Appendix B
EIT Data Acquisition Code `FASTEIT.BAS`

This program, written in Microsoft® QuickBasic™ (Microsoft Corporation, 1991), initializes the EIT system for operation with the internal electrode counters. After initializing the DT2839 data acquisition board, the code dynamically allocates buffers in the PC memory to store voltage measurements, then arms the EIT electronics. Multiplexers in the electronics select the electrodes that inject and ground current, and also select the electrodes whose voltages will be measured by the data acquisition card. This version of the code activates a series of internal counters that control the multiplexers, so that the card is only responsible for measuring electrode voltages. All voltages are stored in the buffers until all projection sets are collected, then the voltage values are transferred to arrays for averaging calculations.

Several subroutines from the Data Translation SP0131 Software Toolkit (Data Translation, Inc., 1994) are used to initialize the data acquisition board, issue commands to the EIT electronics, and acquire voltage measurements. These subroutines are proprietary and are not listed here, but may be recognized by the prefix "`dt.`" For example, the subroutine that opens communications with the DT2839 board is called by the statement:

```
istat = dt.initialize(SADD(idrivername$), VARSEG(idrivername$), ihandle).
```

The arguments in parentheses may be instructions to the board or data returned by the board; a nonzero value of `istat` identifies an error in execution.

```
'***
'***  Steve and Darin's FAST EIT code
'***    Written 7/97-12/97 by dlg and slc
'***    Electrode selection is "hard-wired" through IC counters;
'***    data acquisition board only starts counters and collects
'***    voltage data.  To conserve memory, dynamic allocation is
'***    used.  Arrays are dimensioned when first used and deallocated
'***    when they are no longer needed.
'***

'$DYNAMIC

DECLARE SUB exit.error (status AS INTEGER, message AS STRING)
DECLARE SUB cleanup (dummy)
DECLARE SUB eistats (elec%, proj%, slength%, carrsum&(), quadsum&(), Vsum!(), resp$)

'*********************************************************************
' Definition files for extended memory manager, error codes, counter/
'   timer subroutines and software tools; these files define several
'   constants used in subroutine calls, which are found in ALL CAPS in
'   the code
```

```
'$INCLUDE: 'd:\toolkit\dtst_xmm.bi'
'$INCLUDE: 'd:\toolkit\dtst_err.bi'
'$INCLUDE: 'd:\toolkit\dtst_ctr.bi'
'$INCLUDE: 'd:\toolkit\dtst_tls.bi'

DIM digitalio AS DIGITALIOSTRUCT
DIM sap AS SETACQPARAMSSTRUCT
DIM board AS BOARDSTRUCT

DEFINT I-N
DEFSNG A-H
DEFLNG O-Z

CONST slength% = 16       ' number of oversamples on each channel

CONST currlatch% = 0      ' addresses of counters, muxes, etc., and
CONST voltlatch% = 32     '   commands to arm them
CONST gainlatch% = 64
CONST fastflag% = 1
CONST enable% = 16

CLS
PRINT
PRINT "This EIT program arms the internal counters, which select the current,"
PRINT "ground and measurement electrodes.  The A/D card records the output"
PRINT "of the demodulators when triggered by the EIT electronics."
PRINT

DO
  INPUT "Enter the number of electrodes (8 or 16): ", elec%
  IF elec% <> 8 AND elec% <> 16 THEN
    PRINT "Incorrect input."
  END IF
LOOP UNTIL elec% = 8 OR elec% = 16

resetcntr% = 96 + (16 - elec%) / 4       ' commands to reset and start
startcntr% = 128 + (16 - elec%) / 4 + 1  '   hardware counters with
                                         '   desired number of electrodes

'***********************************************************************
' Compute quantities for memory and buffer allocation

icombs = elec% ^ 2 * (elec% - 1) / 2  ' number of current, ground and
                          '   measurement combinations in one projection
                          '   set; one scan of the channel/gain list is
                          '   made for each electrode combination

icglsize = 2 * slength%   ' size of channel/gain list in number of samples;
                          '   list is scanned once for each electrode
                          '   combination, and specifies two channels
                          '   (Ch. 0 = carrier, Ch. 1 = quadrature)

isamptot& = CLNG((icombs - 1)) * CLNG(icglsize) ' total samples in one
                          '   projection set, minus one electrode combination
                          '   (in fast mode, system never sends a trigger
                          '   pulse for the final combination)

iblksize = 32             ' number of kbytes per extended memory block
                          '   (this number found to work best through
                          '   trial and error; required to be a multiple
                          '   of 2)
```

```
DO

'********************************************************************
' Initialize system for data acquisition

  INPUT "Enter the number of consecutive projections (1 to 100): ", proj%
  IF proj% < 1 THEN proj% = 1
  IF proj% > 100 THEN proj% = 100

  INPUT "Do you wish to change the amplifier gain from the default (Y/N) (default N)?
       ", gainset$

  IF gainset$ = "Y" OR gainset$ = "y" THEN

    PRINT "Input the index for the amplifier gain:"
    PRINT "   N = 0       Gain =    1"
    PRINT "       1                10"
    PRINT "       2               100"
    PRINT "       3              1000"
    INPUT "Recommended value is N = 1 (default). ", igainpwr

    IF igainpwr < 0 OR igainpwr > 3 THEN igainpwr = 1

  ELSE

    igainpwr = 1

  END IF

  gain% = 32 * igainpwr

'********************************************************************
' Initialize and reset the DT283X board
'*** SADD and VARSEG return addresses, not string expresions themselves

  idrivername$ = "DT283X$0" + CHR$(0)
  iunit = 0       'Board is unit 0

  istat = dt.initialize(SADD(idrivername$), VARSEG(idrivername$), ihandle)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.initialize")

  istat = dt.ct31.initialize(ihandle)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.ct31.initialize")

  istat = dt.reset(iunit, ihandle)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.reset")

'********************************************************************
' Set the board parameters

  board.dmachannel1 = 5            'DMA channel 1 selection
  board.dmachannel2 = 6            'DMA channel 2 selection
  board.interruptlevel = 10        'interrupt level
  board.boardtimeout = 10          '10-second wait before timeout error returned
'  board.boardtimeout = 0           'infinite patience
  board.adsetupbits = ADTPOL + ADTRIGGER + ADSCANENABLE
                              'a-to-d setup codes, external triggered
                              '  scan (see 'dtst_tls.bi' for codes)
  board.dasetupbits = NODACS       'd-to-a setup code (no d-to-a)

  istat = dt.set.board(iunit, ihandle, board)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.set.board")
```

```
'*********************************************************************
' Create extended memory blocks and buffers

  isection = ADSECTION

  istat = dt.xm.initialize
  IF istat <> 0 THEN CALL exit.error(istat, "dt.xm.initialize")

' Select trial number of data samples per buffer (two bytes per sample;
'   best initial guess for buffer size arrived at by trial and error)

  IF elec% = 16 THEN
    isamptrial = 4 * 1024
  ELSE
    isamptrial = 2 * 1024
  END IF

' Allocate arrays:
'   ixmhdl() = extended memory (XM) block handle
'   ihdl() = buffer handle
'   xmaddr&() = memory address of XM block
'   isamp(i) = number of samples in buffer i
'   ibuff(,) = conventional memory array to receive data from XM buffers

  DIM ixmhdl(1 TO 16), ihdl(1 TO 16), xmaddr&(1 TO 16), isamp(1 TO 16),
      ibuff(isamptrial, 1 TO 16)

  ihdlcnt = 1             ' tally of buffer handles
  isampcnt& = 0           ' tally of samples for which buffer space has been
                          '   allocated

  isamp(ihdlcnt) = isamptrial

' Begin allocation

  DO

' Allocate extended memory block, get its info, and lock it

    istat = dt.xm.allocate.block(iblksize, ixmhdl(ihdlcnt))
    IF istat <> 0 AND istat <> 24579 THEN CALL exit.error(istat,
      "dt.xm.allocate.block")

    istat = dt.xm.get.emb.info(ixmhdl(ihdlcnt), ilockcnt, ifrembs, iblklen) '***
      diagnostic
'    PRINT "Lock flag, remaining free XM blocks, block length in kbytes, istat =" '***
      diagnostic
'    PRINT ilockcnt, ifrembs, iblklen, istat '*** diagnostic

    istat = dt.xm.lock.block(ixmhdl(ihdlcnt), xmaddr&(ihdlcnt))
    IF istat <> 0 THEN CALL exit.error(istat, "dt.xm.lock.block")

' Create a single buffer from the extended memory block; if the buffer
'   crosses a DMA page boundary, recreate the buffer using the maximum
'   buffer size that fits within the page

    istat = dt.xm.create.buffer(iunit, isection, isamp(ihdlcnt), xmaddr&(ihdlcnt),
      ihdl(ihdlcnt))
    IF istat = 12294 THEN
      isamp(ihdlcnt) = isamp(ihdlcnt) - (isamp(ihdlcnt) MOD icglsize)
      istat = dt.xm.create.buffer(iunit, isection, isamp(ihdlcnt), xmaddr&(ihdlcnt),
       ihdl(ihdlcnt))
      IF istat <> 0 THEN
      PRINT "Error during re-creation of buffer #"; ihdlcnt
      CALL exit.error(istat, "dt.xm.create.buffer")
     END IF
   ELSEIF istat <> 0 THEN
     PRINT "Error during creation of buffer #"; ihdlcnt
     CALL exit.error(istat, "dt.xm.create.buffer")
   END IF
'    PRINT "Buffer "; ihdlcnt; " created with "; isamp(ihdlcnt); " samples" '***
      diagnostic

   isampcnt& = isampcnt& + CLNG(isamp(ihdlcnt))
'    PRINT "Total samples allocated = "; isampcnt& '*** diagnostic
'    INPUT resp$

' Choose a trial size for the next buffer, based on number of samples
'    still to be allocated

   IF isampcnt& < isamptot& AND (isamptot& - isampcnt&) >= CLNG(isamptrial) THEN
     ihdlcnt = ihdlcnt + 1
     isamp(ihdlcnt) = isamptrial
   ELSEIF isampcnt& < isamptot& AND (isamptot& - isampcnt&) < CLNG(isamptrial) THEN
     ihdlcnt = ihdlcnt + 1
     isamp(ihdlcnt) = CINT(isamptot& - isampcnt&)
   ELSEIF isampcnt& = isamptot& THEN
'      PRINT "All buffers created" '*** diagnostic
   ELSE
     PRINT "Error in creation of buffers: samples created, samples required = "
     PRINT isampcnt&, isamptot&
     CALL exit.error(666, "MAIN")
   END IF

  LOOP UNTIL isampcnt& >= isamptot&

'*** diagnostic print of XM block info

'  PRINT "XM block", "handle" '***
'  FOR iii = 1 TO ihdlcnt '***
'    PRINT iii, ixmhdl(iii) '***
'  NEXT iii '***
'  INPUT resp$

'*********************************************************************
' Create a channel/gain list (since the board can apply a different
'    gain to each channel)
'
'       DT2839 channel gains:   gain    igain()=
'                                 1       0
'                                 2       1
'                                 4       2
'                                 8       3
'
'    Board output range is set at +/- 10V.

  DIM ichan(icglsize - 1)              ' channel array for channel/gain table
  DIM igain(icglsize - 1)              ' gain array for channel/gain table
'  PRINT "Channel/gain table dimensioned..." '*** diagnostic

  FOR jj = 1 TO slength%
    ichan(2 * jj - 2) = 0: ichan(2 * jj - 1) = 1
    igain(2 * jj - 2) = 0: igain(2 * jj - 1) = 0
  NEXT jj

' Subroutine 'dt.create.cgl' expects ichan() and igain() arrays to be
'   passed by reference, not by value, so last two arguments must be the
'   first elements in each array

  istat = dt.create.cgl(icglsize, ichan(0), igain(0))
  IF istat <> 0 THEN CALL exit.error(istat, "dt.create.cgl")

'**********************************************************************
' Set the acquisition parameters
'*** Constants defined in library 'dtst_tls.bi' --- dlg, 7/97

' Clock rate must be reduced from 416 kHz to 320 kHz when channel gain > 1.
'   If value of arate is not a possible conversion rate, driver rounds up
'   to the next highest possible rate.

  IF igain(0) = 0 AND igain(1) = 0 THEN
    arate = 400000  ' rounded up to 416000
  ELSE
    arate = 300000  ' rounded up to 320000
  END IF

  digitalio.command = SETXFER
  digitalio.direction = DIOOUTPUT

  sap.section = isection
  sap.transfertype = BDUALTRIGSCAN
  sap.clockrate = arate
  sap.cutoff = 0

  istat = dt.set.acq(iunit, ihandle, isection, sap)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.set.acq")

'**********************************************************************
' Initialize raw data arrays

  DIM carrsum&(1 TO (elec% - 1), 2 TO elec%, 1 TO elec%) ' sums of carrier voltages
  DIM quadsum&(1 TO (elec% - 1), 2 TO elec%, 1 TO elec%) ' sums of quadrature voltages
  DIM Vsum!(1 TO 100, 1 TO elec%) ' sums of off-centerline electrode voltages
                              '   for 180-degree injection/ground cases
                              '   indexed by projection and relative
                              '   location
'  PRINT "carrsum&, quadsum&, Vsum! dimensioned..." '*** diagnostic

  FOR icurrent = 1 TO elec% - 1         'icurrent = index of current injection
      electrode
    FOR iground = icurrent + 1 TO elec% 'iground = index of current return electrode
      FOR ivolt2 = 1 TO elec%           'ivolt2 = index of voltage meas. electrode
       carrsum&(icurrent, iground, ivolt2) = 0
       quadsum&(icurrent, iground, ivolt2) = 0
      NEXT ivolt2
    NEXT iground
  NEXT icurrent

  FOR i = 1 TO proj%
    FOR ivolt2 = 1 TO elec%
      Vsum!(i, ivolt2) = 0!
    NEXT ivolt2
  NEXT i

'**********************************************************************
'
' Arrangement of ports which communicate to the EIT electronics in
'   fast mode:
'
```

```
' Port 0 Bit:  00      01      02      03      04      05      06      07
'             |count  | 16/8 |  not used     | mux   |  latch address,       |
'             |start  | flag |               |enable| counter controls      |
'
' Port 1 Bit:  10      11      12      13      14      15      16      17
'             | slow/|      not used         | mux   | PGA202 gain | not   |
'             | fast |                       |enable|             | used  |
'             | flag |
'
' To send instructions to the correct mux, the latch address I/O lines
'   must be set as follows:
'
'                                     Port 0 Bit:   07   06   05   In decimal
' latch current injection & ground counters  L    L    L       0
' latch voltage measurement counters         L    L    H      32
' latch gain mux                             L    H    L      64
' reset electrode counters                   L    H    H      96
' start electrode counters                   H    L    L     128
'
'   To enable the current muxes, bit 04 must be set high (16 in decimal)
'   as the current counters are latched; likewise for bit 14 and the
'   voltage muxes.  Bit 10 must be set high (1 in decimal) to start the
'   fast electrode counters.
'
' To select the number of electrodes, the following bit must be set
'   at the same time the electrodes counters are reset and started:
'
' Port 0 Bit:      01     In decimal   Electrodes
'                   L          0           16
'                   H          2            8
'
' To set the PGA202 amplifier gain, I/O lines must be set as follows:
'
' Port 1 Bit:      16     15     In decimal   Gain
'                   L      L          0          1
'                   L      H         32         10
'                   H      L         64        100
'                   H      H         96       1000
'
'*****

' Send commands to set gain mux and activate fast hardware counters.  The
'   'dt.ct31.gate.delay' command sends a pulse on CLK1 to "latch" the
'   gain mux (i.e., prompt it to accept instructions).

  digitalio.dioport = PORT0
  digitalio.diovalue = gainlatch%
  istat = dt.set.dio(iunit, ihandle, digitalio)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.set.dio (PGA202)")

  digitalio.dioport = PORT1
  digitalio.diovalue = fastflag% + gain%
  istat = dt.set.dio(iunit, ihandle, digitalio)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.set.dio (gain value)")

  istat = dt.ct31.gate.delay(iunit, 0, 0, 1, 2)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.ct31.gate.delay (PGA202)")

' Send commands to latch current counters and enable current muxes

  digitalio.dioport = PORT0
  digitalio.diovalue = currlatch% + enable%
  istat = dt.set.dio(iunit, ihandle, digitalio)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.set.dio (current mux PORT 0)")

  digitalio.dioport = PORT1
  digitalio.diovalue = enable%
  istat = dt.set.dio(iunit, ihandle, digitalio)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.set.dio (current mux PORT 1)")

  istat = dt.ct31.gate.delay(iunit, 0, 0, 1, 2)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.ct31.gate.delay (current muxes)")

' Send commands to latch voltage counters and enable voltage muxes

  digitalio.dioport = PORT0
  digitalio.diovalue = voltlatch% + enable%
  istat = dt.set.dio(iunit, ihandle, digitalio)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.set.dio (voltage mux PORT 0)")

  digitalio.dioport = PORT1
  digitalio.diovalue = enable%
  istat = dt.set.dio(iunit, ihandle, digitalio)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.set.dio (voltage mux PORT 1)")

  istat = dt.ct31.gate.delay(iunit, 0, 0, 1, 2)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.ct31.gate.delay (voltage muxes)")

'****************** START OF DATA ACQUISITION LOOP ******************

  CLS
  PRINT "Acquiring projections...": PRINT

'  OPEN "d:\data\diagnose.dat" FOR OUTPUT AS #2 '*** diagnostic dump file

  FOR projloop% = 1 TO proj%

' Get clock time at start of projection routine

    starttime! = TIMER

' Send command to reset address counters (LS163's) and choose number of
'   electrodes.  The 'dt.ct31.gate.delay' call sends a pulse on CLK1 to
'   latch the LS163's.

    digitalio.dioport = PORT0
    digitalio.diovalue = resetcntr%
    istat = dt.set.dio(iunit, ihandle, digitalio)
    IF istat <> 0 THEN CALL exit.error(istat, "dt.set.dio (reset counters)")

    istat = dt.ct31.gate.delay(iunit, 0, 0, 1, 2)
    IF istat <> 0 THEN CALL exit.error(istat, "dt.ct31.gate.delay (reset counters)")

' Reset A/D buffers

    FOR iii = 1 TO ihdlcnt
      istat = dt.reset.buffer(ihdl(iii))
      IF istat <> 0 THEN CALL exit.error(istat, "dt.reset.buffer")
    NEXT iii

' Start fast hardware counters

    digitalio.dioport = PORT0
    digitalio.diovalue = startcntr%
    istat = dt.set.dio(iunit, ihandle, digitalio)
    IF istat <> 0 THEN CALL exit.error(istat, "dt.set.dio (start counters)")
    istat = dt.ct31.gate.delay(iunit, 0, 0, 1, 2)
    IF istat <> 0 THEN CALL exit.error(istat, "dt.ct31.gate.delay (start counters)")

' Start the A/D process in the background

    istat = dt.start.acq(iunit, ihandle, isection)
'    PRINT " acq istat   ", istat '*** diagnostic
    IF istat <> 0 AND istat <> 1 THEN
      istat2 = dt.stop.acq(iunit, ihandle, isection)
      CALL exit.error(istat, "dt.start.acq")
    END IF

' Wait until all buffers are inactive; exit on any unexpected error code

'    prevhdl% = 0 '*** diagnostic setup

    DO
      istat = dt.check.buffer(iunit, isection, bufhdl%, bufstat%)

'      IF bufhdl% <> prevhdl% OR istat <> 1 THEN '*** diagnostic prints
'        PRINT #2, istat, bufhdl%, bufstat% '***
'        prevhdl% = bufhdl% '***
'      END IF '***

      IF istat > 1 AND istat <> 3 THEN
       istat2 = dt.stop.acq(iunit, ihandle, isection)
       CALL exit.error(istat, "dt.check.buffer")
      END IF
    LOOP UNTIL istat = 3

'    PRINT #2, "Buffer I/O complete" '*** diagnostic

' Release all buffers for transfer to conventional memory

    istat = dt.wait.buffer(iunit, isection, bufhdl%)
    IF istat <> 0 THEN CALL exit.error(istat, "dt.wait.buffer")

'    INPUT resp$

' Transfer buffer contents to DOS-accessible conventional memory; note
'   that buffers are used Last-In, First-Out

'    PRINT "Ready to transfer buffer to DOS memory..." '*** diagnostic
    FOR iii = ihdlcnt TO 1 STEP -1
      istat = dt.xm.move.xm.to.dos(ixmhdl(iii), 0&, ibuff(0, iii), CLNG(isamp(iii) *
       2))
'      PRINT "Buffer "; iii; " moved to DOS"'*** diagnostic
      IF istat <> 0 THEN CALL exit.error(istat, "dt.xm.move.xm.to.dos")
    NEXT iii

'*** diagnostic disk dump of averages from each channel/gain scan
'    FOR iii = ihdlcnt TO 1 STEP -1
'      FOR jjj = 0 TO isamp(iii) - 1 STEP icglsize
'        ocave& = 0: oqave& = 0
'        FOR kkk = 0 TO slength% - 1
'          ocave& = ocave& + ibuff(((2 * kkk) + jjj), iii)
'          oqave& = oqave& + ibuff(((2 * kkk + 1) + jjj), iii)
'        NEXT kkk
'        PRINT #2, iii, jjj, ocave& / CLNG(slength%), oqave& / CLNG(slength%)
'      NEXT jjj
'    NEXT iii
```

```
'*** diagnostic dump of raw contents of buffers (disk space hog!)
'    FOR iii = ihdlcnt TO 1 STEP -1
'      FOR jjj = 0 TO isamp(iii) - 1 STEP 2
'        PRINT #2, iii, jjj, ibuff(jjj, iii), ibuff(jjj + 1, iii)
'      NEXT jjj
'    NEXT iii

' Oversampled voltage measurements have been sent to DOS buffer arrays,
'   alternating between carrier and quadrature values; measurements are
'   now stripped from the buffers and placed in the 'carrsum&' and
'   'quadsum&' arrays.  Each channel/gain list scan of 2 * slength%
'   data points is associated with a distinct electrode combination.

   icurrent = 1
   iground = 2
   ivolt2 = 1

   FOR iii = ihdlcnt TO 1 STEP -1
     FOR jjj = 0 TO isamp(iii) - 1 STEP (slength% * 2)

      ocave& = 0: oqave& = 0

      FOR kkk = 0 TO slength% - 1
        ocave& = ocave& + ibuff(((2 * kkk) + jjj), iii)
        oqave& = oqave& + ibuff(((2 * kkk + 1) + jjj), iii)
      NEXT kkk

      carrsum&(icurrent, iground, ivolt2) = ocave& + carrsum&(icurrent, iground,
      ivolt2)
      quadsum&(icurrent, iground, ivolt2) = oqave& + quadsum&(icurrent, iground,
      ivolt2)

' Calculate average voltages for each projection set for each case where
'   the current injection and ground are 180 degrees opposed.  There are
'   (elec%/2) cases per projection set, and the voltages are recorded and
'   averaged for each electrode.

      IF iground = (icurrent + (elec% / 2)) THEN
        k = ivolt2 - icurrent + 1
        IF k < 1 THEN k = elec% + 1 - icurrent + ivolt2
        Vsum!(projloop%, k) = Vsum!(projloop%, k) + SQR(CSNG(ocave& ^ 2 + oqave& ^
      2))
      END IF

' Advance electrode indices for next scan's worth of data

      ivolt2 = ivolt2 + 1
      IF ivolt2 > elec% THEN
        ivolt2 = 1
        iground = iground + 1
        IF iground > elec% THEN
          icurrent = icurrent + 1
          iground = icurrent + 1
        END IF
      END IF

     NEXT jjj
   NEXT iii

' Copy data from similar case to "fudge" data for last electrode combination
'   (recall that in fast mode, the electronics shut down before the trigger
'   pulse for the last electrode combination makes it to the DT card)
```

```
    carrsum&(elec% - 1, elec%, elec%) = carrsum&(elec% - 2, elec% - 1, elec% - 1)
    quadsum&(elec% - 1, elec%, elec%) = quadsum&(elec% - 2, elec% - 1, elec% - 1)

    FOR k = 1 TO elec%
      Vsum!(projloop%, k) = Vsum!(projloop%, k) / CSNG((elec% / 2) * (slength%))
    NEXT k

' Record time at which projection work ends

    endtime = TIMER

    PRINT USING " Projection ### of ### acquired in ##.## seconds"; projloop%; proj%;
       endtime - starttime!
' proj% on previous line added by TJO 11/6/96
    PRINT USING " Mean Cross Electric Voltage, 1 to #, projection ### = #####.## ";
       (elec% / 2 + 1); projloop%; Vsum!(projloop%, 1)
    PRINT

  NEXT projloop%

'  CLOSE #2 '*** diagnostic dump file

'*********************************************************************
' Clean up extended memory and buffers, terminate communication with
'   DT board, and erase arrays no longer needed

  CALL cleanup(dummy)

  istat = dt.terminate(ihandle)
  IF istat <> 0 THEN CALL exit.error(istat, "dt.terminate")

  ERASE ixmhdl, ihdl, xmaddr, isamp, ibuff
  ERASE ichan, igain

  INPUT " Hit <return> to continue. ", resp$

'*********************************************************************
' Call subroutines to compute voltage statistics and output results

  CALL eistats(elec%, proj%, slength%, carrsum&(), quadsum&(), Vsum!(), resp$)

  ERASE carrsum&, quadsum&, Vsum!

LOOP WHILE resp$ = "C" OR resp$ = "c"

PRINT : PRINT " Program stop.": PRINT

CLOSE
END

REM $STATIC
SUB cleanup (dummy)
' This subroutine unlocks and frees extended memory, deletes buffer
'   transfer lists, and stops the DT board's counter.  It can be called
'   after normal or abnormal termination of data acquisition.

SHARED iunit, isection, ihandle, ihdlcnt
SHARED ixmhdl(), ihdl()

  FOR iii = 1 TO ihdlcnt
'      PRINT "Buffer = "; iii'*** diagnostic
     istat = dt.xm.unlock.block(ixmhdl(iii))
     IF istat <> 0 THEN PRINT "dt.xm.unlock.block, istat = ", istat
```

```
        istat = dt.xm.free.block(ixmhdl(iii))
        IF istat <> 0 THEN PRINT "dt.xm.free.block, istat = ", istat
        istat = dt.delete.buffer(iunit, isection, ihdl(iii))
        IF istat <> 0 THEN PRINT "dt.delete.buffer, istat = ", istat
    NEXT iii

    istat = dt.ct31.terminate(ihandle)
    IF istat <> 0 THEN PRINT "dt.ct31.terminate, istat = ", istat

'   PRINT "Cleanup complete." '*** diagnostic

'   CLOSE #2 '*** diagnostic dump file

END SUB

SUB exit.error (status AS INTEGER, message AS STRING)
' This version of 'exit.error' copied from module 'bas_sub.bas' to
'     avoid loading the entire module and to speed up execution of
'     FASTMEM.BAS ---dlg, 9/11/97

    SHARED ihandle
    IF status = 0 THEN EXIT SUB

    PRINT : PRINT "Error "; status; " returned by routine "; message

'   Call subroutine to close buffers and clean up communications
    CALL cleanup(dummy)

    PRINT "Program terminating status="; dt.terminate(ihandle)
    STOP
END SUB
```