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-FLASH2 hosted on PowerPC and 80x86 Processors

Document number:	9005007	Revision:	1.0	Date: 10/20//99
Engineering Approval:				Date:
Quality Representative Approval:				Date:

Acknowledgments

1 Driver Overview

The purpose of this document is to describe how to interface with the PMC-FLASH2 VxWorks Device Driver developed by Chandler/May, Incorporated (CMI). This software provides the interface between "Application Software" and the FLASH2 Board. The interface to this board is at the I/O system level. It requires no knowledge of the actual board addressing of the PLX PCI bus master interface.

The FLASH2 Driver Software executes under control of the VxWorks operating system. The FLASH2 is implemented as a standard VxWorks device driver written in the 'C' programming language. The FLASH2 Driver Software is designed to operate on CPU boards containing MPC603, MPC604, and MPC750 processors as well as VME and CompactPCI 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, and the SCI JTT 686 CPU board.

The General Standards Corporation (GSC) FLASH2 board is a static RAM and FLASH memory device that fits into a PCI Mezzanine Card slot. The primary usage for this SRAM/FLASH memory card is for non-volatile storage. The FLASH2 board can be delivered in several different configurations. Some boards may have SRAM only. Other boards may have 512Kbytes of SRAM and up to 127.5Mbytes of FLASH memory. Currently, this driver only supports the SRAM portion of FLASH2 boards.

2 Referenced Documents

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

- PMC-FLASH2 User's Manual – Revision A, 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.

3 Making the Device Driver

In order to use the FLASH2 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. The user is also required to modify the #define **GS_FLASH2_SRAM_SIZE** located in the file flash2_drv.h. This definition indicates to the driver the size of the SRAM in bytes on the FLASH2 card. Once **BSP** and **GS_FLASH2_SRAM_SIZE** are set correctly, the user can then begin compiling by executing **make**.

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

The output file from the build procedure should be:

flash2_drv.o	Relocatable/loadable module for the device driver.

4 Virtual-to-Physical Memory Mapping

If your BSP does not already have the PCI memory space addresses required by the PMC-FLASH2 card mapped by the **PHYS_MEM_DESC** structure then it will be necessary for the user to add this region of memory required into **sysPhysMemDesc** and then rebuild the kernel. This is done by editing sysLib.c and adding the following lines of code to the **sysPhysMemDesc** table.

```
/* GS PMC-FLASH2 SRAM – PMC Site 1 */
{
 (void *) GS_FLASH2_PMC_1_LOCAL_ADRS,
 (void *) GS_FLASH2_PMC_1_LOCAL_ADRS,
 GS_FLASH2_SRAM_SIZE,
 VM_STATE_MASK_VALID | VM_STATE_MASK_WRITABLE |
 VM_STATE_MASK_CACHEABLE,
 VM_STATE_VALID | VM_STATE_WRITABLE | VM_STATE_CACHEABLE
},

/* This map is only needed on CPU boards
 * with 2 PMC sites
 */
/* GS PMC-FLASH2 SRAM – PMC Site 2 */
{
 (void *) GS_FLASH2_PMC_2_LOCAL_ADRS,
 (void *) GS_FLASH2_PMC_2_LOCAL_ADRS,
 GS_FLASH2_SRAM_SIZE,
 VM_STATE_MASK_VALID | VM_STATE_MASK_WRITABLE |
 VM_STATE_MASK_CACHEABLE,
 VM_STATE_VALID | VM_STATE_WRITABLE | VM_STATE_CACHEABLE
},
```

where **GS_FLASH2_PMC_1_LOCAL_ADRS** is defined to be

CPU_PCI_ISA_MEM_ADRS + **GS_FLASH2_PMC1_PCI_ADRS**,

and where **GS_FLASH2_PMC_2_LOCAL_ADRS** is defined to be

CPU_PCI_ISA_MEM_ADRS + **GS_FLASH2_PMC2_PCI_ADRS**.

GS_FLASH2_PMC1_PCI_ADRS, **GS_FLASH2_PMC2_PCI_ADRS** and **GS_FLASH2_SRAM_SIZE** can be found in the file flash2_drv.h. **GS_FLASH2_PMC1_PCI_ADRS** and **GS_FLASH2_PMC2_PCI_ADRS** are user selectable offsets into PCI memory space while **GS_FLASH2_SRAM_SIZE** is based on the physical size of SRAM on the FLASH2 card. **CPU_PCI_ISA_MEM_ADRS**, which is the base address for PCI memory space on the CPU board being used, is BSP dependant. The following table describes where this #define can be found, the name used, and its value for several BSP's.

BSP	File Name	#define	Value
mv1603	mv1600.h	CPU_PCI_ISA_MEM_ADRS	0xc000 0000
mv1604	mv1600.h	CPU_PCI_ISA_MEM_ADRS	0xc000 0000
mv2302	mv2600.h	CPU_PCI_ISA_MEM_ADRS	Pseudo PREP: 0xc000 0000 Extended VME: 0xfd00 0000
mv2304	mv2600.h	CPU_PCI_ISA_MEM_ADRS	Pseudo PREP: 0xc000 0000 Extended VME: 0xfd00 0000
mv2400	mv2400.h	PCI_MSTR_MEMIO_LOCAL	Pseudo PREP: 0xc000 0000 Extended VME: 0xfd00 0000
mv2603	mv2600.h	CPU_PCI_ISA_MEM_ADRS	Pseudo PREP: 0xc000 0000 Extended VME: 0xfd00 0000
mv2604	mv2600.h	CPU_PCI_ISA_MEM_ADRS	Pseudo PREP: 0xc000 0000 Extended VME: 0xfd00 0000
mv2700	mv2600.h	CPU_PCI_ISA_MEM_ADRS	Pseudo PREP: 0xc000 0000 Extended VME:

			0xfd00 0000
pcore604	pcore60x.h	Not Defined	0xb000 0000
jtt686	pci_vme.h	CPU_PCI_ISA_MEM_ADRS	Pseudo PREP: 0xc000 0000 Extended VME: 0xf103 0000

5 Driver Interface

The FLASH2 Driver conforms to the device driver standards required by the VxWorks Operating System and contains the following standard driver entry points.

- FLASH2DrvInstall() – Installs the device driver for use with multiple FLASH2 Cards
- FLASH2DrvRemove() – Removes the device driver from use
- open() – opens a driver interface to one FLASH2 Card
- close() – closes a driver interface to one FLASH2 Card
- read() – currently has no functionality for this particular device
- write() – currently has no functionality for this particular device
- ioctl() – performs various control and setup functions on the FLASH2 Card

The FLASH2 Device Driver provides a standard I/O system interface to the GSC PMC-FLASH2 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 PLX registers. It is not necessary, however, to go through the VxWorks I/O system to access SRAM. The user can simply access SRAM directly in memory since the driver installation has made the FLASH2 board accessible in PCI memory space.

The FLASH2 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, ...).

It is important to note that the FLASH2 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.

5.1 FLASH2DrvInstall()

The FLASH2DrvInstall () 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 FLASH2DrvRemove() function.

Once this installation has been performed successfully the user has access to the FLASH2 device in memory. He does not have to open the device unless he wishes to perform any ioctl() calls such as accessing PLX registers, reading device error codes, or getting the board installation address. Keep in mind that the board installation address can be determined from the driver header files. Refer to the section on virtual-to-physical memory mapping to determine this address.

The FLASH2DrvInstall () 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 FLASH2 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 FLASH2 Card to respond over the PCI Bus
    - Installs a device for the PMC Slot

PROTOTYPE:

extern int FLASH2DrvInstall(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 FLASH2 VxWorks Device Driver. */
iStatus = FLASH2DrvInstall(TRUE);
```

5.2 FLASH2DrvRemove()

The FLASH2DrvRemove() function is used to remove the FLASH2 Device Driver from the VxWorks operating system. This function should only be called after a call to the FLASH2DrvInstall() function. The FLASH2DrvRemove() function closes all the open devices for each PMC slot and removes the device driver from the operating system.

PROTOTYPE:

extern int FLASH2DrvRemove(void);

Returns OK on success and ERROR on failure

EXAMPLE:

```
STATUS iStatus;

/* Remove the FLASH2 Driver */
iStatus = FLASH2DrvRemove();
```

5.3 open()

The open() function is the standard VxWorks entry point to open a connection to a FLASH2 Card in one PMC Slot. This function may only be called after a call to the FLASH2DrvInstall() function is made.

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 FLASH2 Board is in:

- FLASH2_PMC1 - PMC slot 1
- FLASH2_PMC2 - PMC slot 2

iFlags - is not used.

iMode - is not used.

Returns OK on success and ERROR on failure

EXAMPLE:

```
int         FileDesc[2];
LOCAL char  *devName[] = {FLASH2_PMC1, FLASH2_PMC2};
int         FLASH2Slot = 1;


/*   open the FLASH2 device for slot 1   */
FileDesc[FLASH2Slot] = open(devName[FLASH2Slot], O_RDWR, 0644);

if (FileDesc[FLASH2Slot] == ERROR)
{
   logMsg("Cannot Open Device Error %s\n\n",
            (int) devName[FLASH2Slot], 0, 0, 0, 0, 0);
}
```

5.4 close()

The close() function is the standard VxWorks entry point to close a connection to a FLASH2 Card in one PMC Slot. This function should only be called after the open function has been successfully called for a slot where an FLASH2 Card resides. The close function closes the interface to an FLASH2 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 FLASH2Slot = 1;

/*  close the device on slot 2  */
if  (close(FileDesc[FLASH2Slot]) == ERROR)
{
   logMsg("Close Error for Slot #%d\n\n", FLASH2Slot, 0, 0, 0, 0, 0);
}
FileDesc[FLASH2Slot] = ERROR;
```

5.5 read()

The read() function is the standard VxWorks entry point to receive data from an opened device. However, since the FLASH2 card is simply mapped into PCI memory space as a memory device, the user can access it directly instead of calling this function.

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 nothing since the driver does not implement this function.

5.6 write()

The write() function is the standard VxWorks entry point to write data to an opened device. However, since the FLASH2 card is simply mapped into PCI memory space as a memory device, the user can access it directly instead of calling this function.

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 nothing since the driver does not implement this function.

5.7 ioctl()

The ioctl() function is the standard VxWorks entry point to perform control and setup operations on an FLASH2 Card in one PMC Slot. This function should only be called after the open function has been successfully called for a slot where an FLASH2 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.

READ_REGISTER - Reads a specified FLASH2 register.

WRITE_REGISTER - Writes to a specified FLASH2 register.

GET_DEVICE_ERROR - Returns the error that occurred during the last access to the FLASH2 driver.

GET_BASE_ADDRESS – Retrieves the base address of the installed FLASH2 card.

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 FLASH2 board device has been successfully opened.

arg PARAMETER:

Not used.

EXAMPLE:

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

if (ioctl(FileDesc[FLASH2Slot], NO_COMMAND, 0) == ERROR)
{
   logMsg("ioctl NO_COMMAND Failed for Slot #%d\n\n", FLASH2Slot,
            0, 0, 0, 0, 0);
}
```

5.7.2 READ_REGISTER

The READ_REGISTER function reads and returns the contents of one of the FLASH2 registers.

arg PARAMETER:

REG_PARAM *

where REG_PARAM is defined to be

```
typedef struct RegParam
{
    int             eFLASH2Register;
    ULONG    *pulValue;
} REG_PARAM;
```

and,

int eFLASH2Register - One of the following registers to read. Refer to the FLASH2 and PLX hardware documentation for a description of each register.

***** 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         FLASH2Slot = 1;

theReg.pulValue = &ulValue;
theReg.eFLASH2Register = PCI_TO_LOC_ADDR_0_RNG;

if (ioctl(FileDesc[FLASH2Slot], READ_REGISTER, (int) &theReg) ==
         ERROR)
{

   logMsg("Read Register Failed for Slot #%d\n\n", FLASH2Slot,
            0, 0, 0, 0, 0);
}
```

5.7.3 WRITE_REGISTER

The WRITE_REGISTER function writes a value to one of the FLASH2 Registers.

arg PARAMETER:

REG_PARAM *

where REG_PARAM is defined to be

```
typedef struct RegParam
{
    int             eFLASH2Register;
    ULONG    *pulValue;
} REG_PARAM;
```

and,

int eFLASH2Register - One of the following registers to write. Refer to the FLASH2 and PLX Hardware documentation for a description of each register.

***** 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 = 0xFC000000;
int   FLASH2Slot = 1;

theReg.pulValue = &ulValue;
theReg.eFLASH2Register = PCI_TO_LOC_ADDR_0_RNG;

if (ioctl(FileDesc[FLASH2Slot], WRITE_REGISTER, (int) &theReg) ==
         ERROR)
{
   logMsg("Write Register Failed for Slot #%d\n\n", FLASH2Slot,

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

5.7.4 GET_DEVICE_ERROR

The GET_DEVICE_ERROR function will return the error that occurred on the last call to one of the FLASH2 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.

DEVICE_ADD_ERROR - Failure occurred when the FLASH2DrvInstall 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 FLASH2DrvInstall function if an invalid driver number was obtained when trying to add the device driver to the VxWorks operating system. Also returned from the FLASH2DrvRemove function if the driver failed to remove the device driver from the VxWorks operating system.

ALREADY_INSTALLED_ERR - Returned from the FLASH2DrvInstall function if the driver has already been installed.

PCI_CONFIG_ERR - Returned from the FLASH2DrvInstall function if a read or write of a PCI Configuration Register fails.

EXAMPLE:

```
int FileDesc[2];
int FLASH2Slot = 1;
int Status;

/* Send the Get Device Error Code Command for this channel   */
if (ioctl(FileDesc[FLASH2Slot], GET_DEVICE_ERROR, (int) &Status) ==
ERROR)
{
   logMsg("Get Device Error Code Failed for Slot #%d\n\n",
          FLASH2Slot, 0, 0, 0, 0, 0);
}
```

5.7.5 GET_BASE_ADDRESS

The GET_BASE_ADDRESS function will return to the user the address at which the board was installed using the FLASHDrvInstall() routine.

arg PARAMETER:

unsigned long * - Pointer to the location where the base address is to be written.

EXAMPLE:

```
int FileDesc[2];
int FLASH2Slot = 1;
ULONG ulBaseAddress;


if (ioctl(FileDesc[FLASH2Slot], GET_BASE_ADDRESS, (int)
&ulBaseAddress) == ERROR)
{
   logMsg("Get Base Address Failed for Slot #%d\n\n", FLASH2Slot, 0,
   0, 0, 0, 0);
}
```