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-6SDI hosted on PowerPC and 80x86 Processors

Document number:	9005005	Revision:	1.0	Date: 8/11/99
Engineering Approval:				Date:
Quality Representative Approval:				Date:

Acknowledgments

1 Scope

The purpose of this document is to describe how to interface with the PMC-6SDI VxWorks Device Driver developed by Chandler/May, Incorporated (CMI). This software provides the interface between "Application Software" and the 6SDI 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 6SDI Driver Software executes under control of the VxWorks operating system. The 6SDI is implemented as a standard VxWorks device driver written in the 'C' programming language. The 6SDI 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. Examples are the Force PPC/PowerCore-6604 CPU board, the Motorola MVME1600, MVME2300, MVME2400, MVME2600, and MVME2700 series boards, and the SCI JTT 686 CPU board.

2 Hardware Overview

The General Standards Corporation (GSC) 6SDI board is a single-width analog input interface that fits into a PCI Mezzanine Card slot. This board provides 6 16-bit analog input channels. These channels can be customized as single-ended or differential input channels via software configuration. Each channel can be controlled by separate clock rates or can be synchronized to perform at the same rate. Because the board is capable of clock synchronization, it is possible to connect multiple boards daisy-chained together. It also provides for minimum off-line maintenance by providing calibration and self-testing functions.

The 6SDI board includes two rate generators and a DMA controller. Each rate controller is provided to control the rates at which input channels are scanned. The DMA transfers are supported when the board is acting as the bus master.

The configuration of the interrupting capability of the 6SDI board is described in the hardware manual for the board. The 6SDI 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-6SDI 16-Bit, 6-Channel Sigma-Delta Analog Input PMC Board User's Manual – Revision 3, 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 6SDI 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 that 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:

6sdi_drv.o	Relocatable/loadable module for the device driver.
6sdi_menu.o	Relocatable/loadable module for the sample menu tool.

5 Driver Interface

The 6SDI Driver conforms to the device driver standards required by the VxWorks Operating System and contains the following standard driver entry points.

- 6SDIDrvInstall() - Installs the device driver for use with multiple 6SDI Cards
- 6SDIDrvRemove() - Removes the device driver from use
- open() - opens a driver interface to one 6SDI Card
- close() - closes a driver interface to one 6SDI Card
- read() - reads data received from a 6SDI Card
- ioctl() - performs various control and setup functions on the 6SDI Card

The 6SDI Device Driver provides a standard input system interface to the GSC PMC-6SDI card for VxWorks applications that 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 6SDI card and device driver. It can also be used to break the learning curve somewhat for programming the 6SDI device.

If the user wishes to use the 6SDI 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 6SDI interrupts through software control except for PCI interrupts controlled through the Shared Runtime - Interrupt Control/Status register. 6SDI 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 6SDI 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 6SDI 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 6SDIDrvInstall()

The 6SDIDrvInstall () 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 6SDIDrvRemove() function.

The 6SDIDrvInstall () 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 6SDI 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 6SDI 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 6SDIDrvInstall(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 6SDI VxWorks Device Driver. */
iStatus = 6SDIDrvInstall(TRUE);
```

5.2 6SDIDrvRemove()

The 6SDIDrvRemove() function is used to remove the 6SDI Device Driver from the VxWorks operating system. This function should only be called after a call to the 6SDIDrvInstall() function. The 6SDIDrvRemove() function closes all the open devices for each PMC slot and removes the device driver from the operating system.

PROTOTYPE:

extern int 6SDIDrvRemove(void);

Returns OK on success and ERROR on failure

EXAMPLE:

```
STATUS iStatus;

/* Remove the 6SDI Driver */
iStatus = 6SDIDrvRemove();
```

5.3 open()

The open() function is the standard VxWorks entry point to open a connection to a 6SDI Card in one PMC Slot. This function may only be called after a call to the 6SDIDrvInstall() function is made.

PROTOTYPE:

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

Where:

cName - name of the device being opened

iFlags - access flag for cName (not used).

iMode - permissions of cName (not used).

Returns OK on success and ERROR on failure

EXAMPLE:

```
int         FileDesc[2];
LOCAL char  *devName;
int         6SDISlot = 1;


/*   open the 6SDI device for slot 1   */
FileDesc[6SDISlot] = open(devName[6SDISlot], O_RDWR, 0644);

if (FileDesc[6SDISlot] == ERROR)
{
   logMsg("Cannot Open Device Error %s\n\n",
            (int) devName[6SDISlot], 0, 0, 0, 0, 0);
}
```

5.4 close()

The close() function is the standard VxWorks entry point to close a connection to a 6SDI Card in one PMC Slot. This function should only be called after the open function has been successfully called for a slot where a 6SDI Card resides. The close function closes the interface to a 6SDI 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 6SDISlot = 1;

/*  close the device on slot 2  */
if  (close(FileDesc[6SDISlot]) == ERROR)
{
   logMsg("Close Error for Slot #%d\n\n", 6SDISlot, 0, 0, 0, 0, 0);
}
FileDesc[6SDISlot] = ERROR;
```

5.5 read()

The read() function is the standard VxWorks entry point to receive channel data from a 6SDI 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 6SDI Card resides

The 6SDI 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. Regardless of configuration, the read() function will read the data from these channels as the buffer receives them. This condition is caused by the independence of the channel sample rates. Because each channel is capable of having its own clock rate, channel data is read as a 32-bit block. The first 16 bits are the channel data value, followed by three bits identifying the channel number.

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   6SDISlot = 1;
char  pulBuffer[MAXSAMPLES * 2];


/* Configure driver read() mode */
if( ioctl( FileDesc[6SDISlot], READ_MODE_CONFIG, DMA_MODE) == ERROR )
{
    logMsg("ioctl READ_MODE_CONFIG Failed for Slot #%d\n\n",
```

```
           6SDISlot, 0, 0, 0, 0, 0);
}

/* Configure Input Channel Mode */
if( ioctl( FileDesc[6SDISlot], INPUT_FUNCTION, SINGLE_ENDED) == ERROR )
{
    logMsg("ioctl INPUT_FUNCTION Failed for Slot #%d\n\n",
           6SDISlot, 0, 0, 0, 0, 0);
}

/* Read from the 6SDI device  */
iNumBytesRead = read(FileDesc[6SDISlot],
                     pulBuffer,
                     sizeof(pulBuffer));

if (iNumBytesRead == 0)
{
   logMsg("Read failed for Slot #%d\n", 6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6 ioctl()

The ioctl() function is the standard VxWorks entry point to perform control and setup operations on a 6SDI Card in one PMC Slot. This function should only be called after the open function has been successfully called for a slot where a 6SDI 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 6SDI board.

READ_REGISTER - Reads a specified 6SDI register.

WRITE_REGISTER - Writes to a specified 6SDI register.

START_DMA - Starts a DMA read from the 6SDI 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 6SDI driver.

READ_MODE_CONFIG - Configures the 6SDI read() mode (FIFO scan reads or DMA enabled FIFO reads).

INPUT_FUNCTION - Configures the 6SDI input channels (single-ended or differential mode).

CALIBRATION_MODE – Sets and runs calibration operation.

INTERRUPT_EVENT – Sets interrupt condition.

INPUT_DATA_FORMAT – Sets Binary Offset or Two's Compliment format for data.

BOARD_ROLE – Selects boards mode of operation (target or initiator).

SYNCH_CHANNELS – Synchronizes converters.

ENABLE_PCI_INTERRUPTS – Enables PCI interrupts in order for the 6SDI to produce a local interrupt request.

DISABLE_PCI_INTERRUPTS – Disables PCI Interrupts.

INPUT_TEST – Performs a self-test for 6SDI board validation.

VOLTAGE_RANGE – Sets the voltage range for all channels.

AUTO_PASS – Retrieves the status of autocalibration.

CHECK_CHANNELS – Retrieves the status of channel readiness according to present board activities.

BUFFER_STATUS_FLAG – Retrieves the input buffer status flag information.

ASSIGN_RATES_1 – Assign channel group 0 to a rate generator or external clock.

ASSIGN_RATES_2 – Assign channel group 1 to a rate generator or external clock.

SET_CHAN0_DIVISOR – Sets rate divisor for channel 0.

SET_CHAN1_DIVISOR – Sets rate divisor for channel 1.

SET_CHAN2_DIVISOR – Sets rate divisor for channel 2.

SET_CHAN3_DIVISOR – Sets rate divisor for channel 3.

SET_CHAN4_DIVISOR – Sets rate divisor for channel 4.

SET_CHAN5_DIVISOR – Sets rate divisor for channel 5.

SET_RATE_GEN_A – Sets value to control rate frequency generator A.

SET_RATE_GEN_B – Sets value to control rate frequency generator B.

DISABLE_BUFFER_INPUT – Disables any input to data buffer.

ENABLE_BUFFER_INPUT – Enables any input to data buffer.

CLEAR_BUFFER – Clears input buffer.

SET_THRESHOLD – Sets buffer threshold value.

CLEAR_INT_REQUEST – Clears the 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.

5.6.1 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 6SDI board device has been successfully opened.

arg PARAMETER:

Not used.

EXAMPLE:

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

if (ioctl(FileDesc[6SDISlot], NO_COMMAND, 0) == ERROR)
{
   logMsg("ioctl NO_COMMAND Failed for Slot #%d\n\n", 6SDISlot,
            0, 0, 0, 0, 0);
}
```

5.6.2 INIT_BOARD

The INIT_BOARD function initializes the board and sets all defaults.

arg PARAMETER:

Not used.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;

if (ioctl(FileDesc[6SDISlot], INIT_BOARD, 0) == ERROR)
{
   logMsg("Board Initialization Failed for Slot #%d\n\n", 6SDISlot,
            0, 0, 0, 0, 0);
}
```

5.6.3 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             e6SDIRegister;
    ULONG    *pulValue;
} REG_PARAM;
```

and,

int e6SDIRegister - One of the following registers to read. Refer to the 6SDI hardware documentation for a description of each register.

***** 6SDI Registers *****

BOARD_CTRL_REG

RATE_CTRL_A_REG

RATE_CTRL_B_REG

RATE_ASSIGN_REG

RATE_DIV_01_REG

RATE_DIV_23_REG

RATE_DIV_45_REG

BUFFER_THRESHOLD_REG

DATA_BUFFER_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         6SDISlot = 1;

theReg.pulValue = &ulValue;
theReg.e6SDIRegister = BOARD_CTRL_REG;

if (ioctl(FileDesc[6SDISlot], READ_REGISTER, (int) &theReg) ==
          ERROR)
{

   logMsg("Read Register Failed for Slot #%d\n\n", 6SDISlot,
             0, 0, 0, 0, 0);
}
```

5.6.4 WRITE_REGISTER

The WRITE_REGISTER function writes a value to one of the 6SDI Registers.

arg PARAMETER:

REG_PARAM *

where REG_PARAM is defined to be

```
typedef struct RegParam
{
    int             e6SDIRegister;
    ULONG    *pulValue;
} REG_PARAM;
```

and,

int e6SDIRegister - One of the following registers to write. Refer to the 6SDI Hardware documentation for a description of each register.

***** 6SDI Registers *****

BOARD_CTRL_REG

RATE_CTRL_A_REG

RATE_CTRL_B_REG

RATE_ASSIGN_REG

RATE_DIV_01_REG

RATE_DIV_23_REG

RATE_DIV_45_REG

BUFFER_THRESHOLD_REG

DATA_BUFFER_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   6SDISlot = 1;

theReg.pulValue = &ulValue;
theReg.e6SDIRegister = OUT_Q_PORT;

if (ioctl(FileDesc[6SDISlot], WRITE_REGISTER, (int) &theReg) ==
          ERROR)
{
   logMsg("Write Register Failed for Slot #%d\n\n", 6SDISlot,

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

5.6.5 START_DMA

The START_DMA function configures the 6SDI DMA registers for a DMA transfer 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 6SDI DMA Mode Register.

ULONG ulDMALocalAddress - Value to be written to the 6SDI 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 6SDI DMA Byte Count Register.

ULONG ulDMADescriptorPtr - Value to be written to the 6SDI DMA Descriptor Pointer Register.

ULONG ulDMAArbitration - Value to be written to the 6SDI DMA Arbitration Register.

ULONG ulDMAThreshold - Value to be written to the 6SDI DMA Threshold Register.

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

DMA READ EXAMPLE:

```
#define     DWORD_COUNT 80
int         iIndex, FileDesc[2], 6SDISlot = 1;
DMA_PARAM   DMAParameters;
ULONG       pulBuffer[DWORD_COUNT];
REG_PARAM   theReg;
ULONG       ulValue;

/* Scan input channels.
*/
if(ioctl(FileDesc[6SDISlot], SCAN_INPUTS, 0) == ERROR)
{
   logMsg("Input Scan Failed\n\n", 0, 0, 0, 0, 0, 0);
}

/* Setup parameters to perform a DMA Read from the 6SDI Board. */
DMAParameters.DMAChannel         = 0;
DMAParameters.ulDMAMode          = 0x943;
DMAParameters.ulDMALocalAddress  = (ULONG) pulBuffer;
DMAParameters.ulDMAByteCount     = DWORD_COUNT * 4;
DMAParameters.ulDMADescriptorPtr = 0xA;
DMAParameters.ulDMAArbitration   = 0;
DMAParameters.ulDMAThreshold     = 0;

if (ioctl(FileDesc[6SDISlot], START_DMA, (int) &DMAParameters) ==
ERROR)
{
   logMsg("Start DMA Failed for Slot #%d\n\n", 6SDISlot,
            0, 0, 0, 0, 0);
}

/* Wait for the DMA to Complete. */
theReg.pulValue      = &ulValue;
theReg.e6SDIRegister = DMA_CMD_STATUS;
do
{
   if (ioctl(FileDesc[6SDISlot], READ_REGISTER, (int) &theReg) ==
ERROR)
   {
      logMsg("Read Register Failed for Slot #%d\n\n", 6SDISlot,
               0, 0, 0, 0, 0);
      break;
```

```
   }
} while (! (ulValue & 0x10));

/* Clear the DMA channel 0/1 command/status register.
*/
ulValue = 0;
theReg.pulValue      = &ulValue;
theReg.e6SDIRegister = DMA_CMD_STATUS;

if (ioctl(FileDesc[6SDISlot], WRITE_REGISTER, (int) &theReg) ==
          ERROR)
{
          logMsg("Write Register Failed\n\n",
                 0, 0, 0, 0, 0, 0);
}
```

5.6.6 REG_FOR_INT_NOTIFY

The REG_FOR_INT_NOTIFY function will register or unregister for notification that an interrupt has occurred on the 6SDI board. If this function is called with a pointer to a subroutine, that routine will be invoked when an 6SDI 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 6SDI Board that has interrupted and will be one of the following:

- 6SDI_PMC1
- 6SDI_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 6SDISlot = 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[6SDISlot], REG_FOR_INT_NOTIFY, (int) intHandler)
          == ERROR)
{
```

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

5.6.7 GET_DEVICE_ERROR

The GET_DEVICE_ERROR function will return the error that occurred on the last call to one of the 6SDI 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 6SDIDrvInstall function fails when probing the 6SDI card's Board Status Register.

DEVICE_ADD_ERROR - Failure occurred when the 6SDIDrvInstall 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 6SDIDrvInstall function if an invalid driver number was obtained when trying to add the device driver to the VxWorks operating system. Also returned from the 6SDIDrvRemove function if the driver failed to remove the device driver from the VxWorks operating system.

ALREADY_INSTALLED_ERR - Returned from the 6SDIDrvInstall function if the driver has already been installed.

PCI_CONFIG_ERR - Returned from the 6SDIDrvInstall function if a read or write of a PCI Configuration Register fails.

INVALID_BOARD_STATUS_ERR - Returned from the 6SDIDrvInstall function if an invalid board status is read from the 6SDI Board.

FIFO_BUFFER_ERR - If during a read() transaction the FIFO buffer is indicated to be empty by the status of the buffer status flags or more data is requested than what is available, the driver will return the number of bytes that could be written along with throwing this error condition.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int Status;

/* Send the Get Device Error Code Command for this channel   */
if (ioctl(FileDesc[6SDISlot], GET_DEVICE_ERROR, (int) &Status) ==
ERROR)
{
   logMsg("Get Device Error Code Failed for Slot #%d\n\n",
          6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.8 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 where the DMA capability of the board is taken advantage.

arg PARAMETER:

int * - Pointer to one of the following values:

- SCAN_MODE
- DMA_MODE

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int iMode;

iMode = DMA_MODE;

if (ioctl(FileDesc[6SDISlot], READ_MODE_CONFIG, (int) &iMode) ==
ERROR)
{
   logMsg("Read Mode Configuration Failed for Slot #%d\n\n",
   6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.9 INPUT_FUNCTION

The INPUT_FUNCTION 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 6SDISlot = 1;
int iMode;

iMode = DIFFERENTIAL;

if (ioctl(FileDesc[6SDISlot], INPUT_FUNCTION, (int) &iMode) ==
ERROR)
{
   logMsg("Input Configuration Failed for Slot #%d\n\n", 6SDISlot,
   0, 0, 0, 0, 0);
}
```

5.6.10 CALIBRATION_MODE

The CALIBRATION_MODE function performs the calibration operation. Refer to the PMC-6SDI User's Manual for more information on this operation.

arg PARAMETER:

Not used.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;

if (ioctl(FileDesc[6SDISlot], CALIBRATION_MODE, 0) ==
   ERROR)
{
   logMsg("Calibration Failed for Slot #%d\n\n", 6SDISlot,
            0, 0, 0, 0, 0);
}

LOOP_TEST_CHANNEL
```

5.6.11 INTERRUPT_EVENT

The INTERRUPT_EVENT function will set the interrupt condition for a single local interrupt request.

arg PARAMETER:

int * - Pointer to one of the following values:

- INIT_COMPLETE
- AUTOCAL_COMPLETE
- CHANNELS_READY
- FLAG_TO_HIGH (Buffer threshold flag has had a LOW_TO_HIGH transition)
- FLAG_TO_LOW (Buffer threshold flag has had a HIGH_TO_LOW transition)
- BUFFER_ALMOST_FULL
- BUFFER_ALMOST_EMPTY

EXAMPLE:

```
int   FileDesc[2];
int   6SDISlot = 1;
int   Source;

Source = CHANNELS_READY;

if (ioctl(FileDesc[6SDISlot], INTERRUPT_EVENT, (int) &Source) ==
ERROR)
{
   logMsg("Interrupt Selection Failed for Slot #%d\n\n", 6SDISlot,
            0, 0, 0, 0, 0);
}
```

5.6.12 INPUT_DATA_FORMAT

The INPUT_DATA_FORMAT function sets the convention for the data to either binary offset or two's complement.

arg PARAMETER:

int * - Pointer to one of the following values:

- BINARY_OFFSET
- TWOS_COMP

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;


if (ioctl(FileDesc[6SDISlot], INPUT_DATA_FORMAT, BINARY_OFFSET) ==
          ERROR)
{
   logMsg("Select Data Format Failed for Slot #%d\n\n",
         6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.13 BOARD_ROLE

The BOARD_ROLE function will set the mode of operation for a particular board in a multiple board setup. A board is either the initiator or the target. The initiator of the sampling clock and synchronization command. All other boards are designated as targets.

arg PARAMETER:

int * - Pointer to one of the following values:

- INITIATOR
- TARGET

EXAMPLE:

```
int   FileDesc[2];
int   6SDISlot = 1;
int   Role;

Source = INITIATOR;

if (ioctl(FileDesc[6SDISlot], BOARD_ROLE, (int) &Role) == ERROR)
{
   logMsg("Select Board Role Failed for Slot #%d\n\n", 6SDISlot,
            0, 0, 0, 0, 0);
}
```

5.6.14 SYNCH_CHANNELS

The SYNCH_CHANNELS function synchronizes the channel converters. This makes it possible to have simultaneous data conversions to a common clock rate.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int   FileDesc[2];
int   6SDISlot = 1;

if (ioctl(FileDesc[6SDISlot], SYNCH_CHANNELS, 0) == ERROR)
{
   logMsg("Synchronize Channels Failed for Slot #%d\n\n",
            6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.15 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   6SDISlot = 1;

if (ioctl(FileDesc[6SDISlot], ENABLE_PCI_INTERRUPTS, 0) == ERROR)
{
   logMsg("PCI Interrupt Enable Failed for Slot #%d\n\n",
            6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.16 DISABLE_PCI_INTERRUPTS

The DISABLE_PCI_INTERRUPTS function disables the PCI interrupts.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int   FileDesc[2];
int   6SDISlot = 1;

if (ioctl(FileDesc[6SDISlot], DISABLE_PCI_INTERRUPTS, 0) == ERROR)
{
   logMsg("PCI Interrupts Disable Failed for Slot #%d\n\n",
           6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.17 INPUT_TEST

The INPUT_TEST function will perform a system level validation of operation precision. There are two tests, positive reference test and zero input test. During the positive reference test, an internal voltage reference is connected to any or all input channels. The zero input test consists of any or all input channels being connected to the internal ground.

arg PARAMETER:

int * - Pointer to one of the following values:

- · POSITIVE_REF
- · ZERO

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int Test;

Test = ZERO;

if (ioctl(FileDesc[6SDISlot], INPUT_TEST, (int) &Test) == ERROR)
{
   logMsg("Input Test Failed for Slot #%d\n\n", 6SDISlot,
0, 0, 0, 0, 0);
}
```

5.6.18 VOLTAGE_RANGE

The VOLTAGE_RANGE function selects the voltage range at which the channels are to operate. There are four ranges in the selection.

arg PARAMETER:

int * - Pointer to one of the following values:

- PLUS_MIN_1_25
- PLUS_MIN_2_5
- PLUS_MIN_5
- PLUS_MIN_10

EXAMPLE

```
int FileDesc[2];
int 6SDISlot = 1;
int Range;

Test = ZERO;

if (ioctl(FileDesc[6SDISlot], VOLTAGE_RANGE, (int) &Range) == ERROR)
{
   logMsg("Voltage Range Selection Failed for Slot #%d\n\n", 6SDISlot,
0, 0, 0, 0, 0);
}
```

5.6.19 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:

- **AUTOCAL_FAILED**
- **AUTOCAL_PASSED**

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int Status;

if (ioctl(FileDesc[6SDISlot], AUTO_PASS, (int) &Status) == ERROR)
{
   logMsg("Get Autocalibration Status Failed for Slot #%d\n\n",
    6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.20 CHECK_CHANNELS

The CHECK_CHANNELS function gets the status of the channels. Channels will either be ready or not ready. The status depends on whether or not a procedure is complete and/or successful.

arg PARAMETER:

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

- **CHANS_READY**
- **CHANS_NOT_READY**

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int Status;

if (ioctl(FileDesc[6SDISlot], CHECK_CHANNEL, (int) &Status) == ERROR)
{
   logMsg("Get Channel Status Failed for Slot #%d\n\n",
    6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.21 BUFFER_STATUS_FLAG

The BUFFER_STATUS_FLAG function gets the status of the buffer space. The flag is set when the buffer threshold value is exceeded. The threshold value is discussed in a later section.

arg PARAMETER:

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

- **BUFFER_FULL**
- **BUFFER_EMPTY**

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int Status;

if (ioctl(FileDesc[6SDISlot], BUFFER_STATUS_FLAG, (int) &Status) ==
ERROR)
{
   logMsg("Get Buffer Status Failed for Slot #%d\n\n",
    6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.22 ASSIGN_RATES_1

The ASSIGN_RATES_1 function assigns a rate generator or external clock to channel group 0. This function can also disable data input from channels in group 0. Group 0 is defined as channels 0, 1, and 2.

arg PARAMETER:

int * - Pointer to one of the following:

- GENERATOR_A
- GENERATOR_B
- EXTERNAL_CLOCK
- DISABLED

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int Rate;

if (ioctl(FileDesc[6SDISlot], ASSIGN_RATES_1, (int) &Rate) == ERROR)
{
   logMsg("Group0 Rate Assignment Failed for Slot #%d\n\n",
    6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.23 ASSIGN_RATES_2

The ASSIGN_RATES_2 function assigns a rate generator or external clock to channel group 1. This function can also disable data input from channels in group 1. Group 1 is defined as channels 3, 4, and 5.

arg PARAMETER:

int * - Pointer to one of the following:

- GENERATOR_A
- GENERATOR_B
- EXTERNAL_CLOCK
- DISABLED

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int Rate;

if (ioctl(FileDesc[6SDISlot], ASSIGN_RATES_2, (int) &Rate) == ERROR)
{
   logMsg("Group1 Rate Assignment Failed for Slot #%d\n\n",
    6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.24 SET_CHAN0_DIVISOR

The SET_CHAN0_DIVISOR function sets the rate divisor for channel 0. The rate generator frequency is calculated by using the rate divisor, iNdiv, and the sampling frequency for the channel as:

$$Fgen\ (kHz) = 64 * Fsamp * iNdiv,$$

where Fsamp is in kilohertz. The valid range for the divisor is from 1 to 32.

arg PARAMETER:

int * - Pointer to the integer used in calculation.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int iNdiv;

iNdiv = 0x0020;

/* Set Rate Divisor for Channel 0. */
if (ioctl(FileDesc[6SDISlot], SET_CHAN0_DIVISOR, iNdiv) == ERROR)
{
   logMsg("Set Rate Divisor for Channel 0 Failed\n\n", 0, 0, 0,
          0, 0, 0 );
}
```

5.6.25 SET_CHAN1_DIVISOR

The SET_CHAN1_DIVISOR function sets the rate divisor for channel 1. The rate generator frequency is calculated by using the rate divisor, iNdiv, and the sampling frequency for the channel as:

$$Fgen\ (kHz) = 64 * Fsamp * iNdiv,$$

where Fsamp in kilohertz. The valid range for the divisor is from 1 to 32.

arg PARAMETER:

int * - Pointer to the integer used in calculation.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int iNdiv;

iNdiv = 0x0020;

/* Set Rate Divisor for Channel 1. */
if (ioctl(FileDesc[6SDISlot], SET_CHAN1_DIVISOR, iNdiv) == ERROR)
{
   logMsg("Set Rate Divisor for Channel 1 Failed\n\n", 0, 0, 0,
          0, 0, 0 );
}
```

5.6.26 SET_CHAN2_DIVISOR

The SET_CHAN2_DIVISOR function sets the rate divisor for channel 2. The rate generator frequency is calculated by using the rate divisor, iNdiv, and the sampling frequency for the channel as:

$$Fgen\ (kHz) = 64 * Fsamp * iNdiv,$$

where Fsamp is in kilohertz. The valid range for the divisor is from 1 to 32.

arg PARAMETER:

int * - Pointer to the integer used in calculation.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int iNdiv;

iNdiv = 0x0020;

/* Set Rate Divisor for Channel 2. */
if (ioctl(FileDesc[6SDISlot], SET_CHAN2_DIVISOR, iNdiv) == ERROR)
{
   logMsg("Set Rate Divisor for Channel 2 Failed\n\n", 0, 0, 0,
          0, 0, 0 );
}
```

5.6.27 SET_CHAN3_DIVISOR

The SET_CHAN3_DIVISOR function sets the rate divisor for channel 3. The rate generator frequency is calculated by using the rate divisor, iNdiv, and the sampling frequency for the channel as:

$$Fgen\ (kHz) = 64 * Fsamp * iNdiv,$$

where Fsamp is in kilohertz. The valid range for the divisor is from 1 to 32.

arg PARAMETER:

int * - Pointer to the integer used in calculation.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int iNdiv;

iNdiv = 0x0020;

/* Set Rate Divisor for Channel 3. */
if (ioctl(FileDesc[6SDISlot], SET_CHAN3_DIVISOR, iNdiv) == ERROR)
{
   logMsg("Set Rate Divisor for Channel 3 Failed\n\n", 0, 0, 0,
          0, 0, 0 );
}
```

5.6.28 SET_CHAN4_DIVISOR

The SET_CHAN4_DIVISOR function sets the rate divisor for channel 4. The rate generator frequency is calculated by using the rate divisor, iNdiv, and the sampling frequency for the channel as:

$$Fgen\ (kHz) = 64 * Fsamp * iNdiv,$$

where Fsamp in kilohertz. The valid range for the divisor is from 1 to 32.

arg PARAMETER:

int * - Pointer to the integer used in calculation.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int iNdiv;

iNdiv = 0x0020;

/* Set Rate Divisor for Channel 4. */
if (ioctl(FileDesc[6SDISlot], SET_CHAN4_DIVISOR, iNdiv) == ERROR)
{
   logMsg("Set Rate Divisor for Channel 4 Failed\n\n", 0, 0, 0,
          0, 0, 0 );
}
```

5.6.29 SET_CHAN5_DIVISOR

The SET_CHAN5_DIVISOR function sets the rate divisor for channel 5. The rate generator frequency is calculated by using the rate divisor, iNdiv, and the sampling frequency for the channel as:

$$Fgen\ (kHz) = 64 * Fsamp * iNdiv,$$

where Fsamp is in kilohertz. The valid range for the divisor is from 1 to 32.

arg PARAMETER:

int * - Pointer to the integer used in calculation.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int iNdiv;

iNdiv = 0x0020;

/* Set Rate Divisor for Channel 5. */
if (ioctl(FileDesc[6SDISlot], SET_CHAN5_DIVISOR, iNdiv) == ERROR)
{
   logMsg("Set Rate Divisor for Channel 5 Failed\n\n", 0, 0, 0,
          0, 0, 0 );
}
```

5.6.30 SET_RATE_GEN_A

The SET_RATE_GEN_A function sets the rate control value, iNrate, associated with a particular group of channels. The rate generator frequency is calculated as:

$$iNrate = (0.063875 * Fgen) - 511$$

The valid range for the rate control value is from 0 to 511. This value must use the highest sample rate in the channel group.

arg PARAMETER:

int * - Pointer to the integer used in calculation.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int iNrate;

iNrate = 0x0100;

/* Program Rate Generator A. */
if (ioctl(FileDesc[6SDISlot], SET_RATE_GEN_A, iNrate) == ERROR)
{
   logMsg("Set Rate Generator A Failed\n\n", 0, 0, 0,
          0, 0, 0 );
}
```

5.6.31 SET_RATE_GEN_B

The SET_RATE_GEN_B function sets the rate control value, iNrate, associated with a particular group of channels. The rate generator frequency is calculated as:

$$iNrate = (0.063875 * Fgen) - 511$$

The valid range for the rate control value is from 0 to 511. This value must use the highest sample rate in the channel group.

arg PARAMETER:

int * - Pointer to the integer used in calculation.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;
int iNrate;

iNrate = 0x0100;

/* Program Rate Generator B. */
if (ioctl(FileDesc[6SDISlot], SET_RATE_GEN_B, iNrate) == ERROR)
{
   logMsg("Set Rate Generator B Failed\n\n", 0, 0, 0,
          0, 0, 0 );
}
```

5.6.32 DISABLE_BUFFER_INPUT

The DISABLE_BUFFER_INPUT function disables input to the buffer from the channels.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;

if (ioctl(FileDesc[6SDISlot], DISABLE_BUFFER_INPUT, 0) == ERROR)
{
   logMsg("Disable Buffer Input Failed for Slot #%d\n\n", 6SDISlot,
0, 0, 0, 0, 0);
}
```

5.6.33 ENABLE_BUFFER_INPUT

The ENABLE_BUFFER_INPUT function enables input to the buffer from the channels.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;

if (ioctl(FileDesc[6SDISlot], ENABLE_BUFFER_INPUT, 0) == ERROR)
{
   logMsg("Enable Buffer Input Failed for Slot #%d\n\n", 6SDISlot,
0, 0, 0, 0, 0);
}
```

5.6.34 CLEAR_BUFFER

The CLEAR_BUFFER function clears the buffer, although it does not stop input from channels.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;


if (ioctl(FileDesc[6SDISlot], CLEAR_BUFFER, 0) == ERROR)
{
    logMsg("Clear Buffer Failed for Slot #%d\n\n", 6SDISlot, 0, 0,
           0, 0, 0);
}
```

5.6.35 SET_THRESHOLD

The SET_THRESHOLD function sets the buffer threshold value. This value may be used, along with the buffer threshold flag, to observe the amount of data in the buffer. Once the threshold value is set, the user may watch the flag for transition. If the flag is set, the threshold value has been exceeded.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;


if (ioctl(FileDesc[6SDISlot], SET_THRESHOLD, 0) == ERROR)
{
   logMsg("Set Threshold Value Failed for Slot #%d\n\n",
            6SDISlot, 0, 0, 0, 0, 0);
}
```

5.6.36 CLEAR_INT_REQUEST

The CLEAR_INT_REQUEST function clears the interrupt request flag after an interrupt has occurred.

arg PARAMETER:

Not Used.

EXAMPLE:

```
int FileDesc[2];
int 6SDISlot = 1;


if (ioctl(FileDesc[6SDISlot], CLEAR_INT_REQUEST, 0) == ERROR)
{
   logMsg("Clear Interrupt Request Flag Failed for Slot #%d\n\n",
         6SDISlot, 0, 0, 0, 0, 0);
}
```