              add 1 to acc d
        std     nsamp             nsamp = # samples +1
        wai                       wait for next byte
        ldaa    inbyte            get # chan
        inca                      acc a=# chan +1
        staa    nchan             nchan = # chan +1
        ldd     nsamp             acc d=# samples +1
        subd    #$01              acc d = # samples
        asld                      acc d = 2* #samples
        std     nbyte             nbyte = 2* #samples
        clra                      acc a=0, hibyte of d
        ldab    nchan             acc b =# chan +1, d lobyte
        xgdy                      reg y = #chan +1
        ldd     nbyte             acc d = 2* # samples
```

```
mult     dey                         decr reg y, loop counter
         beq     mult1               quit if multiply done
         addd    nbyte               acc d=acc d + nbyte
         bra     mult                loop nchan times
mult1    std     nbyte               nbyte=nchan*2*# samples
         clra                        acc a =0
         staa    inbyte              clear inbyte
         jmp     main                 for return to main
*
* interrupt service routine for the RS232 port
seri     ldaa    scsr,x              read,discard stat byte
         ldaa    scdr,x              rd byte, clear com flag
         staa    inbyte              save to int ram
         rti
*
* interrupt service routine for output compare 4
timi     ldd     toc4,x              get current timer value
         addd    stime               add stime to curr time
         std     toc4,x              update output compare val
         bclr    tflg1,x $ef         clear the intr flag
         rti
*
* end of source code for mtdas6.asm
```

Appendix B: Personal Computer Software Source Code

This code is written in the "C" programming language and can be compiled to an executable file by Borlands's Turbo C version 2 using the small memory model runtime library.

```
/* MT-TST4.C  loads nsamp samples, nchan chans, at stime */
/* per sample without pacing of the ram dump.   Setup    */
/* parameters can be selected.                           */
/* displ result on graphics screen, waits for kbhit();   */
/* note: must have egavga.bgi in curr dir when running   */

#include <dos.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>

/*  lobyte, hibyte: rec'd DAS data  ch: keybrd char      */
/*  k,m,n loop: counters  quit: boolean to exit routine  */
/*  status: RS232 status byte  dat: converted DAS result */
unsigned char lobyte, hibyte, ch;
unsigned char nchan=0x04;       /* number of chan        */
unsigned int  k, m, n, quit, status, dat;
unsigned int  nsamp=0x0200;     /* number of samples     */
unsigned int  stime=0x007D;     /* intersample time      */
unsigned int  rdat[8192];       /* storage for rec'd data */
float tmp;                    /* temp for conversion     */
union REGS regs;              /* BIOS register vars      */
int gdriver=EGA;              /* EGAVGA vars for         */
int gmode=EGAHI;              /*  video mode             */

unsigned int Init232(void);       /* init RS232 port      */
unsigned int Transmit(char);      /* send one byte        */
unsigned char Receive(void);      /* read RS232 port      */
unsigned int GetStatus(void);     /* read status          */
void BuildScreen(void);           /* setup graph display  */
void DataDisplay(void);           /* graph DAS results    */
unsigned char GetNChan(void);     /* interpret oper choice*/
unsigned int GetSTime(void);      /*   for acquisition    */
unsigned int GetNSamp(void);      /*   parameters         */
void BuildMenu(void);             /* setup main menu      */
void SendParams(void);            /* transmit DAS params  */

void main(void){
         quit=0;
         initgraph( &gdriver, &gmode,"" );
         Init232();
         while (!quit){
                 restorecrtmode();
                 _setcursortype(0);
                 BuildMenu();
                 SendParams();
                 ch=getch();  if (!ch) getch();
                 ch=toupper(ch);
                 switch(ch){
                 case 'D': { status=GetStatus()&0x0100;
```

```
                        while (status) { Receive();
                        status=GetStatus()&0x0100; }
                        setgraphmode(EGAHI);
                        BuildScreen();
                        Transmit('A');
                        DataDisplay();
                        while (!kbhit());
                        ch=getch(); if (!ch) getch();
                        _setcursortype(0);
                        break; }
                case 'S': { nsamp=GetNSamp();
                        SendParams();
                        break; }
                case 'R': { stime=GetSTime();
                        SendParams();
                        break; }
                case 'N': { nchan=GetNChan();
                        SendParams();
                        break; }
                case 'Q': { quit=1; break; }
                default : {textcolor(14); gotoxy(50,4);
                        cputs("D S M N Q only!");
                        delay(2000);
                        gotoxy(30,4);cputs("      ");}
                }     /* end switch */
        }   /* end while */
        _setcursortype(2);
        return;
        }   /* end main */

unsigned int Init232(void){
regs.h.ah=0;                /* initialize comm port */
regs.h.al=0xE3;            /*9600 baud, no parity, 1 stop, */
                            /*  8 data bits   */
regs.x.dx=0;                /*  comm port 1 */
int86( 0x14, &regs, &regs );
return regs.x.ax;
}

unsigned int Transmit( char dat ){
regs.h.ah=1;                /* send one character */
regs.h.al=dat;              /* char to send */
regs.x.dx=0;                /* comm port 1 */
int86( 0x14, &regs, &regs );
return regs.x.ax;
}

unsigned char Receive(void){
regs.h.ah=2;
regs.x.dx=0;
int86( 0x14, &regs, & regs );
return regs.h.al;
}

unsigned int GetStatus(void){
regs.h.ah=3;                /* get status word */
```

```
regs.x.dx=0;              /* comm port 1 */
int86( 0x14, &regs, &regs );
return regs.x.ax;
}

void BuildScreen(void) {
char chnum[17];           /* strings to hold DAS params */
char snum[17];
char ratenum[17];
         setcolor(15);
         line(50,10,50,310); line(50,160,562,160);
         for (k=0; k<11; k++)
                 line(45,10+k*30,55,10+k*30);
         for (k=0; k<9; k++)
                 line(50+64*k,155,50+64*k,165);
         itoa(nsamp/4,snum,10);
         outtextxy(170,168,snum);
         itoa(nsamp/2,snum,10);
         outtextxy(298,168,snum);
         itoa(3*nsamp/4,snum,10);
         outtextxy(426,168,snum);
         itoa(nsamp,snum,10);
         outtextxy(554,168,snum);
         outtextxy(25,  6," 5");
         outtextxy(25, 36," 4");
         outtextxy(25, 66," 3");
         outtextxy(25, 96," 2");
         outtextxy(25,126," 1");
         outtextxy(25,156," 0");
         outtextxy(25,186,"-1");
         outtextxy(25,216,"-2");
         outtextxy(25,246,"-3");
         outtextxy(25,276,"-4");
         outtextxy(25,306,"-5");
         for (k=0; k<nchan; k++){
                 if (k<4) setcolor(k+10); else
                         setcolor(k-2);
                 outtextxy( 150+50*k,320,"CH");
                 itoa(k+1,chnum,10);
                 outtextxy(170+50*k,320,chnum);
                 }
         setcolor(15);
         outtextxy(64,0,"#chan=");
         outtextxy(152,0,"#samples=");
         outtextxy(292,0,"mS/sample=");
         itoa(nchan,chnum,10);  outtextxy(116,0,chnum);
         itoa(nsamp,snum,10);  outtextxy(224,0,snum);
         sprintf(ratenum,"%.1f",0.008*stime);
         outtextxy(376,0,ratenum);
         setcolor(14);
         outtextxy(200,330,"Press any key \
for the main menu");
         return;
         }
```

```
void DataDisplay(void){
int color, x;           /* display of chan num, horiz loc */
int ndx;         /* index in data array           */
        for (n=0; n<nsamp; n++) {
                for (m=0; m<nchan; m++){
                        status=0;
                        while (!status)
                        status=GetStatus() & 0x0100;
                        lobyte=Receive();
                        status=0;
                        while (!status)
                        status=GetStatus() & 0x0100;
                        hibyte=Receive();
                        ndx=m + nchan*n;
                        rdat[ndx]=lobyte + hibyte<<8;
                        tmp=(float)(hibyte*256+lobyte);
                        dat=310-(int)(tmp*0.073);
                        if ( nsamp==256 )  x=2*n;
                        if ( nsamp==512 )  x=n;
                        if ( nsamp==1024  ) x=n/2;
                        if (m<4) color=10+m; else
                                color=m-2;
                        putpixel(x+50,dat,color);
                        }   /* for m */
                if ( kbhit() ) return;
                if ( stime>0x1800 ) Transmit('C');
                }    /* for n */
        return;
        }

unsigned int GetNSamp(void){
int done=0;
        textcolor(15);
        gotoxy(10,4); putch('D');  gotoxy(10,6);
                putch('S');
        gotoxy(10,8); putch('R');  gotoxy(10,10);
                putch('N');
        gotoxy(10,12); putch('Q');
        textcolor(10);
        gotoxy(50,6); cputs("A  256  samples");
        gotoxy(50,7); cputs("B  512  samples");
        gotoxy(50,8); cputs("C  1024 samples");
        textcolor(12);
        gotoxy(50,6);  putch('A');  gotoxy(50,7);
                putch('B');
        gotoxy(50,8); putch('C');
        while (!done){
                ch=getch();  if (!ch) ch=getch();
                ch=toupper(ch);
                switch(ch){
                case 'A': {nsamp=256; done=1; break; }
                case 'B': {nsamp=512; done=1; break; }
                case 'C': {nsamp=1024; done=1; break;}
                default : { textcolor(14);
                        gotoxy(50,11);
                        cputs("A-C only!");
```

```
                    break; }
            case 'H': { stime=0x30D4;  done=1;
                    break; }
            case 'I': { stime=0x61A8;  done=1;
                    break; }
            case 'J': { stime=0xF424;  done=1;
                    break; }
            default : {  textcolor(14);
                    gotoxy(50,16);
                    cputs("A-G only!");
                    delay(2000);
                    gotoxy(50,16); cputs("     ");}
            }   /* end switch */
        }  /* end while */
        return stime;
        }

unsigned char GetNChan(void){
int done=0;
        textcolor(15);
        gotoxy(10,4);  putch('D');
        gotoxy(10,6);  putch('S');
        gotoxy(10,8);  putch('R');
        gotoxy(10,10); putch('N');
        gotoxy(10,12); putch('Q');
        textcolor(10);
        gotoxy(50,10);  cputs("A  Four  Channels");
        gotoxy(50,11);  cputs("B  Eight Channels");
        textcolor(12);
        gotoxy(50,10);  putch('A');
        gotoxy(50,11);  putch('B');
        while (!done){
            ch=getch();  if (!ch) getch();
            ch=toupper(ch);
            switch(ch){
            case 'A': { nchan=4;  done=1; break; }
            case 'B': { nchan=8;  done=1; break; }
            default : {  textcolor(14);
                    gotoxy(50,14);
                    cputs("A,B only!");
                    delay(2000);
                    gotoxy(50,14); cputs("     ");}
            }  /* end switch */
        }  /* end while */
        return nchan;
        }

void BuildMenu(void){
        textcolor(15);  textbackground(1);
        clrscr();
        gotoxy(35,2);  cputs("Main Menu");
        gotoxy(10,4);
        cputs("Data Acquisition and Display");
        gotoxy(10,6);  cputs("Sample Size -- SetUp");
        gotoxy(10,8);
        cputs("Rate of Sampling -- SetUp");
```

```
                        delay(2000);
                        gotoxy(50,11); cputs("      ");}
                } /* end switch */
        } /* end while */
        return nsamp;
        }

unsigned int GetSTime(void) {
int done=0;
        textcolor(15);
        gotoxy(10,4); putch('D');  gotoxy(10,6);
        putch('S');
        gotoxy(10,8); putch('R');  gotoxy(10,10);
        putch('N');
        gotoxy(10,12); putch('Q');
        gotoxy(45,6); cputs("Fill Ram");
        gotoxy(60,6); cputs("Immediate");
        textcolor(10);
        gotoxy(45,8);  cputs("A  2000 Hz");
        gotoxy(45,9);  cputs("B  1000 Hz");
        gotoxy(45,10);  cputs("C   500 Hz");
        gotoxy(45,11);  cputs("D   200 Hz");
        gotoxy(45,12);  cputs("E   100 Hz");
        gotoxy(45,13);  cputs("F    50 Hz");
        gotoxy(60,8);  cputs("G  20 Hz");
        gotoxy(60,9);  cputs("H  10 Hz");
        gotoxy(60,10); cputs("I   5 Hz");
        gotoxy(60,11); cputs("J   2 Hz");
        textcolor(12);
        gotoxy(45,8);  putch('A');
        gotoxy(45,9);  putch('B');
        gotoxy(45,10); putch('C');
        gotoxy(45,11); putch('D');
        gotoxy(45,12); putch('E');
        gotoxy(45,13); putch('F');
        gotoxy(60,8);  putch('G');
        gotoxy(60,9);  putch('H');
        gotoxy(60,10); putch('I');
        gotoxy(60,11);  putch('J');
        while (!done){
                ch=getch();  if (!ch) getch();
                ch=toupper(ch);
                switch(ch){
                case 'A': { stime=0x003E; done=1;
                        break; }
                case 'B': { stime=0x007D;  done=1;
                        break; }
                case 'C': { stime=0x00FA;  done=1;
                        break; }
                case 'D': { stime=0x0271;  done=1;
                        break; }
                case 'E': { stime=0x04E2;  done=1;
                        break; }
                case 'F': { stime=0x09C4;  done=1;
                        break; }
                case 'G': { stime=0x186A;  done=1;
```

```
        gotoxy(10,10);
        cputs("Number of Channels -- SetUp");
        gotoxy(10,12); cputs("Quit to DOS");
        textcolor(12);
        gotoxy(10,4);  putch('D');
        gotoxy(10,6);  putch('S');
        gotoxy(10,8);  putch('R');
        gotoxy(10,10); putch('N');
        gotoxy(10,12); putch('Q');
        textcolor(15);
        gotoxy(20,16); cputs("Current SetUp");
        gotoxy(10,17);  cputs("Sample Size   =");
        gotoxy(10,18);  cputs("Sample Rate   =");
        gotoxy(35,18);  cputs("millisec/sample");
        gotoxy(10,19);  cputs("Number of Chan=");
        textcolor(10);
        gotoxy(26,17);  cprintf("%i",nsamp);
        gotoxy(26,18);
        cprintf("%.1f", ((float)(stime))/125.0 );
        gotoxy(26,19);  cprintf("%i",nchan);
        return;
        }

void SendParams(void){
char dat;
        status=GetStatus() & 0x0100;
        while (status) { Receive();
                 status=GetStatus() & 0x0100; }
        Transmit('P');
        status=0;
        while ( !status ) status=GetStatus() & 0x4000;
        dat = stime & 0x00FF;
        Transmit( dat );
        status=0;
        while ( !status ) status=GetStatus() & 0x4000;
        dat = stime >> 8;
        Transmit( dat );
        status=0;
        while ( !status ) status=GetStatus() & 0x4000;
        dat = nsamp & 0x00FF;
        Transmit( dat );
        status=0;
        while ( !status ) status=GetStatus() & 0x4000;
        dat = nsamp >> 8;
        Transmit( dat );
        status=0;
        while ( !status ) status=GetStatus() & 0x4000;
        Transmit( nchan );
        return;
        }
```

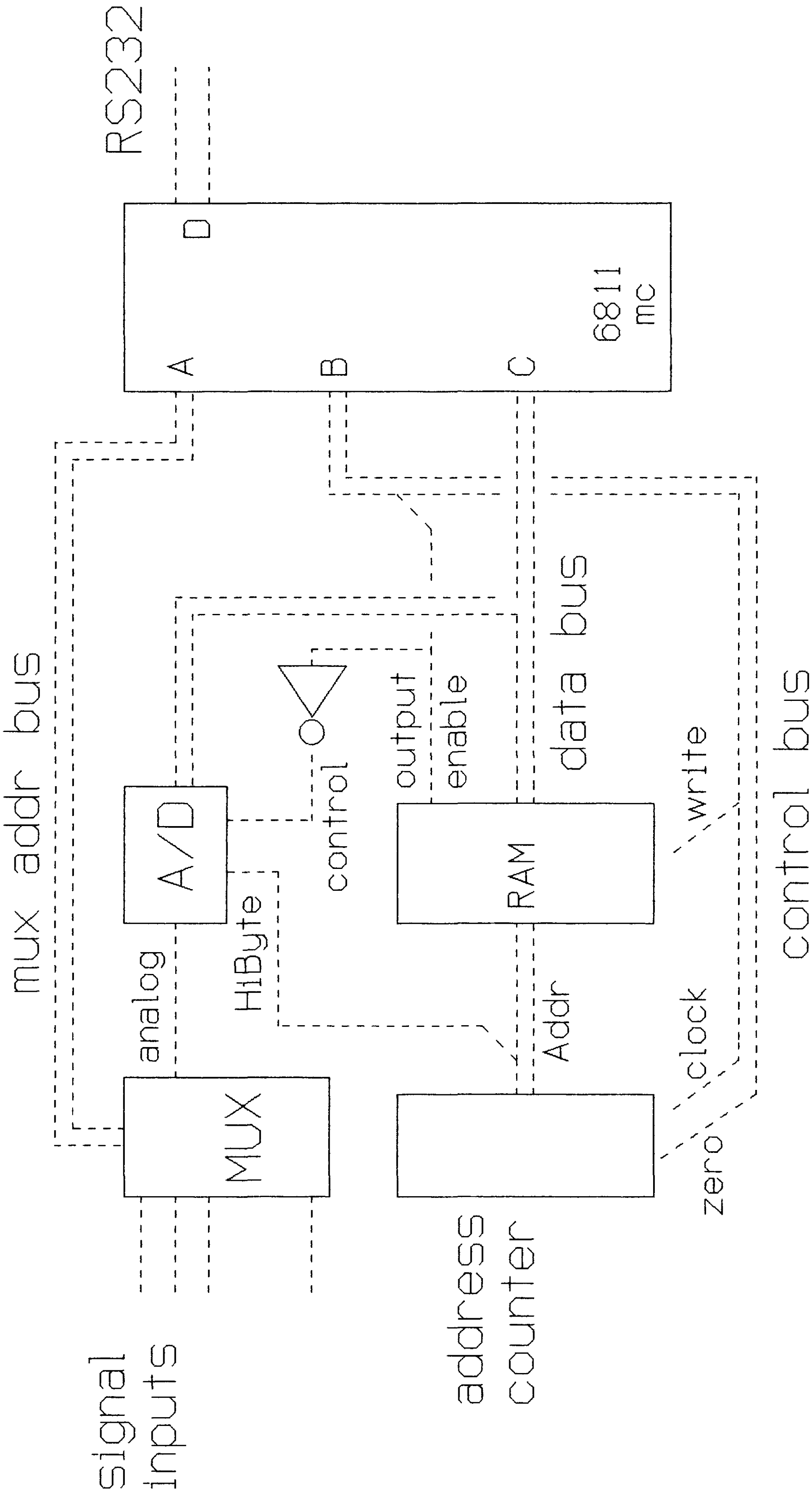

Figure 1 DAS block Diagram

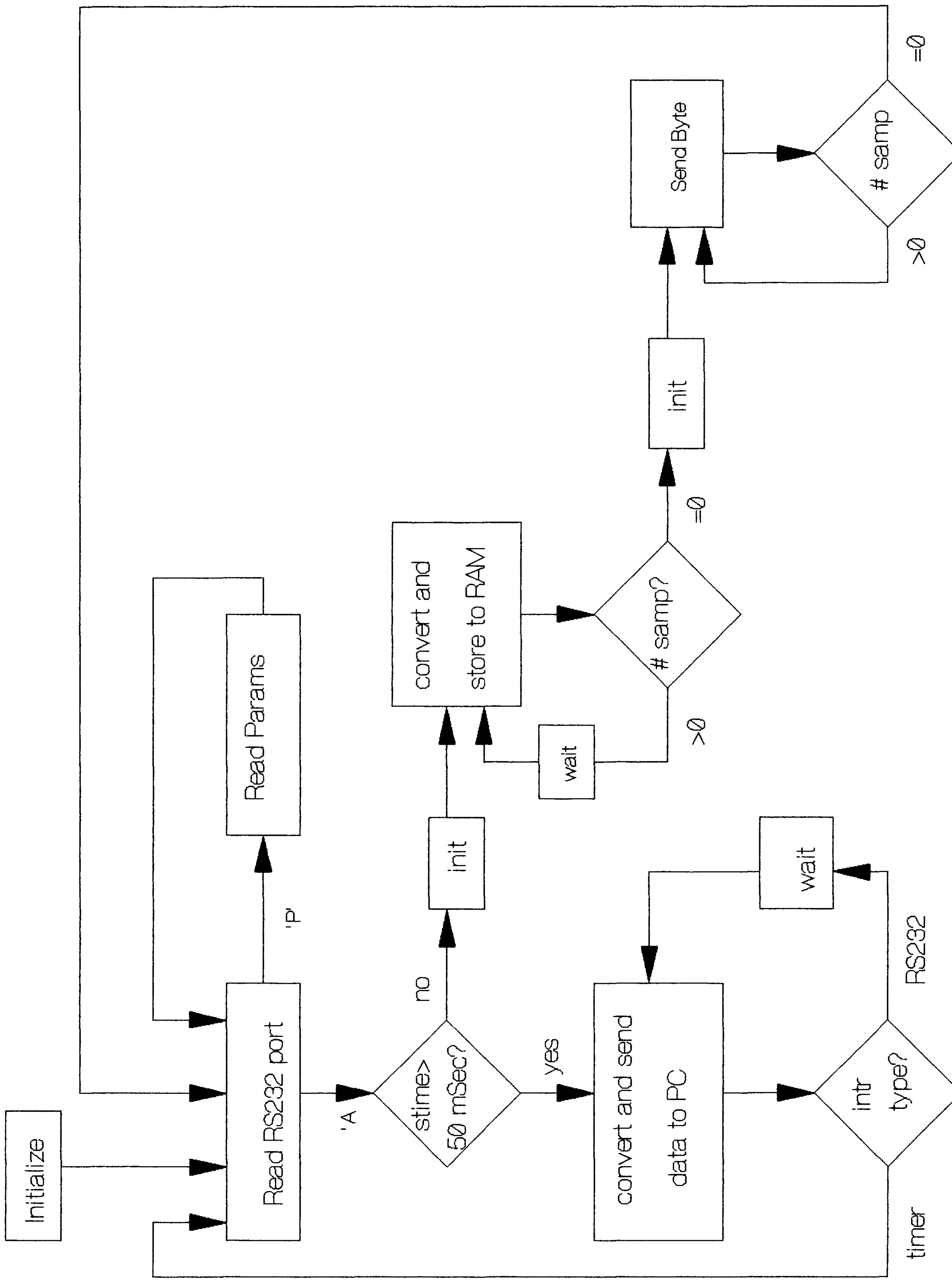

Figure 2 DAS Software Flow Chart

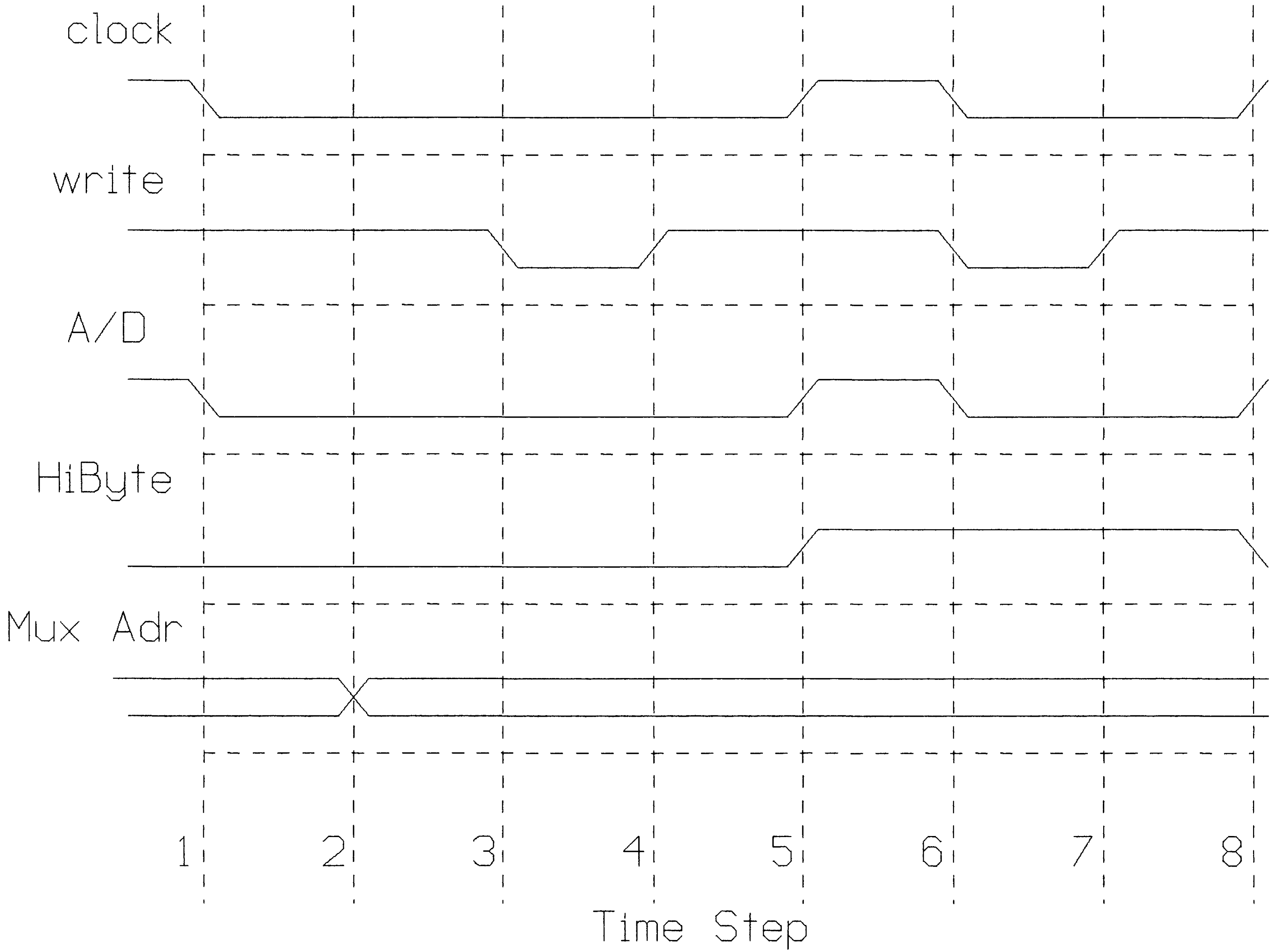

Figure 3 Timing Diagram

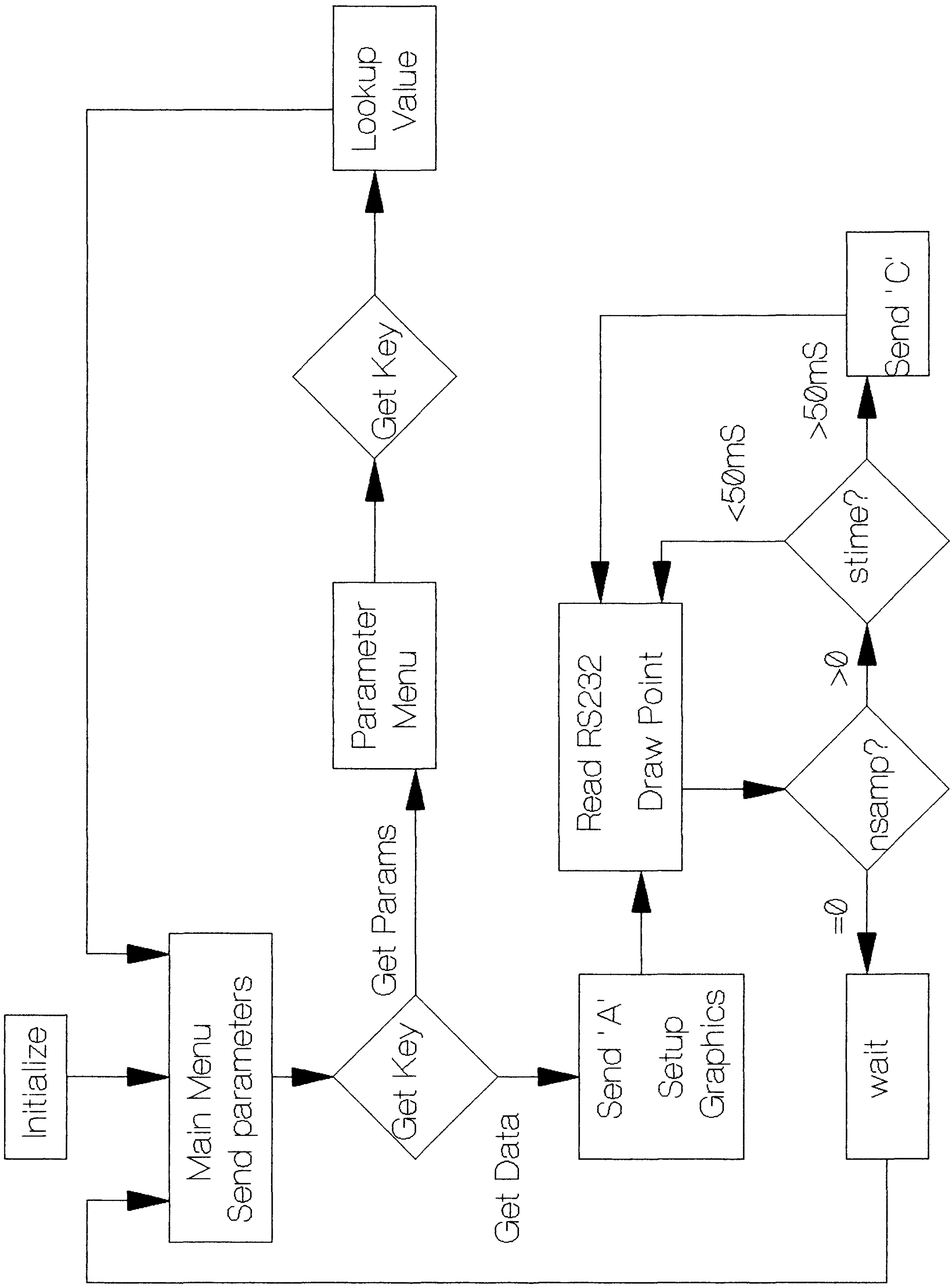

Figure 4 PC Software Flow Chart

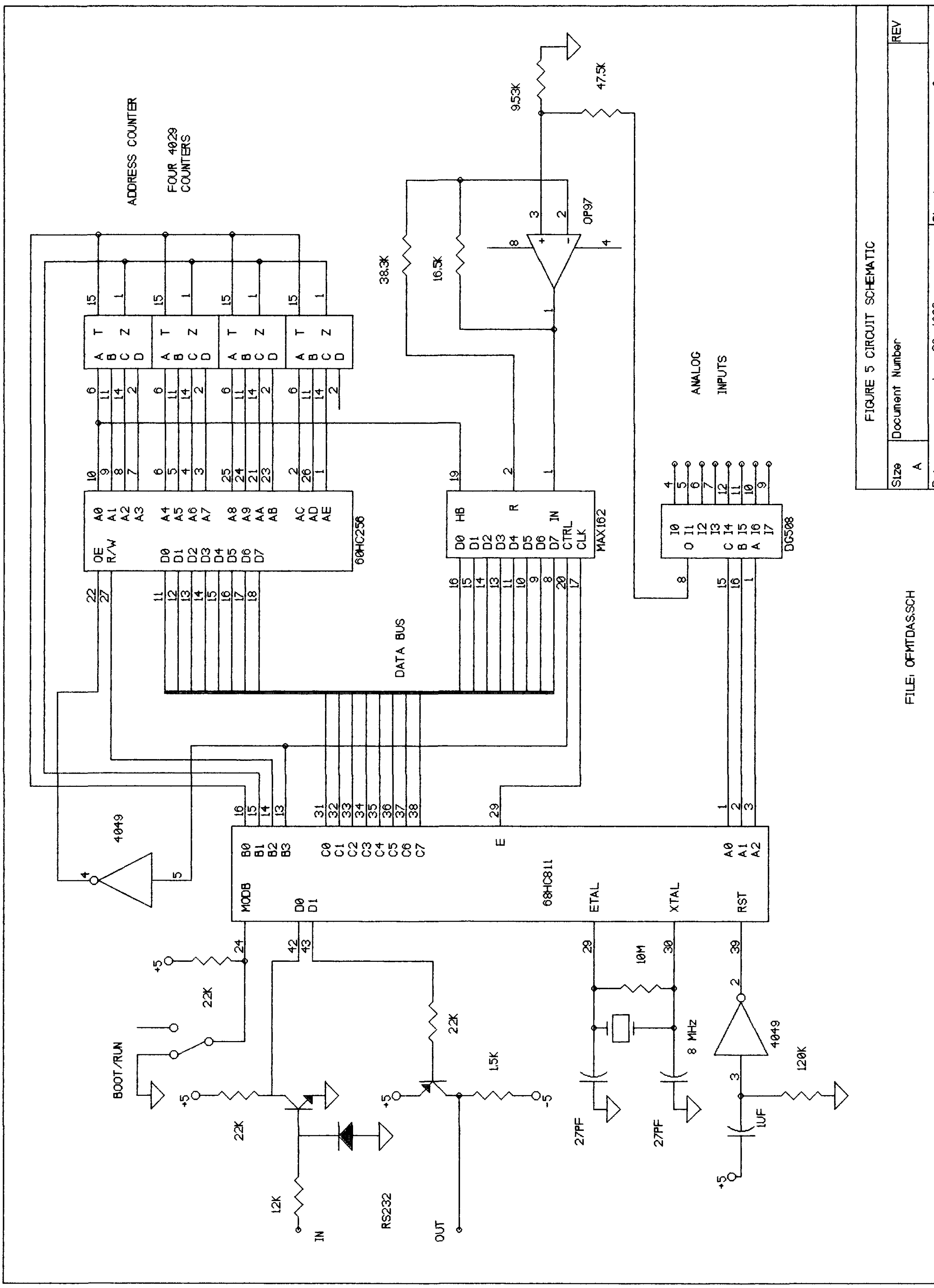

Figure 5 Circuit Schematic