CMI
125 West Park Loop
Huntsville, AL 36806
Phone 256.722.0175
Fax 256.722.0144

Chandler/May, Inc.

VxWorks Device Driver User's Manual

VxWorks Device Driver Software for the General Standards PMC-16LCAIO hosted on PowerPC and 80x86 Processors

Document number:	9005008	Revision:	1.0	Date: 04/13/00
Engineering Approval:				Date:
Quality Representative Approval:				Date:

Acknowledgments

1 Scope

The purpose of this document is to describe how to interface with the PMC-16LCAIO VxWorks Device Driver developed by Chandler/May, Incorporated (CMI). This software provides the interface between "Application Software" and the 16LCAIO Board. The interface to this board is at the I/O system level. It requires no knowledge of the actual board addressing of control/data register locations. It does, however, require some knowledge of the individual bit representations for most control/data registers on the device.

The 16LCAIO Driver Software executes under control of the VxWorks operating system. The 16LCAIO is implemented as a standard VxWorks device driver written in the 'C' programming language. The 16LCAIO Driver Software is designed to operate on CPU boards containing MPC603, MPC604, and MPC750 processors as well as VME CPU boards containing 80x86 processors with the same bus interface hardware as PowerPC boards. For example, the Force PPC/PowerCore-6604 CPU board, the Motorola MVME1600, MVME2300, MVME2400, MVME2600, and MVME2700 series boards, the Omnibyte Galaxy+, and the SCI JTT 686 CPU board.

2 Hardware Overview

The General Standards Corporation (GSC) 16LCAIO board is a single-width analog I/O interface that fits into a PCI Mezzanine Card slot. This board provides 32 16-bit analog input channels, 4 16-bit analog output channels, and a 16-bit bi-directional digital port with two auxiliary control lines. The output channels are capable of supporting synchronous and asynchronous modes. The inputs can be customized as single-ended or differential input channels via software configuration. It also provides for minimum off-line maintenance by providing calibration and self-testing functions.

The 16LCAIO board includes two rate generators and a DMA controller. The rate controllers are provided to control the rate at which input/output channels are sampled. The DMA transfers are supported when the board is acting as the bus master and the local bursting mode disabled.

The configuration of the interrupting capability of the 16LCAIO board is described in the hardware manual for the board. The 16LCAIO Device Driver must be used correctly in accordance with the hardware configuration in order to provide consistent results.

3 Referenced Documents

The following documents provided reference material used in the development of this design:

- PMC-16LCAIO User's Manual, 16-Bit Analog Input/Output PMC Board – Revision 022800, General Standards Corporation.
- PLX Technology, Inc. PCI 9080 PCI Bus Master Interface Chip data sheet.
- Motorola MVME1603/1604 Single Board Computer Programmer's Reference Guide.
- Motorola MVME2300-Series VME Processor Module Programmer's Reference Guide.
- Motorola MVME2600/2700 Single Board Computer Programmer's Reference Guide.
- Force PPC/PowerCore-6603/4 Technical Reference Manual.

4 Making the Device Driver

In order to use the 16LCAIO Device Driver for a particular target CPU platform, the driver object files must be built by "making" or compiling the software modules. The object modules are those that are loaded by the VxWorks target processor and contain functions, which can then be executed. The Wind River Tornado environment makes this process easy with one simple command: **make**. **make** uses a file, called a makefile, which tells the development system which source modules are to be compiled, the parameters and options to use when compiling, and any other miscellaneous file operations a user may need to build a particular system of object modules. The makefile included contains several Board Support Package dependent switches that must be defined correctly for successful compilation and use. The user is only required to set the **BSP** variable in the makefile. Once **BSP** is set correctly, the user can then begin compiling by executing **make**.

The modules in the make file should begin compiling and the display should reflect a successful compilation of all modules.

The output files from the build procedure should be:

16lcaio_drv.o	Relocatable/loadable module for the device driver.
16lcaio_menu.o	Relocatable/loadable module for the sample menu tool.

5 Driver Interface

The 16LCAIO Driver conforms to the device driver standards required by the VxWorks Operating System and contains the following standard driver entry points.

- 16LCAIODrvInstall() - Installs the device driver for use with multiple 16LCAIO Cards
- 16LCAIODrvRemove() - Removes the device driver from use
- open() - opens a driver interface to one 16LCAIO Card
- close() - closes a driver interface to one 16LCAIO Card
- read() - reads data received from a 16LCAIO Card
- write() - writes data to be transmitted by a 16LCAIO Card
- ioctl() - performs various control and setup functions on the 16LCAIO Card

The 16LCAIO Device Driver provides a standard I/O system interface to the GSC PMC-16LCAIO card for VxWorks applications, which run on the VxWorks target processor. The device driver is installed and devices created through the use of standard VxWorks I/O system functions. The functions of the driver can then be used to access the board.

Included in the device driver software package is a menu driven board testing program and source code. This program is delivered undocumented and unsupported but may be used to exercise the 16LCAIO card and device driver. It can also be used to break the learning curve somewhat for programming the 16LCAIO device.

If the user wishes to use the 16LCAIO Device Driver with the interrupting capability of the board then a user supplied Interrupt Service Routine (ISR) must be written. The driver will call this ISR when an interrupt is received from the board. There are limitations on the functionality of a VxWorks ISR. These are documented in the VxWorks Programmer's Guide and must be strictly followed in writing the ISR.

The Device Driver initializes the board to disable all types of 16LCAIO interrupts through software control except for PCI interrupts controlled through the Shared Runtime - Interrupt Control/Status register. 16LCAIO Interrupts must be enabled through the use of the ioctl function in order to take advantage of the interrupting capability of the board. The ioctl function must also be used to specify the user supplied ISR which will be invoked when an interrupt is received from the board. If interrupting is enabled and the user supplied ISR has not been specified then nothing will happen in the driver when an interrupt is received from the board.

The 16LCAIO Device Driver allows for multiple boards on a single PCI bus. Each board will be addressed as a separate VxWorks I/O system device. This device will be created when the driver is installed and is then available for all driver operations (open, close, etc.).

It is important to note that the 16LCAIO device driver is target processor dependent and thus BSP dependent. System calls are made within the driver, which are only available through certain board support packages. This is due to the fact that PCI memory and I/O space could be mapped differently for each target processor board. Also, it may be possible that the PMC slot interrupt level may be mapped differently for each target processor board.

5.1 16LCAIODrvInstall()

The 16LCAIODrvInstall () function installs the device driver into the VxWorks operating system. This function must be called prior to using any of the other driver functions. This function should not be called again without first calling the 16LCAIODrvRemove() function.

The 16LCAIODrvInstall () function performs the following operations:

- Installs the device driver into the VxWorks operating system
- Performs the following for each PMC Slot on the processor board
  - Determines if this slot contains a PCI card by examining the CPU board's registers
  - Determines if the slot contains a 16LCAIO board by examining the PCI Configuration Device Type and Vendor ID Registers
  - Programs the PCI Configuration Base Address and Configuration Address Registers with predefined addresses
  - Enables the 16LCAIO Card to respond over the PCI Bus
  - Connects the driver interrupt handler for the interrupt number
  - Installs a device for the PMC Slot
  - Enables the PCI Interrupt for the PMC Slot

PROTOTYPE:

extern int 16LCAIODrvInstall(BOOL bDebug);

Where:

bDebug - A Boolean that is sent to the driver to enable debugging. If enabled the driver will display error and status messages on the console during driver access. Note: this should not be enabled during time sensitive processes.

Returns OK on success and ERROR on failure

EXAMPLE:

```
STATUS iStatus;

/* Install the 16LCAIO VxWorks Device Driver. */
iStatus = 16LCAIODrvInstall(TRUE);
```

5.2 16LCAIODrvRemove()

The 16LCAIODrvRemove() function is used to remove the 16LCAIO Device Driver from the VxWorks operating system. This function should only be called after a call to the 16LCAIODrvInstall() function. The 16LCAIODrvRemove() function closes all the open devices for each PMC slot and removes the device driver from the operating system.

PROTOTYPE:

extern int 16LCAIODrvRemove(void);

Returns OK on success and ERROR on failure

EXAMPLE:

```
STATUS iStatus;

/* Remove the 16LCAIO Driver */
iStatus = 16LCAIODrvRemove();
```

5.3 open()

The open() function is the standard VxWorks entry point to open a connection to a 16LCAIO Card in one PMC Slot. This function may only be called after a call to the 16LCAIODrvInstall() function is made. Each PMC device can be opened in analog mode, digital mode, or both analog and digital modes. In other words, a user can associate two file descriptors with one device. One file descriptor can represent the analog portion of the 16LCAIO and the other can represent the digital portion of the 16LCAIO device. The user is responsible for keeping in mind which mode is being used, so that the correct commands are being executed.

PROTOTYPE:

extern int open(const char *cName, int iFlags, int iMode)

Where:

cName - name of the device being opened which is one of the following depending on the slot the 16LCAIO Board is in and what mode:

- 16LCAIO_PMC1/a - PMC slot 1, analog mode
- 16LCAIO_PMC1/d - PMC slot 1, digital mode
- 16LCAIO_PMC2/a - PMC slot 2, analog mode
- 16LCAIO_PMC2/d - PMC slot 2, digital mode

iFlags - is not used.

iMode - is not used.

Returns OK on success and ERROR on failure

EXAMPLE:

```
int         FileDesc[2];
LOCAL char  *devName[] = {16LCAIO_PMC1/a, 16LCAIO_PMC2/d};
int         16LCAIOSlot = 1;


/*   open the 16LCAIO device for slot 1   */
FileDesc[16LCAIOSlot] = open(devName[16LCAIOSlot], O_RDWR, 0644);

if (FileDesc[16LCAIOSlot] == ERROR)
{
   logMsg("Cannot Open Device Error %s\n\n",
```

```
                (int) devName[16LCAIOSlot], 0, 0, 0, 0, 0);
}
```

5.4 close()

The close() function is the standard VxWorks entry point to close a connection to a 16LCAIO Card in one PMC Slot. This function should only be called after the open function has been successfully called for a slot where a 16LCAIO Card resides. The close function closes the interface to a 16LCAIO device.

PROTOTYPE:

extern STATUS close(int iFd);

Where:

iFd - File Descriptor returned from a call to the open function.

Returns OK if successful or ERROR if unsuccessful.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;

/*  close the device on slot 2  */
if  (close(FileDesc[16LCAIOSlot]) == ERROR)
{
   logMsg("Close Error for Slot #%d\n\n", 16LCAIOSlot, 0, 0, 0, 0, 0);
}
FileDesc[16LCAIOSlot] = ERROR;
```

5.5 read()

The read() function is the standard VxWorks entry point to receive channel data from a 16LCAIO Card FIFO in one PMC Slot. This function should only be called after the open function has been successfully called for a slot where a 16LCAIO Card resides. The 16LCAIO device can be opened as an analog device, a digital device, or both analog and digital. This is accomplished in the open() call.

As an analog device, the 16LCAIO has two data configurations in which the input channels can be read, single-ended and differential. Depending on the read mode of the driver which can be set using the ioctl() function, the FIFO data will either be transferred to the user buffer using the PLX 9080 DMA capability or will be accessed directly and assigned 32 bits at a time. Regardless of configuration, the read() function will read these channels in sequential order starting with channel 0. In both modes the number of 32 bit words to read per scan is determined by reading the number of active channels. In order to keep data aligned properly in the input FIFO data is read on a per scan basis but is not restricted to single scans within one read() operation. Therefore, if the user does not pass in a buffer large enough to accept a full scan of data, then that read() operation is returned with zero bytes read. The analog input buffer register contains a channel flag, which gets set when channel 0 is encountered. Otherwise, the flag is ignored.

As a digital device, the driver will make a scan of the digital I/O port for each 16 bits requested. If the high and low digital bytes are designated as inputs, bits 0-15 of the user buffer will be filled in with that data. If the bidirectional digital bytes are configured as outputs, then bits 0-15 of the user buffer will be filled with zeros. Regardless, bit 16 will be set equal to the auxiliary input line and bits 17-31 will be invalid data.

PROTOTYPE:

```
extern int read(int iFd, char *cBuffer, size_t iMaxbytes);
```

Where:

iFd - File Descriptor returned from a call to the open function.

CBuffer - pointer to character array to store read bytes.

iMaxbytes - maximum number of bytes to read.

Returns Number of bytes read if successful or ERROR if unsuccessful.

EXAMPLE:

```
#define MAXSAMPLES 32

int   FileDesc[2];
int   iNumBytesRead;
int   16LCAIOSlot = 1;
char  pulBuffer[MAXSAMPLES * 4];


/* Configure driver analog read() mode */
if(ioctl(FileDesc[16LCAIOSlot], READ_MODE_CONFIG, DMA_MODE) == ERROR)
{
    logMsg("ioctl READ_MODE_CONFIG Failed for Slot #%d\n\n",
           16LCAIOSlot, 0, 0, 0, 0, 0);
}

/* Configure Analog Input Channel Mode */
if(ioctl(FileDesc[16LCAIOSlot], INPUT_CONFIG, SINGLE_ENDED) == ERROR)
{
    logMsg("ioctl INPUT_MODE_CONFIG Failed for Slot #%d\n\n",
           16LCAIOSlot, 0, 0, 0, 0, 0);
}

/* Read from the 16LCAIO analog device  */
iNumBytesRead = read(FileDesc[16LCAIOSlot],
                     pulBuffer,
                     sizeof(pulBuffer));

if (iNumBytesRead == 0)
{
   logMsg("Read failed for Slot #%d\n", 16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.6 write()

The write() function is the standard VxWorks entry point to transmit channel data to the 16LCAIO Card FIFO in one PMC Slot. This function should only be called after the open function has been successfully called for a slot where a 16LCAIO Card resides. The 16LCAIO device can be opened as an analog device, a digital device, or both analog and digital. This is accomplished in the open() call.

As an analog device, the data written to the analog output channels should have the data to be written followed by the channel number. In some cases, flags should be set and accompany the data and channel information. Therefore, there is an **OUT_DATA_FORMAT** structure provided specifically for this purpose. It can be found below or in the header file of the driver. The two flags, which will be discussed in later sections, are used in certain output clocking and sync modes. It should be noted that it is the user's responsibility to ensure that these flags are set properly; also, to understand the purpose for these flags.

As a digital device, the upper and lower bytes should be configured for outputs and data written in a standard format.

PROTOTYPE:

extern int write(int iFd, char *cBuffer, size_t iNBytes);

Where:

iFd - File Descriptor returned from a call to the open function.

cBuffer - pointer to WRITE_PARAM structure containing an array of channel data and sample data to write.

iNBytes - total number of bytes to write.

Returns Number of bytes written if successful or ERROR if unsuccessful.

```
typedef struct OutDataFormat
{

    USHORT usData;
    USHORT usChannelTag;
    USHORT usGroupEnd;
    USHORT usBurstEnd;

} OUT_DATA_FOMAT;
```

EXAMPLE:

Analog:

```
int               FileDesc_A;
int               i, iNumBytesWritten;
OUT_DATA_FORAMT   pulBuffer[4];

FileDesc_A = open("16LCAIO_PMC1/a", O_RDWR, 0644);

if ( FileDesc_A == ERROR)
{
   logMsg("Cannot Open Device Error \n\n",
            0, 0, 0, 0, 0, 0);
}

for ( i = 0; i < 4; i++ )
{
   pulBuffer[i].usData = 0xAAAA;
   pulBuffer[i].usChannelTag = (USHORT)i;
}


iNumBytesWritten = write(FileDesc_A,
                         (char*)&pulBuffer,
                         sizeof(pulBuffer));

if (iNumBytesWritten == 0)
{
   logMsg("write failed \n", 0, 0, 0, 0, 0, 0);
}
else
{
   if (iNumBytesWritten != (4*sizeof(OUT_DATA_FORMAT))/2)
   {
      logMsg("Only wrote %d bytes\n",
             iNumBytesWritten, 0, 0, 0, 0, 0);
   }
}
```

Digital:

```
int         FileDesc_D;
int         i, iNumBytesWritten;
ULONG       pulBuffer[4];

FileDesc_D = open("16LCAIO_PMC1/d", O_RDWR, 0644);

if (FileDesc_D == ERROR)
```

```
{
   logMsg("Cannot Open Device Error \n\n",
            0, 0, 0, 0, 0, 0);
}

for ( i = 0; i < 4; i++ )
{
   pulBuffer[i] = 0xAAAA;
}

iNumBytesWritten = write(FileDesc_D,
                         (char*)&pulBuffer,
                         sizeof(pulBuffer));

if (iNumBytesWritten == 0)
{
   logMsg("write failed \n", 0, 0, 0, 0, 0, 0);
}
else
{
   if (iNumBytesWritten != sizeof(WRITE_PARAM)/2)
   {
      logMsg("Only wrote %d bytes\n",
             iNumBytesWritten, 0, 0, 0, 0, 0);
   }
}
```

5.7 ioctl()

The ioctl() function is the standard VxWorks entry point to perform control and setup operations on a 16LCAIO Card in one PMC Slot. This function should only be called after the open function has been successfully called for a slot where a 16LCAIO Card resides. The ioctl() function will perform different functions based upon the function parameter. These functions will be described in the following subparagraphs.

PROTOTYPE:

extern int ioctl(int iFd, int iFunction, int iArg);

Where:

iFd - File Descriptor returned from a call to the open function.

iFunction - The ioctl function to perform which is one of the following:

NO_COMMAND - Empty command, performs nothing.

INIT_BOARD - Initializes the 16LCAIO board.

READ_REGISTER - Reads a specified 16LCAIO register.

WRITE_REGISTER - Writes to a specified 16LCAIO register.

START_DMA - Starts a DMA read from the 16LCAIO board

REG_FOR_INT_NOTIFY - Registers the application code to be notified when an interrupt occurs.

GET_DEVICE_ERROR - Returns the error that occurred during the last access to the 16LCAIO driver.

READ_MODE_CONFIG - Configures the 16LCAIO read() mode (FIFO scan reads or DMA enabled FIFO reads).

WRITE_MODE_CONFIG - Configures the 16LCAIO write() mode (FIFO scan writes or DMA enabled FIFO writes).

INPUT_CONFIG - Configures the 16LCAIO input channels (single-ended or differential mode).

INPUT_TEST – Performs a self-test for 16LCAIO board validation.

VOLTAGE_RANGE – Sets the voltage range for both inputs and outputs.

IO_DATA_FORMAT – Selects analog input/output data format (offset binary or two's compliment).

OUTPUT_CLK_MODE – Sets output clocking mode (simultaneous or sequential mode).

OUTPUT_SYNC_MODE – Sets output sync mode (continuous or burst mode).

FUNCTION_LOOPING – Selects output function looping (circular/closed buffer or non-looping/open buffer).

ONE_BURST_OUTPUTS – Burst outputs.

ONE_SCAN_INPUTS – Scan inputs.

START_AUTOCAL – Initiates an autocalibration operation.

AUTO_PASS – Retrieves the status of autocalibration.

SCAN_SIZE – Sets number of analog input channels to scan.

INPUT_CLK_SRC – Selects the clocking source for analog inputs.

OUTPUT_CLK_SRC – Selects the clocking source for analog outputs.

BURST_SYNC_SRC – Selects the burst sync source for analog outputs.

EXT_SYNC_SIGNAL_SRC – Selects the signal source for the External Sync output line.

RATE_B_CLK_SRC – Selects the clock input source for the Rate-B generator.

INPUT_SCAN_MODE – Selects analog input scanning mode.

TWO_CHANNEL_SCAN – Sets analog input two-channel size.

ENABLE_PCI_INTERRUPTS – Enables PCI interrupts in order for the 16LCAIO to produce a local interrupt request.

DISABLE_PCI_INTERRUPTS – Disables PCI Interrupts.

SET_RATE_A_GEN – Programs Rate-A generator for specified sample rate frequency.

SET_RATE_B_GEN – Programs Rate-B generator for specified sample rate frequency.

ENABLE_RATE_A_GEN – Enables the Rate-A generator.

DISABLE_ RATE_A_GEN – Disables the Rate-A generator.

ENABLE_RATE_B_GEN – Enables the Rate-B generator.

DISABLE_ RATE_B_GEN – Disables the Rate-B generator.

SET_IN_THRESHOLD – Sets the value at which the input buffer is said to be full.

SET_OUT_THRESHOLD – Sets the value at which the output buffer is said to be full.

CLEAR_IN_BUFFER – Clears (empties) the analog input buffer.

CLEAR_OUT_BUFFER – Clears (empties) the analog output buffer.

BUFF_IN_ STATUS_FLAG – Retrieves the analog input buffer status flag information.

BUFF_OUT_STATUS_FLAG – Retrieves the analog output buffer status flag information.

SELECT_DIGITAL_OUT_HI – Establishes the direction of the upper byte in the digital I/O port register to be output.

SELECT_DIGITAL_OUT_LO – Establishes the direction of the lower byte in the digital I/O port register to be output.

SELECT_DIGITAL_IN_HI – Establishes the direction of the upper byte in the digital I/O port register to be input.

SELECT_DIGITAL_IN_LO – Establishes the direction of the lower byte in the digital I/O port register to be input.

SET_DIGITAL_AUX_OUT – Sets the auxiliary output line.

CLEAR_DIGITAL_AUX_OUT – Clears the auxiliary output line.

CHECK_DIGITAL_AUX_IN – Retrieves the auxiliary input line status.

IRQ0_INT_SRC – Sets IRQ0 interrupt source condition.

IRQ1_INT_SRC – Sets IRQ1 interrupt source condition.

IRQ2_INT_SRC – Sets IRQ2 interrupt source condition.

CLEAR_GRP0_INT_REQ – Clears group 0 interrupt request flag.

CLEAR_GRP1_INT_REQ – Clears group 1 interrupt request flag.

CLEAR_GRP2_INT_REQ – Clears group 2 interrupt request flag.

iArg - The parameters to the specific ioctl() function. See the following subsections for a description of the parameters for each function.

Returns OK if successful or ERROR if unsuccessful.

5.7.1 NO_COMMAND

This is an empty driver entry point. This command may be given to validate that the driver is correctly installed and that the 16LCAIO board device has been successfully opened.

arg PARAMETER:

Not used.

EXAMPLE:

```
int FileDesc[2];
int 16IAOSlot = 1;

if (ioctl(FileDesc[16LCAIOSlot], NO_COMMAND, 0) == ERROR)
{
   logMsg("ioctl NO_COMMAND Failed for Slot #%d\n\n", 16LCAIOSlot,
            0, 0, 0, 0, 0);
}
```

5.7.2 INIT_BOARD

The INIT_BOARD function initializes the board and sets all defaults.

arg PARAMETER:

Not used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;

if (ioctl(FileDesc[16LCAIOSlot], INIT_BOARD, 0) == ERROR)
{
   logMsg("Board Initialization Failed for Slot #%d\n\n",
   16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.3 READ_REGISTER

The READ_REGISTER function reads and returns the contents of one of the 16IAO registers.

arg PARAMETER:

REG_PARAM *

where REG_PARAM is defined to be

```
typedef struct RegParam
{
    int             e16LCAIORegister;
    ULONG    *pulValue;
} REG_PARAM;
```

and,

int e16LCAIORegister - One of the following registers to read. Refer to the 16LCAIO hardware documentation for a description of each register.

***** 16LCAIO Registers *****

BOARD_CTRL_REG

INT_CTRL_REG

INPUT_BUFF_REG

INPUT_BUFF_CTRL_REG

RATE_A_GEN_REG

RATE_B_GEN_REG

OUTPUT_BUFF_CTRL_REG

SCAN_SYNC_CTRL_REG

DIGITAL_PORT_REG

***** DMA Registers *****

DMA_CH_0_MODE

DMA_CH_0_PCI_ADDR

DMA_CH_0_LOCAL_ADDR

DMA_CH_0_TRANS_BYTE_CNT

DMA_CH_0_DESC_PTR

DMA_CH_1_MODE

DMA_CH_1_PCI_ADDR

DMA_CH_1_LOCAL_ADDR

DMA_CH_1_TRANS_BYTE_CNT

DMA_CH_1_DESC_PTR

DMA_CMD_STATUS

DMA_MODE_ARB_REG

DMA_THRESHOLD_REG

***** PCI Configuration Registers *****

DEVICE_VENDOR_ID

STATUS_COMMAND

CLASS_CODE_REVISION_ID

BIST_HDR_TYPE_LAT_CACHE_SIZE

PCI_MEM_BASE_ADDR

PCI_IO_BASE_ADDR

PCI_BASE_ADDR_0

PCI_BASE_ADDR_1

CARDBUS_CIS_PTR

SUBSYS_ID_VENDOR_ID

PCI_BASE_ADDR_LOC_ROM

LAT_GNT_INT_PIN_LINE

***** Local Configuration Registers. *****

PCI_TO_LOC_ADDR_0_RNG

LOC_BASE_ADDR_REMAP_0

MODE_ARBITRATION

BIG_LITTLE_ENDIAN_DESC

PCI_TO_LOC_ROM_RNG

LOC_BASE_ADDR_REMAP_EXP_ROM

BUS_REG_DESC_0_FOR_PCI_LOC

DIR_MASTER_TO_PCI_RNG

LOC_ADDR_FOR_DIR_MASTER_MEM

LOC_ADDR_FOR_DIR_MASTER_IO

PCI_ADDR_REMAP_DIR_MASTER

PCI_CFG_ADDR_DIR_MASTER_IO

PCI_TO_LOC_ADDR_1_RNG

LOC_BASE_ADDR_REMAP_1

BUS_REG_DESC_1_FOR_PCI_LOC

***** Run Time Registers *****

MAILBOX_REGISTER_0

MAILBOX_REGISTER_1

MAILBOX_REGISTER_2

MAILBOX_REGISTER_3

MAILBOX_REGISTER_4

MAILBOX_REGISTER_5

MAILBOX_REGISTER_6

MAILBOX_REGISTER_7

PCI_TO_LOC_DOORBELL

LOC_TO_PCI_DOORBELL

INT_CTRL_STATUS

PROM_CTRL_CMD_CODES_CTRL

DEVICE_ID_VENDOR_ID

REVISION_ID

MAILBOX_REG_0

MAILBOX_REG_1

***** Messaging Queue Registers *****

OUT_POST_Q_INT_STATUS

OUT_POST_Q_INT_MASK

IN_Q_PORT

OUT_Q_PORT

MSG_UNIT_CONFIG

Q_BASE_ADDR

IN_FREE_HEAD_PTR

IN_FREE_TAIL_PTR

IN_POST_HEAD_PTR

IN_POST_TAIL_PTR

OUT_FREE_HEAD_PTR

OUT_FREE_TAIL_PTR

OUT_POST_HEAD_PTR

OUT_POST_TAIL_PTR

Q_STATUS_CTRL_REG

ULONG *pulValue - Pointer to the location where the value read is to be stored

EXAMPLE:

```
int         FileDesc[2];
REG_PARAM   theReg;
ULONG       ulValue;
int         16LCAIOSlot = 1;

theReg.pulValue = &ulValue;
theReg.e16LCAIORegister = BOARD_CTRL_REG;

if (ioctl(FileDesc[16LCAIOSlot], READ_REGISTER, (int) &theReg) ==
          ERROR)
{

   logMsg("Read Register Failed for Slot #%d\n\n", 16LCAIOSlot,
             0, 0, 0, 0, 0);
}
```

5.7.4 WRITE_REGISTER

The WRITE_REGISTER function writes a value to one of the 16LCAIO Registers.

arg PARAMETER:

REG_PARAM *

where REG_PARAM is defined to be

```
typedef struct RegParam
{
    int             e16LCAIORegister;
    ULONG    *pulValue;
} REG_PARAM;
```

and,

int e16LCAIORegister - One of the following registers to write. Refer to the 16LCAIO Hardware documentation for a description of each register.

***** 16LCAIO Registers *****

BOARD_CTRL_REG

INT_CTRL_REG

INPUT_BUFF_CTRL_REG

RATE_A_GEN_REG

RATE_B_GEN_REG

OUTPUT_BUFF_REG

OUTPUT_BUFF_CTRL_REG

SCAN_SYNC_CTRL_REG

DIGITAL_PORT_REG

***** DMA Registers *****

DMA_CH_0_MODE

DMA_CH_0_PCI_ADDR

DMA_CH_0_LOCAL_ADDR

DMA_CH_0_TRANS_BYTE_CNT

DMA_CH_0_DESC_PTR

DMA_CH_1_MODE

DMA_CH_1_PCI_ADDR

DMA_CH_1_LOCAL_ADDR

DMA_CH_1_TRANS_BYTE_CNT

DMA_CH_1_DESC_PTR

DMA_CMD_STATUS

DMA_MODE_ARB_REG

DMA_THRESHOLD_REG

***** PCI Configuration Registers *****

DEVICE_VENDOR_ID

STATUS_COMMAND

CLASS_CODE_REVISION_ID

BIST_HDR_TYPE_LAT_CACHE_SIZE

PCI_MEM_BASE_ADDR

PCI_IO_BASE_ADDR

PCI_BASE_ADDR_0

PCI_BASE_ADDR_1

CARDBUS_CIS_PTR

SUBSYS_ID_VENDOR_ID

PCI_BASE_ADDR_LOC_ROM

LAT_GNT_INT_PIN_LINE

***** Local Configuration Registers. *****

PCI_TO_LOC_ADDR_0_RNG

LOC_BASE_ADDR_REMAP_0

MODE_ARBITRATION

BIG_LITTLE_ENDIAN_DESC

PCI_TO_LOC_ROM_RNG

LOC_BASE_ADDR_REMAP_EXP_ROM

BUS_REG_DESC_0_FOR_PCI_LOC

DIR_MASTER_TO_PCI_RNG

LOC_ADDR_FOR_DIR_MASTER_MEM

LOC_ADDR_FOR_DIR_MASTER_IO

PCI_ADDR_REMAP_DIR_MASTER

PCI_CFG_ADDR_DIR_MASTER_IO

PCI_TO_LOC_ADDR_1_RNG

LOC_BASE_ADDR_REMAP_1

BUS_REG_DESC_1_FOR_PCI_LOC

***** Run Time Registers *****

MAILBOX_REGISTER_0

MAILBOX_REGISTER_1

MAILBOX_REGISTER_2

MAILBOX_REGISTER_3

MAILBOX_REGISTER_4

MAILBOX_REGISTER_5

MAILBOX_REGISTER_6

MAILBOX_REGISTER_7

PCI_TO_LOC_DOORBELL

LOC_TO_PCI_DOORBELL

INT_CTRL_STATUS

PROM_CTRL_CMD_CODES_CTRL

DEVICE_ID_VENDOR_ID

REVISION_ID

MAILBOX_REG_0

MAILBOX_REG_1

***** Messaging Queue Registers *****

OUT_POST_Q_INT_STATUS

OUT_POST_Q_INT_MASK

IN_Q_PORT

OUT_Q_PORT

MSG_UNIT_CONFIG

Q_BASE_ADDR

IN_FREE_HEAD_PTR

IN_FREE_TAIL_PTR

IN_POST_HEAD_PTR

IN_POST_TAIL_PTR

OUT_FREE_HEAD_PTR

OUT_FREE_TAIL_PTR

OUT_POST_HEAD_PTR

OUT_POST_TAIL_PTR

Q_STATUS_CTRL_REG

ULONG *pulValue - Pointer to the location containing the value to be written.

EXAMPLE:

```
int   FileDesc[2];
REG_  PARAM theReg;
ULONG ulValue = 0xAAAA;
int   16LCAIOSlot = 1;

theReg.pulValue = &ulValue;
theReg.e16LCAIORegister = OUT_Q_PORT;

if (ioctl(FileDesc[16LCAIOSlot], WRITE_REGISTER, (int) &theReg) ==
         ERROR)
{
   logMsg("Write Register Failed for Slot #%d\n\n", 16LCAIOSlot,

            0, 0, 0, 0, 0);
}
```

5.7.5 START_DMA

The START_DMA function configures the 16LCAIO DMA registers for a DMA transfer to/from the board, and then starts the transfer.

arg PARAMETER:

DMA_PARAM *

where DMA_PARAM is defined to be

```
typedef struct DMAParam
{
    int     DMAChannel;
    ULONG   ulDMAMode;
    ULONG   ulDMALocalAddress;
    ULONG   ulDMAByteCount;
    ULONG   ulDMADescriptorPtr;
    ULONG   ulDMAArbitration;
    ULONG   ulDMAThreshold;
} DMA_PARAM;
```

and,

int DMAChannel - DMA channel to perform transfer on. Must be one of the following:

- DMA_CHAN_0
- DMA_CHAN_1

ULONG ulDMAMode - Value to be written to the 16LCAIO DMA Mode Register.

ULONG ulDMALocalAddress - Value to be written to the 16LCAIO DMA Local Address Register. Data returned is little endian and may need to be byte/word swapped.

ULONG ulDMAByteCount - Value to be written to the 16LCAIO DMA Byte Count Register.

ULONG ulDMADescriptorPtr - Value to be written to the 16LCAIO DMA Descriptor Pointer Register.

ULONG ulDMAArbitration - Value to be written to the 16LCAIO DMA Arbitration Register.

ULONG ulDMAThreshold - Value to be written to the 16LCAIO DMA Threshold Register.

See the PLX-PCI PCI Bus Master Interface Data Sheet for a description of the DMA registers.

DMA EXAMPLE:

```
#define     DWORD_COUNT 80
int         iIndex, FileDesc[2], 16LCAIOSlot = 1;
DMA_PARAM   DMAParameters;
ULONG       pulBuffer[DWORD_COUNT];
REG_PARAM   theReg;
ULONG       ulValue;

/* Setup parameters to perform a DMA Read from the 16LCAIO Board. */
DMAParameters.DMAChannel         = 0;
DMAParameters.ulDMAMode          = 0x943;
DMAParameters.ulDMALocalAddress  = (ULONG) pulBuffer;
DMAParameters.ulDMAByteCount     = DWORD_COUNT * 4;
DMAParameters.ulDMADescriptorPtr = 0xA;     /* DMA Read */
DMAParameters.ulDMAArbitration   = 0;
DMAParameters.ulDMAThreshold     = 0;

if (ioctl(FileDesc[16LCAIOSlot], START_DMA, (int) &DMAParameters) ==
ERROR)
{
   logMsg("Start DMA Failed for Slot #%d\n\n", 16LCAIOSlot,
            0, 0, 0, 0, 0);
}

/* Wait for the DMA to Complete. */
theReg.pulValue      = &ulValue;
theReg.e16LCAIORegister = DMA_CMD_STATUS;
do
{
    if(ioctl(FileDesc[16LCAIOSlot], READ_REGISTER, (int)&theReg) ==
    ERROR)
    {
      logMsg("Read Register Failed for Slot #%d\n\n", 16LCAIOSlot,
               0, 0, 0, 0, 0);
      break;
   }
} while (! (ulValue & 0x10));
```

```
/* Clear the DMA channel 0/1 command/status register.
*/
ulValue = 0;
theReg.pulValue      = &ulValue;
theReg.e16LCAIORegister = DMA_CMD_STATUS;

if (ioctl(FileDesc[16LCAIOSlot], WRITE_REGISTER, (int) &theReg) ==
          ERROR)
{
          logMsg("Write Register Failed\n\n",
                 0, 0, 0, 0, 0, 0);
}
```

5.7.6 REG_FOR_INT_NOTIFY

The REG_FOR_INT_NOTIFY function will register or unregister for notification that an interrupt has occurred on the 16LCAIO board. If this function is called with a pointer to a subroutine, that routine will be invoked when a 16LCAIO interrupt occurs. If a function is currently registered for interrupt notification and is called with a NULL pointer, the function will no longer be called when an interrupt occurs. The parameter sent to the notification routine will be the slot number of the 16LCAIO Board that has interrupted and will be one of the following:

- 16LCAIO_PMC1
- 16LCAIO_PMC2

Note that the internal driver interrupt handler will clear interrupts after calling the user supplied ISR.

arg PARAMETER:

int (*intHandler)(int) - Pointer to a routine to handle the interrupt notification or a NULL pointer if the caller wants to unregister for interrupt notification.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;

int intHandler(ULONG ulSlotNum)
{
   REG_PARAM   theReg;
   ULONG       ulValue;

/* execute interrupt control here */

   return (0);

}  /* intHandler */



/* Request notification on the user selected conditions. */
if (ioctl(FileDesc[16LCAIOSlot], REG_FOR_INT_NOTIFY, (int)
          intHandler) == ERROR)
{
```

```
    logMsg("Request Interrupt Notification Failed\n\n",0,0,0,0,0,0 );
}
```

5.7.7 GET_DEVICE_ERROR

The GET_DEVICE_ERROR function will return the error that occurred on the last call to one of the 16LCAIO Device Driver entry points. Whenever a driver function is called and it returns an error, this function may be called to determine the cause of the error.

arg PARAMETER:

int * - Pointer to the location of where the error code is to be written. It will be one of the following:

NO_ERR - No Error Occurred.

INVALID_PARAMETER_ERR - An Invalid Parameter was sent to driver.

RESOURCE_ERR - The driver could not obtain a resource (memory or semaphore) to perform its function.

BOARD_ACCESS_ERR - Failure occurred when the 16LCAIODrvInstall function fails when probing the 16LCAIO card's Board Status Register.

DEVICE_ADD_ERROR - Failure occurred when the 16LCAIODrvInstall function fails when trying to add device to the VxWorks Operating System.

ALREADY_OPEN_ERROR - A call to the open driver access routine for a device that is already open.

INVALID_DRV_NUM_ERR - Returned from the 16LCAIODrvInstall function if an invalid driver number was obtained when trying to add the device driver to the VxWorks operating system. Also returned from the 16LCAIODrvRemove function if the driver failed to remove the device driver from the VxWorks operating system.

ALREADY_INSTALLED_ERR - Returned from the 16LCAIODrvInstall function if the driver has already been installed.

PCI_CONFIG_ERR - Returned from the 16LCAIODrvInstall function if a read or write of a PCI Configuration Register fails.

INVALID_BOARD_STATUS_ERR - Returned from the 16LCAIODrvInstall function if an invalid board status is read from the 16LCAIO Board.

FIFO_BUFFER_ERR - If during a read()/write() transaction more data is requested than what is available, the driver will return the number of bytes that could be read/written along with throwing this error condition.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int Status;

/* Send the Get Device Error Code Command for this channel   */
if (ioctl(FileDesc[16LCAIOSlot], GET_DEVICE_ERROR, (int) &Status) ==
ERROR)
{
   logMsg("Get Device Error Code Failed for Slot #%d\n\n",
          16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.8 READ_MODE_CONFIG

The READ_MODE_CONFIG function will configure the driver for the type of read() from the input FIFO to be performed. There are two types of reads. The first being referred to as SCAN_MODE where each sample is read out of the input FIFO one at a time and put into the user buffer given. The other type of read is referred to as DMA_MODE, which utilizes the DMA capability of the board. This mode must be set before attempting to read the analog inputs.

arg PARAMETER:

int * - Pointer to one of the following values:

- SCAN_MODE
- DMA_MODE

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int iMode;

iMode = DMA_MODE;

if (ioctl(FileDesc[16LCAIOSlot], READ_MODE_CONFIG, (int) &iMode) ==
ERROR)
{
   logMsg("Read Mode Configuration Failed for Slot #%d\n\n",
   16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.9 WRITE_MODE_CONFIG

The WRITE_MODE_CONFIG function will configure the driver for the type of write() to the output FIFO to be performed. There are two types of writes. The first type is referred to as SCAN_MODE where each sample is written to the output FIFO one at a time from the given user buffer. The other type of write is referred to as DMA_MODE, which utilizes the DMA capability of the board. This mode must be set before attempting to write the analog outputs.

arg PARAMETER:

int * - Pointer to one of the following values:

- SCAN_MODE
- DMA_MODE

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int iMode;

iMode = DMA_MODE;

if (ioctl(FileDesc[16LCAIOSlot], WRITE_MODE_CONFIG, (int) &iMode) ==
ERROR)
{
   logMsg("Write Mode Configuration Failed for Slot #%d\n\n",
   16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.10 INPUT_CONFIG

The INPUT_CONFIG function will arrange input channels into single-ended or differential mode.

arg PARAMETER:

int * - Pointer to one of the following values:

- SINGLE_ENDED
- DIFFERENTIAL

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int iMode;

iMode = DIFFERENTIAL;

if (ioctl(FileDesc[16LCAIOSlot], INPUT_MODE_CONFIG, (int) &iMode) ==
ERROR)
{
   logMsg("Input Configuration Failed for Slot #%d\n\n",
   16LCAIOSlot,
   0, 0, 0, 0, 0);
}
```

5.7.11 INPUT_TEST

The INPUT_TEST function will perform a system level validation of operation precision. There are several tests, such voltage reference test, zero input test, and monitoring an output channel. During the positive reference test, an internal voltage reference is connected to all input channels. The zero input test consists of all input channels being connected to the internal ground. And, one analog output channel can be monitored by connected to an analog input channel.

arg PARAMETER:

int * - Pointer to one of the following values:

- ZERO
- POSITIVE_REF
- MONITOR_OUT_0
- MONITOR_OUT_1
- MONITOR_OUT_2
- MONITOR_OUT_3

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int Test;

Test = ZERO;

if (ioctl(FileDesc[16LCAIOSlot], INPUT_TEST, (int) &Test) == ERROR)
{
   logMsg("Input Test Failed for Slot #%d\n\n", 16LCAIOSlot,
            0, 0, 0, 0, 0);
}
```

5.7.12 VOLTAGE_RANGE

The VOLTAGE_RANGE function will set the voltage range for the analog inputs and outputs (default is ±10V).

arg PARAMETER:

int * - Pointer to one of the following values:

- PLUS_MIN_2_5
- PLUS_MIN_5
- PLUS_MIN_10

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   *Range;

Range = PLUS_MIN_2_5;

if (ioctl(FileDesc[16LCAIOSlot], VOLTAGE_RANGE,  (int) &Range) ==
ERROR)
{
   logMsg("Voltage Range Selection Failed for Slot #%d\n\n",
       16LCAIOSlot,
            0, 0, 0, 0, 0);
}
```

5.7.13 IO_DATA_FORMAT

The IO_DATA_FORMAT function will set the analog input/output data format to be offset binary or two's compliment.

arg PARAMETER:

int * - Pointer to one of the following values:

- TWOS_COMP
- BINARY_OFFSET

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int *Format;

Format = BINARY_OFFSET;

if (ioctl(FileDesc[16LCAIOSlot], IO_DATA_FORMAT, (int) &Format) ==
          ERROR)
{
   logMsg("IO Data Format Failed for Slot #%d\n\n",
      16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.14 OUTPUT_CLK_MODE

The OUTPUT_CLK_MODE function will select the output clocking mode to be either simultaneous or sequential.

arg PARAMETER:

int * - Pointer to one of the following values:

- SEQUENTIAL
- SIMULTANEOUS

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int *Mode;

Mode = SIMULTANEOUS;

if (ioctl(FileDesc[16LCAIOSlot], OUTPUT_CLK_MODE, (int) &Mode) ==
          ERROR)
{
   logMsg("Output Clock Mode Failed for Slot #%d\n\n",
         16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.15 OUTPUT_SYNC_MODE

The OUTPUT_SYNC_MODE function will select the output sync mode to be either continuous or burst.

arg PARAMETER:

int * - Pointer to one of the following values:

- CONTINUOUS
- BURST

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int *Mode;

Mode = CONTINUOUS;

if (ioctl(FileDesc[16LCAIOSlot], OUTPUT_SYNC_MODE, (int) &Mode) ==
          ERROR)
{
   logMsg("Output Sync Mode Failed for Slot #%d\n\n",
         16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.16 FUNCTIOIN_LOOPING

The FUNCTION_LOOPING function will sets the analog outputs to looping mode. Function looping occurs when an analog output buffer is "closed," so that the output data circulates, i.e. loops. Otherwise, the buffer is said to be "open", or non-looping.

arg PARAMETER:

int * - Pointer to one of the following values:

- OPEN_BURST
- PERIODIC

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int *Mode;

Mode = PERIODIC;

if (ioctl(FileDesc[16LCAIOSlot], FUNCTION_LOOPING, (int) &Mode) ==
          ERROR)
{
   logMsg("Function Looping Failed for Slot #%d\n\n",
         16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.17 ONE_BURST_OUTPUTS

The ONE_BURST_OUTPUTS function will burst the analog outputs provided that the bursting has been enabled through the OUTPUT_SYNC_MODE function.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;

if (ioctl(FileDesc[16LCAIOSlot], ONE_BURST_OUTPUTS, 0) == ERROR)
{
   logMsg("Burst Outputs Failed for Slot #%d\n\n",
            16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.18 ONE_SCAN_INPUTS

The ONE_SCAN_INPUTS function will scan the analog inputs provided that it has been selected as the analog input scan clock source through the INPUT_CLK_SRC function.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;

if (ioctl(FileDesc[16LCAIOSlot], ONE_SCAN_INPUTS, 0) == ERROR)
{
   logMsg("Scan Inputs Failed for Slot #%d\n\n",
            16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.19 START_AUTOCAL

The START_AUTOCAL function initiates an autocalibration operation. No current settings are changed during the operation. It is recommended that an autocalibration be ran after power warm-up and initialization.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;

if (ioctl(FileDesc[16LCAIOSlot], START_AUTOCAL, 0) ==
   ERROR)
{
    logMsg("Start Autocalibration Failed for Slot #%d\n\n",
    16LCAIOSlot,
            0, 0, 0, 0, 0);
}
```

5.7.20 AUTO_PASS

The AUTO_PASS function will return the autocalibration status.

arg PARAMETER:

int * - Pointer to the location of where the status code is to be written. It will be one of the following:

- **NO_PASS**
- **PASSED**

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int Status;

if (ioctl(FileDesc[16LCAIOSlot], AUTO_PASS, (int) &Status) == ERROR)
{
   logMsg("Get Autocalibration Status Failed for Slot #%d\n\n",
    16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.21 SCAN_SIZE

The SCAN_SIZE function will set the number of analog input channels to be scanned. There are four standard sizes 4, 8, 16, and 32 channels. But, there are cases, which will be discussed later, where just one or two channels are chosen to be scanned. Regardless of size, each scan starts with analog input channel 0 and continues in ascending order.

arg PARAMETER:

int * - Pointer to one of the following values:

- FOUR_CHANS
- EIGHT_CHANS
- SIXTEEN_CHANS
- THIRTY_TWO_CHANS

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int Size;

Size = EIGHT_CHANS;

if (ioctl(FileDesc[16LCAIOSlot], SCAN_SIZE, (int)&Size) == ERROR)
{
      logMsg("Scan Size Selection Failed for Slot
         #%d\n\n", 16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.22 INPUT_CLK_SRC

The INPUT_CLK_SRC function selects the clocking source for the analog input scan. The source can be either rate generator, an external source, or the input sync control, which initiates an analog input scan when the ONE_SCAN_INPUTS function is issued. The Rate-A generator is selected as default, and this generator is defaulted to disabled; therefore, the user should be aware and make proper alterations to accommodate.

arg PARAMETER:

int * - Pointer to one of the following values:

- RATE_A_GEN_OUTPUT
- RATE_B_GEN_OUTPUT
- EXT_SYNC_IN_LINE
- BCR_IN_SYNC_CTRL

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   Source;

Source = RATE_B_GEN_OUTPUT;

if (ioctl(FileDesc[16LCAIOSlot], INPUT_CLK_SRC, (int) &Source) ==
ERROR)
{
     logMsg("Input Clock Source Selection Failed for Slot #%d\n\n",
     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.23 OUTPUT_CLK_SRC

The OUTPUT_CLK_SRC function selects the clocking source for the analog outputs. The source can be either rate generator, an external source, or disabled. This source is ignored when in sequential output mode as selected in the OUTPUT_CLK_MODE function.

arg PARAMETER:

int * - Pointer to one of the following values:

- RATE_A_GEN_OUTPUT
- RATE_B_GEN_OUTPUT
- EXT_SYNC_IN_LINE
- DISABLED

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   Source;

Source = DISABLED;

if (ioctl(FileDesc[16LCAIOSlot], OUTPUT_CLK_SRC, (int) &Source) ==
ERROR)
{
     logMsg("Output Clock Source Selection Failed for Slot #%d\n\n",
     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.24 BURST_SYNC_SRC

The BURST_SYNC_SRC function selects the burst sync source for the analog outputs. The source can be either rate generator, an external source, or the output sync control, which initiates an analog output burst when the ONE_BURST_OUTPUTS function is issued. The output sync control is default.

arg PARAMETER:

int * - Pointer to one of the following values:

- RATE_A_GEN_OUTPUT
- RATE_B_GEN_OUTPUT
- EXT_SYNC_IN_LINE
- BCR_OUT_SYNC_CTRL

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   Source;

Source = RATE_A_GEN_OUTPUT;

if (ioctl(FileDesc[16LCAIOSlot], BURST_SYNC_SRC, (int) &Source) ==
ERROR)
{
     logMsg("Burst Sync Source Selection Failed for Slot #%d\n\n",
     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.25 EXT_SYNC_SIGNAL_SRC

The EXT_SYNC_SIGNAL_SRC function selects the signal source for the external sync output line. The source can be the analog input scan clock, the analog output sync, an external source (pass-thru mode), or disabled.

arg PARAMETER:

int * - Pointer to one of the following values:

- INPUT_SCAN_CLK
- OUTPUT_SYNC
- EXT_SYNC_IN_LINE
- DISABLED

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   Source;

Source = OUTPUT_SYNC;

if (ioctl(FileDesc[16LCAIOSlot], EXT_SYNC_SIGNAL_SRC, (int) &Source)
== ERROR)
{
     logMsg("External Sync Source Selection Failed for Slot
     #%d\n\n", 16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.26 RATE_B_CLK_SRC

The RATE_B_CLK_SRC function selects the clock input source for the Rate-B generator. The source can be the either master clock or the Rate-A generator output. Because it is possible for the clocking source be from the Rate-A generator, the generators can be "cascaded." This means that the analog inputs and outputs can be clocked in a fixed integer time ratio. Another reason for using the Rate-A generator for the clocking source would be to produce lower clocking frequencies.

arg PARAMETER:

int * - Pointer to one of the following values:

- MASTER_CLK
- RATEA_GEN_OUTPUT

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   Source;

Source = RATEA_GEN_OUTPUT;

if (ioctl(FileDesc[16LCAIOSlot], RATE_B_CLK_SRC, (int) &Source) ==
ERROR)
{
     logMsg("Rate-B Clocking Source Selection Failed for Slot
     #%d\n\n", 16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.27 INPUT_SCAN_MODE

The INPUT_SCAN_MODE function selects the input scanning mode. The mode is either multiple channels or one single channel. This function is ignored when a two-channel scan size is selected.

arg PARAMETER:

int * - Pointer to one of the following values:

- MULTI_CHANNEL
- SINGLE_CHANNEL

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   Mode;

Mode = SINGLE_CHANNEL;

if (ioctl(FileDesc[16LCAIOSlot], INPUT_SCAN_MODE, (int) &Mode) ==
ERROR)
{
     logMsg("Input Scan Mode Selection Failed for Slot #%d\n\n",
     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.28 SINGLE_CHAN

The SINGLE_CHAN function sets the single analog input channel to be scanned. It is ignored if in Multiple-Channel or Two-Channel scanning mode.

arg PARAMETER:

- int - Pointer to analog input channel to be scanned.

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   Channel;

Channel = 4;

if (ioctl(FileDesc[16LCAIOSlot], SINGLE_CHAN, Channel) == ERROR)
{
     logMsg("Single Channel Selection Failed for Slot #%d\n\n",
     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.29 TWO_CHAN_SCAN

The TWO_CHAN_SCAN function sets the analog input scan size for a two-channel scan. It overrides the input scanning mode set in the INPUT_SCAN_MODE function. Analog input channel 0 and 1 are the scanned channels.

arg PARAMETER:

- Not Used.

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;

if (ioctl(FileDesc[16LCAIOSlot], TWO_CHAN_SCAN, 0) == ERROR)
{
     logMsg("Two-Channel Scan Selection Failed for Slot #%d\n\n",
     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.30 ENABLE_PCI_INTERRUPTS

The ENABLE_PCI_INTERRUPTS function enables the PCI interrupts in order to have a local interrupt request be generated.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;

if (ioctl(FileDesc[16LCAIOSlot], ENABLE_PCI_INTERRUPTS, 0) == ERROR)
{
   logMsg("PCI Interrupt Enable Failed for Slot #%d\n\n",
            16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.31 DISABLE_PCI_INTERRUPTS

The DISABLE_PCI_INTERRUPTS function disables the PCI interrupts.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;

if (ioctl(FileDesc[16LCAIOSlot], DISABLE_PCI_INTERRUPTS, 0) ==
ERROR)
{
   logMsg("PCI Interrupts Disable Failed for Slot #%d\n\n",
           16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.32 SET_RATE_A_GEN

The SET_RATE_A_GEN function will set the Rate-A at which the analog input/output channels are scanned and sampled. This function uses a user-specified divisor, iNrate. The Rate-A generator calculates the clock frequency as:

Frequency (Hz) = 24,000,000 / iNrate

It is advised that the iNrate value remains more than 50h (80 decimal).

arg PARAMETER:

int * - Pointer to the integer used in calculation.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int iNrate;

iNrate = 0x0100;

/* Program Rate-A Generator. */
if (ioctl(FileDesc[16LCAIOSlot], SET_RATE_A_GEN, iNrate) == ERROR)
{
   logMsg("Set Rate-A Generator Failed\n\n", 0, 0, 0,
          0, 0, 0);
}
```

5.7.33 SET_RATE_B_GEN

The SET_RATE_B_GEN function will set the Rate-B at which the analog input/output channels are scanned and sampled. This function uses a user-specified divisor, iNrate. The Rate-B generator calculates the clock frequency as:

Frequency (Hz) = 24,000,000 / iNrate

It is advised that the iNrate value remains more than 50h (80 decimal).

arg PARAMETER:

int * - Pointer to the integer used in calculation.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int iNrate;

iNrate = 0x0100;

/* Program Rate-B Generator. */
if (ioctl(FileDesc[16LCAIOSlot], SET_RATE_B_GEN, iNrate) == ERROR)
{
   logMsg("Set Rate-B Generator Failed\n\n", 0, 0, 0,
          0, 0, 0);
}
```

5.7.34 ENABLE_RATE_A_GEN

The ENABLE_RATE_A_GEN function will enable the Rate-A generator.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], ENABLE_RATE_A_GEN, 0) == ERROR)
{
   logMsg("Rate-A Generator Enable Failed for Slot #%d\n\n",
               16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.35 DISABLE_RATE_A_GEN

The DISABLE_RATE_A_GEN function will disable the Rate-A generator.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], DISABLE_RATE_A_GEN, 0) == ERROR)
{
   logMsg("Rate-A Generator Disable Failed for Slot #%d\n\n",
               16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.36 ENABLE_RATE_B_GEN

The ENABLE_RATE_B_GEN function will enable the Rate-B generator.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], ENABLE_RATE_B_GEN, 0) == ERROR)
{
   logMsg("Rate-B Generator Enable Failed for Slot #%d\n\n",
               16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.37 DISABLE_RATE_B_GEN

The DISABLE_RATE_B_GEN function will disable the Rate-B generator.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], DISABLE__RATE_B_GEN, 0) == ERROR)
{
   logMsg("Rate-B Generator Disable Failed for Slot #%d\n\n",
               16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.38 SET_IN_THRESHOLD

The SET_IN_THRESHOLD function sets the analog input buffer threshold. It can be used along with the threshold flag, which can be monitored using the BUFF_IN_STATUS_FLAG function, to assess the status of the buffer. This value must not exceed the capacity of the input buffer (32K values).

arg PARAMETER:

int * - Pointer to the threshold value.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int Value;

Value = 0x0AAA;

if (ioctl(FileDesc[16LCAIOSlot], SET_IN_THRESHOLD, (int)& Value) ==
          ERROR)
{
   logMsg("Set Input Buffer Threshold Failed for Slot #%d\n\n",
                  16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.39 SET_OUT_THRESHOLD

The SET_OUT_THRESHOLD function sets the analog output buffer threshold. It can be used along with the threshold flag, which can be monitored using the BUFF_OUT_STATUS_FLAG function, to assess the status of the buffer. . This value must not exceed the capacity of the output buffer (32K values).

arg PARAMETER:

int * - Pointer to the threshold value.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int Value;

Value = 0x1BBB;

if (ioctl(FileDesc[16LCAIOSlot], SET_OUT_THRESHOLD, (int)& Value) ==
          ERROR)
{
   logMsg("Set Output Buffer Threshold Failed for Slot #%d\n\n",
                  16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.40 CLEAR_IN_BUFFER

The CLEAR_IN_BUFFER function clears the analog input buffer.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], CLEAR_IN_BUFFER, 0) == ERROR)
{
   logMsg("Clear Input Buffer Failed for Slot #%d\n\n",
         16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.41 CLEAR_OUT_BUFFER

The CLEAR_OUT_BUFFER function clears the analog output buffer.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], CLEAR_OUT_BUFFER, 0) == ERROR)
{
   logMsg("Clear Output Buffer Failed for Slot #%d\n\n",
            16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.42 BUFF_IN_STATUS_FLAG

The BUFF_IN_STATUS_FLAG function will return the status of the analog input buffer.

arg PARAMETER:

- int * - Pointer to the status

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int Status;

if (ioctl(FileDesc[16LCAIOSlot], BUFF_IN_STATUS_FLAG, (int) &Status)
== ERROR)
{
   logMsg("Get Input Buffer Status Failed for Slot #%d\n\n",
          16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.43 BUFF_OUT_STATUS_FLAG

The BUFF_OUT_STATUS_FLAG function will return the status of the analog output buffer.

arg PARAMETER:

- int * - Pointer to the status

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int Status;

if (ioctl(FileDesc[16LCAIOSlot], BUFF_OUT_STATUS_FLAG, (int)
&Status) == ERROR)
{
   logMsg("Get Output Buffer Status Failed for Slot #%d\n\n",
          16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.44 SELECT_DIGITAL_OUT_HI

The SELECT_DIGITAL_OUT_HI function will set the direction of the upper digital byte in the digital I/O port register to be outputs.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], SELECT_DIGITAL_OUT_HI, 0) == ERROR)
{
   logMsg("Select Digital High Byte Output Failed for Slot #%d\n\n",
                     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.45 SELECT_DIGITAL_OUT_LO

The SELECT_DIGITAL_OUT_LO function will set the direction of the lower digital byte in the digital I/O port register to be outputs.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], SELECT_DIGITAL_OUT_LO, 0) == ERROR)
{
   logMsg("Select Digital Low Byte Output Failed for Slot #%d\n\n",
                     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.46 SELECT_DIGITAL_IN_HI

The SELECT_DIGITAL_IN_HI function will set the direction of the upper digital byte in the digital I/O port register to be inputs.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], SELECT_DIGITAL_IN_HI, 0) == ERROR)
{
   logMsg("Select Digital High Byte Input Failed for Slot #%d\n\n",
                     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.47 SELECT_DIGITAL_IN_LO

The SELECT_DIGITAL_IN_LO function will set the direction of the lower digital byte in the digital I/O port register to be inputs.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], SELECT_DIGITAL_IN_LO, 0) == ERROR)
{
   logMsg("Select Digital Low Byte Input Failed for Slot #%d\n\n",
                     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.48 SELECT_DIGITAL_IN_LO

The SELECT_DIGITAL_IN_LO function will set the direction of the lower digital byte in the digital I/O port register to be inputs.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], SELECT_DIGITAL_IN_LO, 0) == ERROR)
{
   logMsg("Select Digital Low Byte Input Failed for Slot #%d\n\n",
                     16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.49 SET_DIGITAL_AUX_OUT

The SET_DIGITAL_AUX_OUT function will set the auxiliary output line in the digital I/O port register.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], SET_DIGITAL_AUX_OUT, 0) == ERROR)
{
   logMsg("Set Digital Auxiliary Output Line Failed for Slot
#%d\n\n",                    16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.50 CLEAR_DIGITAL_AUX_OUT

The CLEAR_DIGITAL_AUX_OUT function will clear the auxiliary output line in the digital I/O port register.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], CLEAR_DIGITAL_AUX_OUT, 0) == ERROR)
{
   logMsg("Clear Digital Auxiliary Output Line Failed for Slot
#%d\n\n",                    16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.51 CHECK_DIGITAL_AUX_IN

The CHECK_DIGITAL_AUX_IN function will return the auxiliary input line in the digital I/O port register status.

arg PARAMETER:

- int * - Pointer to the status

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;
int Status;

if (ioctl(FileDesc[16LCAIOSlot], CHECK_DIGITAL_AUX_IN, (int)
&Status) == ERROR)
{
   logMsg("Get Digital Auxiliary Input Line Status Failed for Slot
#%d\n\n",
          16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.52 IRQ0_INT_SRC

The IRQ0_INT_SRC function will set the interrupt condition for the IRQ0. The conditions are for completion of an autocalibration operation, a transition of the auxiliary input line, or just idle. The user should be aware that all IRQs could be combined for singular interrupts (refer to the General Standards PMC-16LCAIO User's Manual).

arg PARAMETER:

int * - Pointer to one of the following values:

- IDLE
- AUTOCAL_COMPLETE
- AUX_IN_TO_HIGH
- AUX_IN_TO_LOW

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   Source;

Source = AUTOCAL_COMPLETE;

if (ioctl(FileDesc[16LCAIOSlot], IRQ0_INT_SRC, (int) &Source) ==
ERROR)
{
   logMsg("Interrupt Selection for IRQ0 Failed for Slot #%d\n\n",
16LCAIOSlot,
            0, 0, 0, 0, 0);
}
```

5.7.53 IRQ1_INT_SRC

The IRQ1_INT_SRC function will set the interrupt condition for the IRQ1. The conditions are a transition of the analog input buffer threshold flag or just idle.

arg PARAMETER:

int * - Pointer to one of the following values:

- IDLE
- IN_T_HOLD_TO_HIGH
- IN_T_HOLD_TO_LOW

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   Source;

Source = IN_T_HOLD_TO_LOW;

if (ioctl(FileDesc[16LCAIOSlot], IRQ1_INT_SRC, (int) &Source) ==
ERROR)
{
    logMsg("Interrupt Selection for IRQ1 Failed for Slot #%d\n\n",
    16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.54 IRQ2_INT_SRC

The IRQ2_INT_SRC function will set the interrupt condition for the IRQ2. The conditions are for a transition of the analog output buffer threshold flag, an analog output burst completion, or just idle.

arg PARAMETER:

int * - Pointer to one of the following values:

- IDLE
- OUT_T_HOLD_TO_HIGH
- OUT_T_HOLD_TO_LOW
- BURST_COMPLETE

EXAMPLE:

```
int   FileDesc[2];
int   16LCAIOSlot = 1;
int   Source;

Source = BURST_COMPLETE;

if (ioctl(FileDesc[16LCAIOSlot], IRQ2_INT_SRC, (int) &Source) ==
ERROR)
{
    logMsg("Interrupt Selection for IRQ2 Failed for Slot #%d\n\n",
    16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.55 CLEAR_GRP0_INT_REQ

The CLEAR_GRP0_INT_REQ function clears the interrupt request flag for group 0 after an interrupt has occurred.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], CLEAR_GRP0_INT_REQ, 0) == ERROR)
{
    logMsg("Clear Group 0 Interrupt Request Flag Failed for Slot
    #%d\n\n", 16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.56 CLEAR_GRP1_INT_REQ

The CLEAR_GRP1_INT_REQ function clears the interrupt request flag for group 1 after an interrupt has occurred.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], CLEAR_GRP1_INT_REQ, 0) == ERROR)
{
    logMsg("Clear Group 1 Interrupt Request Flag Failed for Slot
    #%d\n\n", 16LCAIOSlot, 0, 0, 0, 0, 0);
}
```

5.7.57 CLEAR_GRP2_INT_REQ

The CLEAR_GRP2_INT_REQ function clears the interrupt request flag for group 2 after an interrupt has occurred.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 16LCAIOSlot = 1;


if (ioctl(FileDesc[16LCAIOSlot], CLEAR_GRP2_INT_REQ, 0) == ERROR)
{
    logMsg("Clear Group 2 Interrupt Request Flag Failed for Slot
    #%d\n\n", 16LCAIOSlot, 0, 0, 0, 0, 0);
}
```