```
EEEEEEEEEE  XX      XX    AAAAAA    MM      MM  PPPPPPPP    LL          EEEEEEEEEE    SSSSSSSS
EEEEEEEEEE  XX      XX    AAAAAA    MM      MM  PPPPPPPP    LL          EEEEEEEEEE    SSSSSSSS
EEEEEEEEEE  XX      XX    AAAAAA    MM      MM  PPPPPPPP    LL          EEEEEEEEEE    SSSSSSSS
EE          XX      XX   AA    AA   MMMM  MMMM  PP      PP  LL          EE          SS
EE          XX      XX   AA    AA   MMMM  MMMM  PP      PP  LL          EE          SS
EE          XX      XX   AA    AA   MMMMMMMMMM  PP      PP  LL          EE          SS
EE            XX  XX     AA    AA   MM  MM  MM  PP      PP  LL          EE          SS
EE            XX  XX     AA    AA   MM  MM  MM  PP      PP  LL          EE          SS
EE            XX  XX     AA    AA   MM  MM  MM  PP      PP  LL          EE          SS
EEEEEEEE        XX       AA    AA   MM      MM  PPPPPPPP    LL          EEEEEEEE      SSSSSS
EEEEEEEE        XX       AA    AA   MM      MM  PPPPPPPP    LL          EEEEEEEE      SSSSSS
EEEEEEEE        XX       AA    AA   MM      MM  PPPPPPPP    LL          EEEEEEEE      SSSSSS
EE            XX  XX     AAAAAAAAAA   MM      MM  PP          LL          EE                  SS
EE            XX  XX     AAAAAAAAAA   MM      MM  PP          LL          EE                  SS
EE            XX  XX     AAAAAAAAAA   MM      MM  PP          LL          EE                  SS
EE          XX      XX   AA    AA   MM      MM  PP          LL          EE                  SS
EE          XX      XX   AA    AA   MM      MM  PP          LL          EE                  SS
EE          XX      XX   AA    AA   MM      MM  PP          LL          EE                  SS
EEEEEEEEEE  XX      XX   AA    AA   MM      MM  PP          LLLLLLLLLL  EEEEEEEEEE  SSSSSSSS
EEEEEEEEEE  XX      XX   AA    AA   MM      MM  PP          LLLLLLLLLL  EEEEEEEEEE  SSSSSSSS
EEEEEEEEEE  XX      XX   AA    AA   MM      MM  PP          LLLLLLLLLL  EEEEEEEEEE  SSSSSSSS
```

```
LL              AAAAAA    BBBBBBBB      IIIIII      000000      CCCCCCCC    000000    NN      NN
LL              AAAAAA    BBBBBBBB      IIIIII      000000      CCCCCCCC    000000    NN      NN
LL            AA      AA  BB      BB      II      00      00  CC          00      00  NN      NN
LL            AA      AA  BB      BB      II      00      00  CC          00      00  NN      NN
LL            AA      AA  BB      BB      II      00      00  CC          00      00  NNNN    NN
LL            AA      AA  BB      BB      II      00      00  CC          00      00  NNNN    NN
LL            AA      AA  BBBBBBBB        II      00      00  CC          00      00  NN  NN  NN
LL            AA      AA  BBBBBBBB        II      00      00  CC          00      00  NN  NN  NN
LL            AAAAAAAAAA  BB      BB      II      00      00  CC          00      00  NN    NNNN
LL            AAAAAAAAAA  BB      BB      II      00      00  CC          00      00  NN    NNNN
LL            AA      AA  BB      BB      II      00      00  CC          00      00  NN      NN      ....
LL            AA      AA  BB      BB      II      00      00  CC          00      00  NN      NN      ....
LLLLLLLLLL    AA      AA  BBBBBBBB      IIIIII      000000      CCCCCCCC    000000    NN      NN      ....
LLLLLLLLLL    AA      AA  BBBBBBBB      IIIIII      000000      CCCCCCCC    000000    NN      NN      ....

FFFFFFFFFF      000000    RRRRRRRR
FFFFFFFFFF      000000    RRRRRRRR
FF            00      00  RR      RR
FF            00      00  RR      RR
FF            00      00  RR      RR
FF            00      00  RR      RR
FFFFFFFF      00      00  RRRRRRRR
FFFFFFFF      00      00  RRRRRRRR
FF            00      00  RR  RR
FF            00      00  RR  RR
FF            00      00  RR    RR
FF            00      00  RR    RR
FF              000000    RR      RR
FF              000000    RR      RR
```

```
LAB

! T
! T
! t
! t
! d
! t
! t
! T

! G
! I

! D
!

! V

! R
! a

! 1

! 1
```

```
File:  LABIOCON.FOR
       Version 'V04-000'

*****************************************************************************
*                                                                           *
*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
*  ALL RIGHTS RESERVED.                                                     *
*                                                                           *
*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
*  TRANSFERRED.                                                             *
*                                                                           *
*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
*  CORPORATION.                                                             *
*                                                                           *
*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
*                                                                           *
*                                                                           *
*****************************************************************************


        Program LABIO_CONNECT

! Define Labio data structures
        Include   'LABCHNDEF.FOR'

! Mailbox Definitions

        Include 'LABMBXDEF.FOR'          !Defines Mailbox Data Structures

! System Service Definitions

        Logical*4 SYS$CREMBX,SYS$ASSIGN
        Logical*4 SUCCESS
        External SS$_ENDOFFILE

! Subroutine Definitions

        Integer CONNECT,DISCONNECT,ABORT,ALLOCATE,DEALLOCATE
        Integer READ_MAILBOX,WRITE_MAILBOX,LABIO_LOG,ACKNOWLEDGE
        Integer CHECK_PID,RETURN_CODE


! Command Data Structures

        Parameter     MAX_COMMAND = 5
        Character*15 COMMAND,COMMAND_TABLE(MAX_COMMAND)
        Data COMMAND_TABLE     /'CONNECT',
        1                       'DISCONNECT',
        1                       'ABORT',
```

```
     1                    'ALLOCATE',
     1                    'DEALLOCATE'/


!
! Map to the Global Data Section 'LABIO_COMMON'
! And Define the Commom Event Flag CLusters
! Request write access to the data base.
!
      Call LABIO_INIT ( 1 )

!
! See if mailbox LABIO_CONNECT exists by attempting to assign it, if
! it does not exist, create it. This mailbox is used to communicate with
! other LABIO processes. Restrict  it to processes within this group.
!
      SUCCESS = SYS$ASSIGN('LABIO_CONNECT',MBX_CHANNEL,,)
      If (.not.  SUCCESS ) Then
        SUCCESS = SYS$CREMBX(,MBX_CHANNEL,,,%Val('FD00'x),,'LABIO_CONNECT')
        If (.not. SUCCESS)
     1   Call FATAL_ERROR( SUCCESS, 'Creating mailbox')
      End If

!
! Tell other processes that we're ready to go.
!
      Call SYS$SETEF( %val( EF_CONNECT ) )
```

```
! Get a command from a requesting processes
!
10      Call READ_MAILBOX       !Get a message
        Call CONNECT_CHECK      !Check the database to clear
                                !any deleted processes.
!
! If I/O status is EOF then process has terminated, ABORT it.
!
        If ( MBX_IO_STATUS .eq. %Loc(SS$_ENDOFFILE) ) Go To 23
!
! Decode characters as a command
!
        If ( MBX_MESSAGE_L .eq. 0 ) Go To 10
        Decode (MBX_MESSAGE_L,100,MBX_MESSAGE,ERR=10) COMMAND
!
! Search Command Table for Command
!
        Do 11 COMMAND_INDEX = 1,MAX_COMMAND
        If( COMMAND .eq. COMMAND_TABLE(COMMAND_INDEX) ) Go To 12
11      Continue
        Go To 13        !Illegal command
!
! Dispatch to correct routine
!
12      Go To (21,22,23,24,25) COMMAND_INDEX
!
! If we get here, it's an unknown command
!
13      Call LABIO_LOG(-1)
!
! CONNECT command
!
21      RETURN_CODE =  CONNECT (MBX_PID)
        Call ACKNOWLEDGE( RETURN_CODE )         !Acknowledge the request
        Call LABIO_LOG ( RETURN_CODE )          !Log the acknowledgement
!
! Disconnect if was bad connect
!
        If (RETURN_CODE .ne. 0 ) Call DISCONNECT(-1)
        Go To 10
!
! DISCONNECT Command
!
22      RETURN_CODE =  DISCONNECT  (MBX_PID)
        Call LABIO_LOG ( RETURN_CODE )          !Log the acknowledgement
        Go To 10
```

```
!
! ABORT command
!
23      RETURN_CODE =  ABORT  (MBX_PID)
        Go To 40
!
! ALLOCATE command
!
24      RETURN_CODE =  ALLOCATE (MBX_PID)
        Go To 40
!
! DEALLOCATE command
!
25      RETURN_CODE = DEALLOCATE (MBX_PID)
        Go To 40
!
! Return status in first character position
!
40      Call ACKNOWLEDGE( RETURN_CODE )       !Acknowledge the request
        Call LABIO_LOG( RETURN_CODE )         !Log the acknowledgement
        Go To 10
!
! Formats
!
100     Format (A)
        End
```

```
        Subroutine CONNECT_CHECK

! This routine checks to make sure all processes
! connected (in CONNECT_BLOCK) actually exist.
! If a process has been deleted, this routine
! removes it from the database by calling ABORT

        Include 'LABCHNDEF.FOR'

        Logical*4 SYS$GETJPI

        Do 10  I = 1, MAX_PID
        PID = CONNECT_BLOCK(I,1)
        If ( PID .ne. 0 ) Then
          If( .not. SYS$GETJPI(%Val(2),PID,,C,,,) ) Call ABORT( PID )
        End If
10      Continue

        Return

        End
```

```
        Logical*4 Function READ_MAILBOX

! This routine reads the LABOI_CONNECT mailbox
! Returns when a message is ready
!

        External IO$_READVBLK
        Include 'LABMBXDEF.FOR'
        Logical*4 SYS$QIOW,SUCCESS

!
! Read for a message from another process
!
        MBX_READ=%LOC(IO$_READVBLK)
        MBX_MESSAGE(1) = ' '

        READ_MAILBOX  = SYS$QIOW(,%Val(MBX_CHANNEL),%Val(MBX_READ),
        1          MBX_IO_STATUS,,,MBX_MESSAGE,
        1          %Val(MBX_MESSAGE),,,,)
        Return

        End
```

```
        Logical*4 Function WRITE_MAILBOX(MBX_CHAN,MESSAGE,MESSAGE_LENGTH)

! This routine writes a message to a mailbox
! Input are the MBX channel, the message, and message length
!

        External IO$_WRITEVBLK,IO$M_NOW
        Logical SYS$QIO
!
! Write response buffer of MBX
!

        MBX_WRITE =%Loc(IO$_WRITEVBLK)+%Loc(IO$M_NOW)

        WRITE_MAILBOX = SYS$QIO(,%Val(MBX_CHAN),%Val(MBX_WRITE),,,,
        1               MESSAGE,%Val(MESSAGE_LENGTH),,,,)

99      Return

        End
```

```
        Logical*4 Function OPEN_MAILBOX(MAILBOX_CHAN,MAILBOX_NAME)

! This routine opens mailbox indicated by MAILBOX_NAME. It returns
! the VMS channel number assignde to it. The mailbox name can be
! padded on the right with blanks.

        Character*(*) MAILBOX_NAME
        Integer   MAILBOX_CHAN
        Logical*4 SYS$ASSIGN,SUCCESS

!
!       Determine length of mailbox name string
!

        MAILBOX_NAME_L=Index(MAILBOX_NAME,' ')-1
        If (MAILBOX_NAME_L .lt. 0 ) MAILBOX_NAME_L=Len(MAILBOX_NAME)

!
!       Assign a channel to mailbox
!       Return status to caller
!
        OPEN_MAILBOX =SYS$ASSIGN(MAILBOX_NAME(:MAILBOX_NAME_L),MAILBOX_CHAN,,)

        Return

        End
```

```
        Subroutine ACKNOWLEDGE (ACK_CODE)
!
! This routine acknowlegdes a request of process, by return the
! command string the process sent us. The string is preceded
! an acknowledge code (ACK_CODE).  The acknowledgement is sent
! via the mailbox the the sending processes had created.
! If that process has not connected to us, we do nothing.

        Include 'LABCHNDEF.FOR'

        Logical*4 WRITE_MAILBOX

        Include 'LABMBXDEF.FOR'
        Integer CONNECT_INDEX,CHECK_PID,ACK_CODE
!
! If process is not in CONNECT_BLOCK, do not respond.
!
        CONNECT_INDEX = CHECK_PID(MBX_PID)

        If (CONNECT_INDEX .ne. 0 ) Then
          Encode( MBX_RESPONSE_L,100,MBX_RESPONSE) ACK_CODE
          MAILBOX = CONNECT_BLOCK(CONNECT_INDEX,2)
          Call WRITE_MAILBOX( MAILBOX, MBX_RESPONSE,
     1                        MBX_MESSAGE_L + MBX_RESPONSE_L )
        End If

        Return

100     Format ( I2 )

        End
```

```
        Subroutine LABIO_LOG( CODE )
!
! This routine logs a message that has been processed. The message
! is written to the log file, along with the time, process ID, IO status
! word and the message length. This routine opens the log file
! if it hasn't been opened.

        Include 'LABMBXDEF.FOR'

        Character*24 TIME
        Logical LOG_OPEN
        Integer CODE

        Data LOG_OPEN/.false./

        Call SYS$ASCTIM(,TIME,,)        !Get the date and time

!
! Open Log file if this is the first time thru
!
        If ( .not. LOG_OPEN ) Then
          Open (Unit = 1, Name='LABIO_LOG', Type='Unknown', Access = 'Append')
          LOG_OPEN = .True.
          Write(1,100) TIME,' Labio Log Opened'
        End If

10      Write(1,200) TIME,MBX_PID,MBX_IO_STATUS,MBX_MESSAGE_L,
        1            CODE,(MBX_MESSAGE(I),I=1,MBX_MESSAGE_L)

        Return

100     Format( 2A )
200     Format( A,Z10,Z10,I10/I3,128A1 )

        End
```

```
        Integer Function CONNECT(REQ_PID)

        Include 'LABCHNDEF.FOR'

        Include 'LABMBXDEF.FOR'
        Character*63 MAILBOX_NAME

        Integer*4 REQ_PID,CHECK_PID
        Logical*4 OPEN_MAILBOX

        CONNECT = 1
!
! Find an empty CONNECT_BLOCK slot
!

        Do 10 I = 1, MAX_PID
        If ( CONNECT_BLOCK(I,1) .eq. 0 ) Go To 20
10      Continue

! We should never get here, since the last slot of
! the CONNECT_BLOCK is a spare for sending message
! disallowing a connect!

        Go To 99
!
! Open user specified MAILBOX
!

20      Decode (MBX_MESSAGE_L,100,MBX_MESSAGE) MAILBOX_NAME
        If( .not. OPEN_MAILBOX( MAILBOX_CHAN,MAILBOX_NAME) ) Go To 99

!
! Allocate the connect block. if it is not a duplicate
! PID, store the PID and mailbox channel in CONNECT_BLOCK
! If it is a duplicate, store the PID as -1.

        If( CHECK_PID(REQ_PID) .eq. 0 ) Then
          CONNECT_BLOCK(I,1) = REQ_PID
          CONNECT = 0
          Else
          CONNECT_BLOCK(I,1) = -1       !Duplicate PID! We will Disconnect
                                        !After Acknowledging request
        End If

        CONNECT_BLOCK(I,2) = MAILBOX_CHAN

        If ( I .ge. MAX_PID ) CONNECT = 1 !No room for process!

99      Return

100     Format(15X,A)

        End
```

```
        Integer Function DISCONNECT(REQ_PID)

! This routine disconnects a process from the LABIO system.
! If it is a valid process, all channels still allocated are
! deallocated, the request is acknowledged, the channel assigned
! to the mailbox is deassigned, and the CONNECT_BLOCK entry is removed.

        Include 'LABCHNDEF.FOR'
        Integer*4 REQ_PID,CHECK_PID

        DISCONNECT = 1

! Find index into connect block

        CONNECT_INDEX = CHECK_PID(REQ_PID)
        If (CONNECT_INDEX .eq. 0 ) Go To 99 !Not connected

! Deallocate all A/D channels

        Call DEALLOCATE_ALL(REQ_PID)

! Acknowledge DISCONNECT request

        Call ACKNOWLEDGE(0)

! Close the mailbox, and zero CONNECT_BLOCK

        Call SYS$DASSGN( %Val(CONNECT_BLOCK(CONNECT_INDEX,2)) )
        CONNECT_BLOCK(CONNECT_INDEX,1) = 0
        CONNECT_BLOCK(CONNECT_INDEX,2) = 0
        DISCONNECT =0

99      Return
        End


        Integer Function ABORT(REQ_PID)
        Call DISCONNECT( REQ_PID )
        Return
        End
```

```
        Integer Function ALLOCATE(REQ_PID)

! This routines allocates an A/D channel to a specific process.
! The process request a channels by number (1-16), specifing
! the asample rate in tics/sample, the buffer size in words, and
! the number of buffers to acquire ( 0 = infinity ). The user can
! default the rate to 1 tic/sample. Default the buffer size to
! the maximum, and the buffer count to 0. If the user reallocates
! the channel, the defaults are the previous values allocated.
! The channel must been INACTIVE if it is reallocated.


        Include 'LABCHNDEF.FOR'
        Include 'LABMBXDEF.FOR'

        Integer*4 REQ_PID       !PID of requesting process
        Integer*4 PARM(4)       !4 input parameters
        Integer*2 CONNECT_INDEX,CHECK_PID
        Integer*4 REQ_AD_CHAN,REQ_TICS,REQ_BUF_SIZE,REQ_BUF_COUNT
        Logical   CHECK_PARM
!
! Get index into CONNECT_BLOCK for REQ_PID
! If index is not > 0 , ignore request
!
        ALLOCATE = 1            !Checking first field

        CONNECT_INDEX = CHECK_PID(REQ_PID)
        If ( CONNECT_INDEX .le. 0 ) Go To 99 !Req. Proc not connected!
!
! Decode message into four fields
!
        Decode ( MBX_MESSAGE_L,100,MBX_MESSAGE) PARM

        REQ_AD_CHAN = PARM(1)   !Requested A/D channel is first parm
        REQ_TICS    = PARM(2)   !Tics/sample is 2nd
        REQ_BUF_SIZE= PARM(3)   !Buffer size is 3rd
        REQ_BUF_COUNT=PARM(4)   !Number of buffers is 4th

        ALLOCATE = 2            !Check next parameter (channel number)

! Valid channel numbers are 1-16

        If (REQ_AD_CHAN .lt. 1 .or. REQ_AD_CHAN .gt. 16) Go To 99

! Requested channel must not allocated, or
! allocated to the requesting process

        If ( AD_BLOCK(2,REQ_AD_CHAN) .ne. 0 .and.
     1    AD_BLOCK(2,REQ_AD_CHAN) .ne. REQ_PID ) Go To 99

! The channel must not be active
        If (AD_BLOCK(1,REQ_AD_CHAN) .gt. INACTIVE ) Go To 99

        ALLOCATE = 3            !Checking next parm (Tics/sample)

! Tics/sample must be between 1 and 2^31-1
```

```
        If( .not. CHECK_PARM(REQ_TICS,AD_BLOCK(3,REQ_AD_CHAN),
        1                    1,'7FFFFFFF'X,1) ) Go To 99

        ALLOCATE = 4    !Checking  parmeter (Buffer size)

!
! Buffer size between 1 and MAX_BUF_SIZE
!
        If( .not. CHECK_PARM(REQ_BUF_SIZE,AD_BLOCK(4,REQ_AD_CHAN),
        1                    1,MAX_BUF_SIZE,MAX_BUF_SIZE) ) Go To 99

        ALLOCATE = 5    ! Checking next parameter (number of buffers)

! Number of buffers to acquire must be between 1 and 2^31-1, or
! zero to indicate no limit

        If ( .not. CHECK_PARM(REQ_BUF_COUNT,AD_BLOCK(5,REQ_AD_CHAN),1,
        1                     '7FFFFFFF'x,0) ) Go To 99

        ALLOCATE = 0    !Everything is acceptable
!
! Enter info into AD_BLOCK
!
        AD_BLOCK(1,REQ_AD_CHAN) = 0             !Lock the data base
!
! Clear associated event flags
!
        Call SYS$CLREF(%Val( EF_NOTIFY_OFF + REQ_AD_CHAN ) )
        Call SYS$CLREF(%Val( EF_ACTIVITY_OFF + REQ_AD_CHAN) )
        Call SYS$CLREF(%Val( EF_STATUS_OFF + REQ_AD_CHAN ) )


        AD_BLOCK(2,REQ_AD_CHAN) = REQ_PID       !Requesting PID
        AD_BLOCK(3,REQ_AD_CHAN) = REQ_TICS      !Tics/sample
        AD_BLOCK(4,REQ_AD_CHAN) = REQ_BUF_SIZE  !Requested buffer size
        AD_BLOCK(5,REQ_AD_CHAN) = REQ_BUF_COUNT !Number of buffers to acquire
        AD_BLOCK(6,REQ_AD_CHAN) = 0             !No buffers acquired
        AD_BLOCK(7,REQ_AD_CHAN) = 0             !No data buffer available
        AD_BLOCK(8,REQ_AD_CHAN) = 0             !Number elements in last buf
        AD_BLOCK(9,REQ_AD_CHAN) = 1             !Current buffer index
        AD_BLOCK(10,REQ_AD_CHAN) = 0            !Current buffer count
        AD_BLOCK(11,REQ_AD_CHAN) = 1            !Tics remaining
        AD_BLOCK(12,REQ_AD_CHAN) = 0            !Offset to next data point
        AD_BLOCK(1,REQ_AD_CHAN) = INACTIVE      !Channel is inactive
        Return
!
!       Error return
!
99      Return          !Return to caller

100     Format(15X,41)

        End
```

```
        Integer Function DEALLOCATE(REQ_PID)

! This routine deallocates a channel previously allocated by
! a process. The channel must be INACTIVE when deallocated.

        Include 'LABCHNDEF.FOR'
        Include 'LABMBXDEF.FOR'

        Integer*4 REQ_PID       !PID of requesting process
        Integer*2 CONNECT_INDEX,CHECK_PID
        Integer*4 REQ_AD_CHAN

! Get index into CONNECT_BLOCK for REQ_PID
! If index is not > 0 , ignore request

        DEALLOCATE = 1          !Checking first field

        CONNECT_INDEX = CHECK_PID(PID)
        If ( CONNECT_INDEX .le. 0 ) Go To 99

        DEALLOCATE = 2

        Decode (MBX_MESSAGE_L,100,MBX_MESSAGE) REQ_AD_CHAN

! Valid channel numbers are 1-16

        If (REQ_AD_CHAN .lt. 1 .or. REQ_AD_CHAN .gt. 16) Go To 99

! Does requesting process own the channel?
        DEALLOCATE = 21

        If (AD_BLOCK(2,REQ_AD_CHAN) .ne. REQ_PID ) Go To 99

! Is the channel inactive, clear the channel parameters
        DEALLOCATE = 22

        If ( AD_BLOCK(1,REQ_AD_CHAN) .ne. INACTIVE ) Go to 99

        Call AD_CANCEL(REQ_AD_CHAN)

        DEALLOCATE = 0          !Everything OK

        Return

!
! ERROR return
!

99      Return

!
! This entry point is used to deallocate all channels
! allocated to a specific process.

        Entry DEALLOCATE_ALL(REQ_PID)
```

```
        DEALLOCATE = 1

! Valid PID?

        CONNECT_INDEX = CHECK_PID(PID)
        If ( CONNECT_INDEX .ne. 0 ) Then
! Look for all A/D channels allocated to process
! and cancel all I/O unconditionally.
          Do 10 AD_CHAN = 1 , MAX_AD_CHANNEL
          If ( AD_BLOCK(2,AD_CHAN) .eq. REQ_PID ) Call AD_CANCEL(AD_CHAN)
10        Continue
          DEALLOCATE_ALL = 0
        End If

        Return
100     Format(15X,I15)
        End
```

```
        Integer*4 Function AD_CANCEL( CHANNEL )

! Clears the parameter table associated with A/D channel

        Include 'LABCHNDEF.FOR'
        Integer CHANNEL

        AD_CANCEL = 1           !Assume error
!
! Legal channel numbers are 1-16
!
        If ( CHANNEL .ge. 1 .and. CHANNEL .le. 16 ) Then
!
! Zero the AD_BLOCK for this channel
!
          Do 10  J = 1 , 16             !Clear everthing
10        AD_BLOCK(J, CHANNEL ) = 0
          AD_CANCEL = 0                 !Everything ok
        End If
!
! Clear associated event flags
!
        Call SYS$CLREF(%Val( EF_NOTIFY_OFF + CHANNEL ) )
        Call SYS$CLREF(%Val( EF_ACTIVITY_OFF + CHANNEL ) )
        Call SYS$CLREF(%Val( EF_STATUS_OFF + CHANNEL ) )

99      Return

        End
```

```
        Logical Function CHECK_PARM(IVAL,OVAL,MIN,MAX,DEFAULT)

! This routine validates and defaults an input parameter (IVAL)
! If IVAL is not 0, it compares it to MIN and MAX, returning TRUE or FALSE.
! If IVAL is 0, and OVAL is not zero, IVAL = OVAL
! If IVAL is 0, and OVAL is zero, IVAL = DEFAULT

        Integer*4 IVAL,OVAL,MIN,MAX,DEFAULT

        CHECK_PARM = .false.    !assume the worst

        If (IVAL .ne. 0 ) Then
          If( IVAL .lt. MIN .or. IVAL .gt. MAX) Go To 99
        Else
          If (OVAL .ne. 0 ) Then
            IVAL = OVAL
          Else
            IVAL = DEFAULT
          End If
        End If

        CHECK_PARM = .true.

99      Return

        END
```

```
        Integer Function CHECK_PID(PID)

! This routine checks to see if a PID is in CONNECT_BLOCK
! If it is, the INDEX into CONNECT_BLOCK is returned. If
! it isn't, 0 is returned

        Include 'LABCHNDEF.FOR'
        Integer*4 PID

! Assume PID is not in database
        CHECK_PID = 0

! If PID is found, return index.

        Do 10 I = 1 , MAX_PID
        If( CONNECT_BLOCK(I,1) .eq. PID ) CHECK_PID = I
10      Continue

        Return
        End
```

XALINK
MAR

DRMASTER
FOR

LABIOLINK
COM

LPATEST
FOR

LABIOPEAK
FOR

LABIOSTRT
COM

XAMESSAGE
MAR

XATEST
FOR

LABIOCOM
FOR

LBRDEMO
COM

LABMBXDEF
FOR

LABIOSAMP
FOR

MAILCOMPRESS
COM

LABCHNDEF
FOR

CONNECT
COM

LABIOCON
FOR

LBRDEMO
FOR

PEAK
FOR

DRCOPYBLD
COM

XIDRIVER
MAR

LABIOSEC
FOR

DRSLAVE
FOR

LABIOACQ
FOR

LABIOCOMP
COM

LABIOSTAT
FOR

TESTLABIO
FOR