Software Interface
CCURPMFC (WC-CP-FIO)

PCIe Programmable Multi-Function I/O Card (MIOC)

Driver	ccurpmfc (WC-CP-FIO)	
OS	RedHawk (CentOS/Rocky or Ubuntu based)	
Vendor	Concurrent Real-Time	
Hardware	PCIe Programmable Multi-Function Card (CP-FPGA-Ax)	
Author	Darius Dubash	
Date	February 5th, 2024	Rev 2024.1

This page intentionally left blank

Table of Contents

This page intentionally left blank

1. Introduction

This document provides the software interface to the ***ccurpmfc*** driver which communicates with the Concurrent Real-Time PCI Express Programmable Multi-Function FPGA I/O Card (MIOC). Low-level programming information is contained in *Concurrent Real-Time PCIe Programmable Multi-Function I/O Cards (MIOC) Design Specification* (No. *0610104*) which is a CCRT internal document that is not supplied to the customer.

The software package that accompanies this board provides the ability for advanced users to communicate directly with the board via the driver *ioctl(2)* and *mmap(2)* system calls. When programming in this mode, the user needs to be intimately familiar with both the hardware and the register programming interface to the board. Failure to adhere to correct programming will result in unpredictable behavior.

Additionally, the software package is accompanied with an extensive set of application programming interface (API) calls that allow the user to access all capabilities of the board. The API library also allows the user the ability to communicate directly with the board through the *ioctl(2)* and *mmap(2)* system calls. In this case, there is a risk of this direct access conflicting with API calls and therefore should only be used by advanced users who are intimately familiar with the hardware, board registers and the driver code.

Various example tests have been provided in the *test* and *test/lib* directories to assist the user in developing their applications.

1.1 Related Documents

- PCIe Programmable Multi-Function Driver Installation on RedHawk Release Notes by Concurrent Real-Time.
- PCIe Programmable Multi-Function Driver Technical Guide by Concurrent Real-Time.
- PCIe Programmable Multi-Function Card I/O (MIOC) Design Specification (No. *0610104*) by Concurrent Real-Time *(internal document)*.

2. Software Support

Software support is provided for users to communicate directly with the board using the kernel system calls *(Direct Driver Access)* or the supplied *API*. Both approaches are identified below to assist the user in software development.

2.1 Direct Driver Access

2.1.1 open(2) system call

In order to access the board, the user first needs to open the device using the standard system call *open(2).*

```
int    fp;
fp = open("/dev/ccurpmfc0", O_RDWR);
```

The file pointer '*fp'* is then used as an argument to other system calls. The user can also supply the O_NONBLOCK flag if the user does not wish to block waiting for reads to complete. In that case, if the read is not satisfied, the call will fail. The device name specified is of the format "/dev/ccurpmfc<num>" where *num* is a digit 0..9 which represents the board number that is to be accessed. Basically, the driver only allows one application to open a board at a time. The reason for this is that the application can have full access to the card, even at the board and API level. If another application were to communicate with the same card concurrently, the results would be unpredictable unless proper synchronization between applications is performed external to the driver API.

This driver allows multiple applications to open the same board by specifying an additional *oflag O_APPEND*. It is then the responsibility of the user to ensure that the various applications communicating with the same cards are properly synchronized. Various tests supplied in this package has the *O_APPEND* flags enabled, however, it is strongly recommended that only one application be run with a single card at a time, unless the user is well aware of how the applications are going to interact with each other and accept any unpredictable results.

2.1.2 ioctl(2) system call

This system call provides the ability to control and get responses from the board. The nature of the control/response will depend on the specific *ioctl* command.

```
int   status;
int   arg;
status = ioctl(fp, <IOCTL_COMMAND>, &arg);
```

where, '*fp'* is the file pointer that is returned from the *open(2)* system call. *<IOCTL_COMMAND>* is one of the *ioctl* commands below and *arg* is a pointer to an argument that could be anything and is dependent on the command being invoked. If no argument is required for a specific command, then set to *NULL*.

Driver IOCTL command:

```
IOCTL_CCURPMFC_ABORT_DMA
IOCTL_CCURPMFC_ADD_IRQ
IOCTL_CCURPMFC_DISABLE_PCI_INTERRUPTS
IOCTL_CCURPMFC_ENABLE_PCI_INTERRUPTS
IOCTL_CCURPMFC_GET_DRIVER_ERROR
IOCTL_CCURPMFC_GET_DRIVER_INFO
IOCTL_CCURPMFC_GET_PHYSICAL_MEMORY
IOCTL_CCURPMFC_GET_ADC_READ_MODE
IOCTL_CCURPMFC_GET_DAC_WRITE_MODE
IOCTL_CCURPMFC_INIT_BOARD
IOCTL_CCURPMFC_INTERRUPT_TIMEOUT_SECONDS
IOCTL_CCURPMFC_MMAP_SELECT
IOCTL_CCURPMFC_NO_COMMAND
IOCTL_CCURPMFC_PCI_CONFIG_REGISTERS
IOCTL_CCURPMFC_REMOVE_IRQ
IOCTL_CCURPMFC_RESET_BOARD
IOCTL_CCURPMFC_SELECT_ADC_READ_MODE
IOCTL_CCURPMFC_SELECT_DAC_WRITE_MODE
IOCTL_CCURPMFC_WAIT_FOR_INTERRUPT
IOCTL_CCURPMFC_WAIT_FOR_DAC_FIFO_TO_DRAIN
IOCTL_CCURPMFC_RELOAD_FIRMWARE
IOCTL_CCURPMFC_GET_ALL_BOARDS_DRIVER_INFO
IOCTL_CCURPMFC_WAKEUP_DIO_COS_INTERRUPT
IOCTL_CCURPMFC_WAIT_FOR_DIO_COS_INTERRUPT
```

IOCTL_CCURPMFC_ABORT_DMA: This *ioctl* does not have any arguments. Its purpose is to abort any DMA already in progress..

IOCTL_CCURPMFC_ADD_IRQ: This *ioctl* does not have any arguments. Its purpose is to setup the driver *interrupt handler* to handle interrupts. If support for MSI interrupts are configured, they will be enabled. Normally, there is no need to call this *ioctl* as the interrupt handler is already added when the driver is loaded. This *ioctl* should only be invoked if the user has issued the *IOCTL_CCURPMFC_REMOVE_IRQ* call earlier to remove the interrupt handler.

IOCTL_CCURPMFC_DISABLE_PCI_INTERRUPTS: This *ioctl* does not have any arguments. Its purpose is to disable PCI interrupts. This call shouldn't be used during normal reads or writes, as calls could time out. The driver handles enabling and disabling interrupts during its normal course of operation.

IOCTL_CCURPMFC_ENABLE_PCI_INTERRUPTS: This *ioctl* does not have any arguments. Its purpose is to enable PCI interrupts. This call shouldn't be used during normal reads or writes as calls could time out. The driver handles enabling and disabling interrupts during its normal course of operation.

IOCTL_CCURPMFC_GET_DRIVER_ERROR: The argument supplied to this *ioctl* is a pointer to the *ccurpmfc_user_error_t* structure. Information on the structure is located in the *ccurpmfc_user.h* include file. The error returned is the last reported error by the driver. If the argument pointer is *NULL*, the current error is reset to *CCURPMFC_SUCCESS.*

IOCTL_CCURPMFC_GET_DRIVER_INFO: The argument supplied to this *ioctl* is a pointer to the *ccurpmfc_driver_info_t* structure. Information on the structure is located in the *ccurpmfc_user.h* include file. This *ioctl* provides useful driver information.

IOCTL_CCURPMFC_GET_PHYSICAL_MEMORY: The argument supplied to this *ioctl* is a pointer to the *ccurpmfc_user_phys_mem_t* structure. Information on the structure is located in the *ccurpmfc_user.h* include file. If physical memory is not allocated, the call will fail; otherwise the call will return the physical memory address and size in bytes. The only reason to request and get physical memory from the driver is to allow the user to perform DMA operations and bypass the driver and library. Care must be taken when performing user level DMA, as incorrect programming could lead to unpredictable results, including but not limited to corrupting the kernel and any device connected to the system.

IOCTL_CCURPMFC_GET_ADC_READ_MODE: The argument supplied to this *ioctl* is a pointer to an *unsigned long int*. The value returned will be one of the ADC read modes as defined by the *enum _ccurpmfc_driver_ADC_read_mode_t* located in the *ccurpmfc_user.h* include file.

IOCTL_CCURPMFC_GET_DAC_WRITE_MODE: The argument supplied to this *ioctl* is a pointer to an *unsigned long int*. The value returned will be one of the DAC write modes as defined by the *enum _ccurpmfc_driver_DAC_write_mode_t* located in the *ccurpmfc_user.h* include file.

IOCTL_CCURPMFC_INIT_BOARD: This *ioctl* does not have any arguments. This call resets the board to a known initial default state. This call is currently identical to the *IOCTL_CCURPMFC_RESET_BOARD* call.

IOCTL_CCURPMFC_INTERRUPT_TIMEOUT_SECONDS: The argument supplied to this *ioctl* is a pointer to an *int*. It allows the user to change the default time out from 30 seconds to user supplied time out. This is the time that the read call will wait before it times out. The call could time out if a DMA fails to complete. The device should have been opened in the block mode (*O_NONBLOCK* not set) for reads to wait for an operation to complete.

IOCTL_CCURPMFC_MMAP_SELECT: The argument to this *ioctl* is a pointer to the *ccurpmfc_mmap_select_t* structure. Information on the structure is located in the *ccurpmfc_user.h* include file. This call needs to be made prior to the *mmap(2)* system call so as to direct the *mmap(2)* call to perform the requested mapping specified by this *ioctl*. The four possible mappings that are performed by the driver are to *mmap* the local register space *(CCURPMFC_SELECT_LOCAL_MMAP)*, the configuration register space *(CCURPMFC_SELECT_CONFIG_MMAP)* the physical memory *(CCURPMFC_SELECT_PHYS_MEM_MMAP)* that is created by the *mmap(2)* system call and the driver/library mapping *(CCURPMFC_SELECT_DRIVER_LIBRARY_MMAP)*.

IOCTL_CCURPMFC_NO_COMMAND: This *ioctl* does not have any arguments. It is only provided for debugging purpose and should not be used as it serves no purpose for the application.

IOCTL_CCURPMFC_PCI_CONFIG_REGISTERS: The argument supplied to this *ioctl* is a pointer to the *ccurpmfc_pci_config_reg_addr_mapping_*t structure whose definition is located in the *ccurpmfc_user.h* include file.

IOCTL_CCURPMFC_REMOVE_IRQ: This *ioctl* does not have any arguments. Its purpose is to remove the interrupt handler that was previously setup. The interrupt handler is managed internally by the driver and the library. The user should not issue this call, otherwise reads will time out.

IOCTL_CCURPMFC_RESET_BOARD: This *ioctl* does not have any arguments. This call resets the board to a known initial default state. This call is currently identical to the *IOCTL_CCURPMFC_INIT_BOARD* call.

IOCTL_CCURPMFC_SELECT_ADC_READ_MODE: The argument supplied to this *ioctl* is a pointer to an *unsigned long int*. The value set will be one of the ADC read modes as defined by the *enum _ccurpmfc_driver_ADC_read_mode_t* located in the *ccurpmfc_user.h* include file.

IOCTL_CCURPMFC_SELECT_DAC_WRITE_MODE: The argument supplied to this *ioctl* is a pointer to an *unsigned long int*. The value set will be one of the DAC write modes as defined by the *enum _ccurpmfc_driver_DAC_write_mode_t* located in the *ccurpmfc_user.h* include file.

IOCTL_CCURPMFC_WAIT_FOR_INTERRUPT: The argument to this *ioctl* is a pointer to the *ccurpmfc_driver_int_t* structure. Information on the structure is located in the *ccurpmfc_user.h* include file. The user can wait for a DMA or Analog signal complete interrupt. If a time out value greater than zero is specified, the call will time out after the specified seconds, otherwise it will not time out.

IOCTL_CCURPMFC_WAIT_FOR_DAC_FIFO_TO_DRAIN: The argument to this *ioctl* is a pointer to *threshold*. The call will block until the samples in the DAC FIFO falls below the threshold value.

IOCTL_CCURPMFC_RELOAD_FIRMWARE: This *ioctl* does not have any arguments. This call performs a reload of the latest firmware that was loaded into the board. Typically, this is used after a new firmware has been installed. It eliminates the need to reboot the kernel after a firmware update.

IOCTL_CCURPMFC_GET_ALL_BOARDS_DRIVER_INFO: The argument to this *ioctl* is a pointer to *ccurpmfc_all_boards_driver_info*. It provides the ability to supply all driver information for all the *ccurpmfc* cards in the system to the user.

IOCTL_CCURPMFC_WAKEUP_DIO_COS_INTERRUPT: This *ioctl* does not have any arguments The purpose of this call is to wake up a process that is blocked using the *IOCTL_CCURPMFC_WAIT_FOR_DIO_COS_INTERRUPT ioctl* waiting for a DIO change-of-state interrupt.
\
IOCTL_CCURPMFC_WAIT_FOR_DIO_COS_INTERRUPT: The argument to this *ioctl* is a pointer to the *ccurpmfc_driver_dio_cos_int_t* structure. Information on the structure is located in the *ccurpmfc_user.h* include file. The user can wait for a DIO complete interrupt with the *WakeupInterruptMas* mask. This call blocks indefinitely until a DIO change-of-state interrupt occurs. If a change-of-state interrupt occurs, this call returns useful DIO related statistics to the user. To cancel a pending wait, users can use the *IOCTL_CCURPMFC_WAKEUP_DIO_COS_INTERRUPT ioctl()* call.

2.1.3 mmap(2) system call

This system call provides the ability to map either the local board registers, the configuration board registers, create and map a physical memory that can be used for user DMA or driver/library structure mapping. Prior to making this system call, the user needs to issue the *ioctl(2)* system call with the *IOCTL_CCURPMFC_MMAP_SELECT* command. When mapping either the local board registers or the configuration board registers, the *ioctl* call returns the size of the register mapping which needs to be specified in the *mmap(2)* call. In the case of mapping a physical memory, the size of physical memory to be created is supplied to the *mmap(2)* call.

```
int *munmap_local_ptr;
```

```
ccurpmfc_local_ctrl_data_t  *local_ptr;
ccurpmfc_mmap_select_t  mmap_select;
unsigned long mmap_local_size;

mmap_select.select = CCURPMFC_SELECT_LOCAL_MMAP;
mmap_select.offset=0;
mmap_select.size=0;
ioctl(fp, IOCTL_CCURPMFC_MMAP_SELECT,(void *)&mmap_select);
mmap_local_size = mmap_select.size;

munmap_local_ptr = (int *) mmap((caddr_t)0, map_local_size,
                   (PROT_READ|PROT_WRITE), MAP_SHARED, fp, 0);

local_ptr = (ccurpmfc_local_ctrl_data_t *)munmap_local_ptr;
local_ptr = (ccurpmfc_local_ctrl_data_t *)((char *)local_ptr +
                                           mmap_select.offset);
.
.
.

if(munmap_local_ptr != NULL)
    munmap((void *)munmap_local_ptr, mmap_local_size);
```

2.1.4 read(2) system call

This system call currently supports ADC programmed I/O reads of channel registers and FIFO. The option selected is determined by the *ccurPMFC_ADC_Set_Driver_Read_Mode()* call.

CCURPMFC_ADC_PIO_CHANNEL: Perform .channel registers programmed I/O reads.
CCURPMFC_ADC_PIO_FIFO: Perform FIFO reads using programmed I/O.

2.1.5 write(2) system call

This system call currently supports DAC programmed I/O writes of channel registers and FIFO. The option selected is determined by the *ccurPMFC_DAC_Set_Driver_Write_Mode()* call.

CCURPMFC_DAC_PIO_CHANNEL: Perform .channel registers programmed I/O writes.
CCURPMFC_DAC_PIO_FIFO: Perform FIFO writes using programmed I/O.

.

2.2 Application Program Interface (API) Access

The API is the recommended method of communicating with the board for most users.

There are a lot of APIs that have multiple arguments to set various parameters. If the user only wishes to change certain parameters for the call, they need to get the current settings via a query API, change only those parameters that need to be modified and then invoke a setting API to update these parameters *(i.e. read/modify/write)*. This is a two API call operation.

A nice feature has been implemented in these APIs to simplify the user programming by having a common parameter CCURPMFC_DO_NOT_CHANGE which is a #define, that can be used for a lot of these calls. Arguments with this parameter will therefore cause the API to perform the read/modify/write operation instead of the user performing the same function with two API calls. The drawback to this approach is that some compilers will complain about the use of this parameter and therefore the user will require appropriate casting to get rid of warnings/errors.

The following are a list of calls that are available.

```
ccurPMFC_Abort_DMA()
ccurPMFC_ADC_Activate()
ccurPMFC_ADC_Get_CSR()
ccurPMFC_ADC_Get_Driver_Read_Mode()
ccurPMFC_ADC_Get_Fifo_Channel_Select()
ccurPMFC_ADC_Get_Fifo_Info()
ccurPMFC_ADC_Get_Fifo_Threshold()
ccurPMFC_ADC_Get_Negative_Cal()
ccurPMFC_ADC_Get_Offset_Cal()
ccurPMFC_ADC_Get_Positive_Cal()
ccurPMFC_ADC_Perform_Auto_Calibration()
ccurPMFC_ADC_Perform_External_Negative_Calibration()
ccurPMFC_ADC_Perform_External_Offset_Calibration()
ccurPMFC_ADC_Perform_External_Positive_Calibration()
ccurPMFC_ADC_Perform_Negative_Calibration()
ccurPMFC_ADC_Perform_Offset_Calibration()
ccurPMFC_ADC_Perform_Positive_Calibration()
ccurPMFC_ADC_Read_Channels()
ccurPMFC_ADC_Read_Channels_Calibration()
ccurPMFC_ADC_Reset_Calibration()
ccurPMFC_ADC_Reset_Fifo()
ccurPMFC_ADC_Set_CSR()
ccurPMFC_ADC_Set_Driver_Read_Mode()
ccurPMFC_ADC_Set_Fifo_Channel_Select()
ccurPMFC_ADC_Set_Fifo_Threshold()
ccurPMFC_ADC_Set_Negative_Cal()
ccurPMFC_ADC_Set_Offset_Cal()
ccurPMFC_ADC_Set_Positive_Cal()
ccurPMFC_ADC_Write_Channels_Calibration()
ccurPMFC_Add_Irq()
ccurPMFC_BoardExpirationTimeRemaining()
ccurPMFC_Clear_Driver_Error()
ccurPMFC_Clear_Lib_Error()
ccurPMFC_Clock_Generator_Soft_Reset()
ccurPMFC_Clock_Get_Generator_CSR()
ccurPMFC_Clock_Get_Generator_Info()
ccurPMFC_Clock_Get_Generator_Input_Clock_Enable()
ccurPMFC_Clock_Get_Generator_Input_Clock_Select()
```

```
ccurPMFC_Clock_Get_Generator_Input_Clock_Status()
ccurPMFC_Clock_Get_Generator_M_Divider()
ccurPMFC_Clock_Get_Generator_N_Divider()
ccurPMFC_Clock_Get_Generator_Output_Config()
ccurPMFC_Clock_Get_Generator_Output_Format()
ccurPMFC_Clock_Get_Generator_Output_Mode()
ccurPMFC_Clock_Get_Generator_Output_Mux()
ccurPMFC_Clock_Get_Generator_P_Divider()
ccurPMFC_Clock_Get_Generator_P_Divider_Enable()
ccurPMFC_Clock_Get_Generator_R_Divider()
ccurPMFC_Clock_Get_Generator_Revision()
ccurPMFC_Clock_Get_Generator_Value()
ccurPMFC_Clock_Get_Generator_Voltage_Select()
ccurPMFC_Clock_Get_Generator_Zero_Delay()
ccurPMFC_ReturnOutputFrequency()
ccurPMFC_Clock_Set_Generator_CSR()
ccurPMFC_Clock_Set_Generator_Input_Clock_Enable()
ccurPMFC_Clock_Set_Generator_Input_Clock_Select()
ccurPMFC_Clock_Set_Generator_M_Divider()
ccurPMFC_Clock_Set_Generator_N_Divider()
ccurPMFC_Clock_Set_Generator_Output_Config()
ccurPMFC_Clock_Set_Generator_Output_Format()
ccurPMFC_Clock_Set_Generator_Output_Mode()
ccurPMFC_Clock_Set_Generator_Output_Mux()
ccurPMFC_Clock_Set_Generator_P_Divider()
ccurPMFC_Clock_Set_Generator_P_Divider_Enable()
ccurPMFC_Clock_Set_Generator_R_Divider()
ccurPMFC_Clock_Set_Generator_Value()
ccurPMFC_Clock_Set_Generator_Voltage_Select()
ccurPMFC_Clock_Set_Generator_Zero_Delay()
ccurPMFC_Close()
ccurPMFC_Compute_All_Output_Clocks()
ccurPMFC_Convert_Physmem2avmm_Address()
ccurPMFC_Create_UserDioCosInterruptHandler()
ccurPMFC_Create_UserProcess()
ccurPMFC_DAC_Activate()
ccurPMFC_DAC_Get_CSR()
ccurPMFC_DAC_Get_Driver_Write_Mode()
ccurPMFC_DAC_Get_Fifo_Channel_Select()
ccurPMFC_DAC_Get_Fifo_Info()
ccurPMFC_DAC_Get_Fifo_Threshold()
ccurPMFC_DAC_Get_Fifo_Write_Count()
ccurPMFC_DAC_Get_Gain_Cal()
ccurPMFC_DAC_Get_Offset_Cal()
ccurPMFC_DAC_Get_Update_Source_Select()
ccurPMFC_DAC_Perform_Auto_Calibration()
ccurPMFC_DAC_Perform_Gain_Calibration()
ccurPMFC_DAC_Perform_Offset_Calibration()
ccurPMFC_DAC_Read_Channels_Calibration()
ccurPMFC_DAC_ReadBack_Channels()
ccurPMFC_DAC_Read_Channels()
ccurPMFC_DAC_Reset_Calibration()
ccurPMFC_DAC_Reset_Fifo()
ccurPMFC_DAC_Set_CSR()
ccurPMFC_DAC_Set_Driver_Write_Mode()
```

```
ccurPMFC_DAC_Set_Fifo_Channel_Select()
ccurPMFC_DAC_Set_Fifo_Threshold()
ccurPMFC_DAC_Set_Fifo_Write_Count()
ccurPMFC_DAC_Set_Gain_Cal()
ccurPMFC_DAC_Set_Offset_Cal()
ccurPMFC_DAC_Set_Update_Source_Select()
ccurPMFC_DAC_Wait_For_Channel_Idle()
ccurPMFC_DAC_Wait_For_Fifo_To_Drain()
ccurPMFC_DAC_Write_Channels()
ccurPMFC_DAC_Write_Channels_Calibration()
ccurPMFC_DataToVolts()
ccurPMFC_Destroy_AllUserProcess()
ccurPMFC_Destroy_UserDioCosInterruptHandler()
ccurPMFC_Destroy_UserProcess()
ccurPMFC_DIO_Activate()
ccurPMFC_DIO_Get_Channels_Polarity()
ccurPMFC_DIO_Get_COS_Channels_Edge_Sense()
ccurPMFC_DIO_Get_COS_Channels_Enable()
ccurPMFC_DIO_Get_COS_Channels_Mode()
ccurPMFC_DIO_Get_COS_Channels_Overflow()
ccurPMFC_DIO_Get_COS_Channels_Status()
ccurPMFC_DIO_Get_Input_Channels_Filter()
ccurPMFC_DIO_Get_Input_Snapshot()
ccurPMFC_DIO_Get_Mode()
ccurPMFC_DIO_Get_Output_Sync()
ccurPMFC_DIO_Get_Ports_Direction()
ccurPMFC_DIO_Read_Custom_Channel_Registers()
ccurPMFC_DIO_Read_Input_Channel_Registers()
ccurPMFC_DIO_Read_Output_Channel_Registers()
ccurPMFC_DIO_Set_Channels_Polarity()
ccurPMFC_DIO_Set_COS_Channels_Edge_Sense()
ccurPMFC_DIO_Set_COS_Channels_Enable()
ccurPMFC_DIO_Set_COS_Channels_Mode()
ccurPMFC_DIO_Set_Input_Channels_Filter()
ccurPMFC_DIO_Set_Input_Snapshot()
ccurPMFC_DIO_Set_Mode()
ccurPMFC_DIO_Set_Output_Sync()
ccurPMFC_DIO_Set_Ports_Direction()
ccurPMFC_DIO_Set_Ports_Direction_To_Input()
ccurPMFC_DIO_Set_Ports_Direction_To_Output()
ccurPMFC_DIO_Write_Output_Channel_Registers()
ccurPMFC_DIO_Write_Output_Channel_High_Registers()
ccurPMFC_DIO_Write_Output_Channel_Low_Registers()
ccurPMFC_Disable_Pci_Interrupts()
ccurPMFC_DMA_Configure()
ccurPMFC_DMA_Fire()
ccurPMFC_Enable_Pci_Interrupts()
ccurPMFC_Fast_Memcpy()
ccurPMFC_Fast_Memcpy_Unlocked()
ccurPMFC_Fast_Memcpy_Unlocked_FIFO()
ccurPMFC_Fraction_To_Hex()
ccurPMFC_Get_All_Boards_Driver_Info()
ccurPMFC_Get_Board_CSR()
ccurPMFC_Get_Board_Info()
ccurPMFC_Get_Calibration_CSR()
```

```
ccurPMFC_Get_Driver_Error()
ccurPMFC_Get_Driver_Info()
ccurPMFC_Get_Interrupt_Status()
ccurPMFC_Get_Interrupt_Timeout_Seconds()
ccurPMFC_Get_Lib_Error_Description()
ccurPMFC_Get_Lib_Error()
ccurPMFC_Get_Library_Info()
ccurPMFC_Get_Mapped_Config_Ptr()
ccurPMFC_Get_Mapped_Driver_Library_Ptr()
ccurPMFC_Get_Mapped_Local_Ptr()
ccurPMFC_Get_Open_File_Descriptor()
ccurPMFC_Get_Physical_Memory()
ccurPMFC_Get_RunCount_UserProcess()
ccurPMFC_Get_TestBus_Control()
ccurPMFC_Get_Value()
ccurPMFC_Hex_To_Fraction()
ccurPMFC_Identify_Board()
ccurPMFC_Initialize_Board()
ccurPMFC_IpCore_COS_Activate()
ccurPMFC_IpCore_COS_Configure()
ccurPMFC_IpCore_COS_Decode_Timestamp()
ccurPMFC_IpCore_COS_Get_Info()
ccurPMFC_IpCore_COS_Read()
ccurPMFC_IpCore_COS_Start_Stop()
ccurPMFC_IpCore_Get_Info()
ccurPMFC_IpCore_Get_Mapped_Ptr()
ccurPMFC_MMap_Physical_Memory()
ccurPMFC_MsgDma_Clone()
ccurPMFC_MsgDma_Configure_Descriptor()
ccurPMFC_MsgDma_Configure_Single()
ccurPMFC_MsgDma_Fire()
ccurPMFC_MsgDma_Fire_Single()
ccurPMFC_MsgDma_Free_Descriptor()
ccurPMFC_MsgDma_Get_Descriptor()
ccurPMFC_MsgDma_Get_Dispatcher_CSR()
ccurPMFC_MsgDma_Get_Prefetcher_CSR()
ccurPMFC_MsgDma_Release()
ccurPMFC_MsgDma_Seize()
ccurPMFC_MsgDma_Setup()
ccurPMFC_Munmap_Physical_Memory()
ccurPMFC_NanoDelay()
ccurPMFC_Open()
ccurPMFC_Pause_UserProcess()
ccurPMFC_Program_All_Output_Clocks()
ccurPMFC_Read()
ccurPMFC_Reload_Firmware()
ccurPMFC_Remove_Irq()
ccurPMFC_Reset_Board()
ccurPMFC_Reset_Clock()
ccurPMFC_Resume_UserProcess()
ccurPMFC_Return_Board_Info_Description()
ccurPMFC_SDRAM_Activate()
ccurPMFC_SDRAM_Get_CSR()
ccurPMFC_SDRAM_Read()
ccurPMFC_SDRAM_Set_CSR()
```

```
ccurPMFC_SDRAM_Write()
ccurPMFC_Set_Board_CSR()
ccurPMFC_Set_Calibration_CSR()
ccurPMFC_Set_Interrupt_Status()
ccurPMFC_Set_Interrupt_Timeout_Seconds()
ccurPMFC_Set_TestBus_Control()
ccurPMFC_Set_Value()
ccurPMFC_SPROM_Read()
ccurPMFC_SPROM_Read_Item()
ccurPMFC_SPROM_Write()
ccurPMFC_SPROM_Write_Item()
ccurPMFC_SPROM_Write_Override()
ccurPMFC_Transfer_Data()
ccurPMFC_Update_Clock_Generator_Divider()
ccurPMFC_UserProcess_Command()
ccurPMFC_VoltsToData()
ccurPMFC_Wait_For_Interrupt()
ccurPMFC_Write()
```

2.2.1 ccurPMFC_Abort_DMA()

This call will abort any DMA operation that is in progress. Normally, the user should not use this call unless they are providing their own DMA handling.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t ccurPMFC_Abort_DMA(void *Handle)

   Description: Abort any DMA in progress

   Input:   void *Handle                        (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_IOCTL_FAILED      (driver ioctl call failed)
************************************************************************/
```

2.2.2 ccurPMFC_ADC_Activate()

This call must be the first call to activate the ADC. Without activation, all other calls to the ADC will fail. The user can also use this call to return the current state of the ADC without any change by specifying a pointer to *current_state* and setting *activate* to *CCURPMFC_ADC_ALL_ENABLE_DO_NOT_CHANGE*. If the ADC is already active and the user issues a *CCURPMFC_ADC_ALL_ENABLE*, no additional activation will be performed. To cause the ADC to go through a full reset, the user needs to issue the *CCURPMFC_ADC_ALL_RESET* which will cause the ADC to disable and then re-enable, setting all its ADC values to a default state. ADC calibration data will also be reset.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Activate (void                       *Handle,
                          _ccurpmfc_adc_all_enable_t  activate,
                          _ccurpmfc_adc_all_enable_t  *current_state)

   Description: Activate/DeActivate ADC module

   Input:   void                              *Handle         (Handle pointer)
            _ccurpmfc_adc_all_enable_t        activate        (activate/deactivate)
               # CCURPMFC_ADC_ALL_DISABLE
               # CCURPMFC_ADC_ALL_ENABLE
               # CCURPMFC_ADC_ALL_RESET      (disable followed by enable)
               # CCURPMFC_ADC_ALL_ENABLE_DO_NOT_CHANGE
   Output:  _ccurpmfc_adc_all_enable_t        *current_state  (active/deactive)
               # CCURPMFC_ADC_ALL_DISABLE
               # CCURPMFC_ADC_ALL_ENABLE
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (device not open)
               # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION          (local region not present)
************************************************************************/
```

2.2.3 ccurPMFC_ADC_Get_CSR()

This call returns information from the ADC registers for the selected channel group.

```
/************************************************************************
```

```
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Get_CSR (void                  *Handle,
                         _ccurpmfc_adc_mask_t  adc_mask,
                         ccurpmfc_adc_csr_t    *adc_csr)

   Description: Get ADC Control and Status information

   Input:   void                                    *Handle  (Handle pointer)
            _ccurpmfc_adc_mask_t                    adc_mask (selected ADC mask)
               # CCURPMFC_ADC_MASK_0_7
               # CCURPMFC_ADC_MASK_8_15
               # CCURPMFC_ALL_ADC_MASK
   Output:  ccurpmfc_adc_csr_t                      *adc_csr (pointer to ADC csr)
               _ccurpmfc_adccsr_update_clock_t      adc_update_clock;
                  # CCURPMFC_ADC_UPDATE_CLOCK_NONE
                  # CCURPMFC_ADC_UPDATE_CLOCK_0
                  # CCURPMFC_ADC_UPDATE_CLOCK_1
                  # CCURPMFC_ADC_UPDATE_CLOCK_2
                  # CCURPMFC_ADC_UPDATE_CLOCK_3
                  # CCURPMFC_ADC_UPDATE_CLOCK_4
                  # CCURPMFC_ADC_UPDATE_CLOCK_5
                  # CCURPMFC_ADC_UPDATE_CLOCK_6
               _ccurpmfc_adccsr_input_signal_t      adc_input_signal;
                  # CCURPMFC_ADC_EXTERNAL_SIGNAL
                  # CCURPMFC_ADC_CALIBRATION_BUS
               _ccurpmfc_adccsr_dataformat_t        adc_data_format;
                  # CCURPMFC_ADC_OFFSET_BINARY
                  # CCURPMFC_ADC_TWOS_COMPLEMENT
               _ccurpmfc_adccsr_input_range_t       adc_input_range;
                  # CCURPMFC_ADC_BIPOLAR_10V
                  # CCURPMFC_ADC_BIPOLAR_5V
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE       (ADC is not active)
*****************************************************************************/
```

2.2.4 ccurPMFC_ADC_Get_Driver_Read_Mode()

This call returns the current driver ADC read mode. When a *read(2)* system call is issued, it is this mode that determines the type of read being performed by the driver.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Get_Driver_Read_Mode (void                                *Handle,
                                      _ccurpmfc_driver_ADC_read_mode_t    *mode)

   Description: Get current ADC read mode that will be selected by the 'read()'
                                                                          call

   Input:   void                                *Handle (Handle pointer)
   Output:  _ccurpmfc_driver_ADC_read_mode_t    *mode   (select ADC read mode)
               # CCURPMFC_ADC_PIO_CHANNEL
               # CCURPMFC_ADC_PIO_FIFO
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (library not open)
```

```
            # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
            # CCURPMFC_LIB_IOCTL_FAILED            (driver ioctl call failed)
            # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
 ***************************************************************************/
```

2.2.5 ccurPMFC_ADC_Get_Fifo_Channel_Select()

This call returns the current Fifo Channel selection mask. Only samples for these selected channels are placed in the fifo during sample collection.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Get_Fifo_Channel_Select(void       *Handle,
                    _ccurpmfc_adc_channel_mask_t  *adc_fifo_channel_select_mask)

   Description: ADC Get Fifo Channel Selection

   Input:   void                          *Handle            (handle pointer)
   Output:  _ccurpmfc_adc_channel_mask_t *adc_fifo_channel_select_mask
                                                             (channel select mask)
               # CCURPMFC_ADC_CHANNEL_MASK_0
               # CCURPMFC_ADC_CHANNEL_MASK_1
               # CCURPMFC_ADC_CHANNEL_MASK_2
               # CCURPMFC_ADC_CHANNEL_MASK_3
               # CCURPMFC_ADC_CHANNEL_MASK_4
               # CCURPMFC_ADC_CHANNEL_MASK_5
               # CCURPMFC_ADC_CHANNEL_MASK_6
               # CCURPMFC_ADC_CHANNEL_MASK_7
               # CCURPMFC_ADC_CHANNEL_MASK_8
               # CCURPMFC_ADC_CHANNEL_MASK_9
               # CCURPMFC_ADC_CHANNEL_MASK_10
               # CCURPMFC_ADC_CHANNEL_MASK_11
               # CCURPMFC_ADC_CHANNEL_MASK_12
               # CCURPMFC_ADC_CHANNEL_MASK_13
               # CCURPMFC_ADC_CHANNEL_MASK_14
               # CCURPMFC_ADC_CHANNEL_MASK_15
               # CCURPMFC_ALL_ADC_CHANNELS_MASK
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
 ***************************************************************************/
```

2.2.6 ccurPMFC_ADC_Get_Fifo_Info()

This call returns ADC FIFO information to the user.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Get_Fifo_Info (void                     *Handle,
                               ccurpmfc_adc_fifo_info_t *adc_fifo)

   Description: Get ADC FIFO control and Status information

   Input:   void                                      *Handle   (Handle pointer)
   Output:  ccurpmfc_adc_fifo_info_t                  *adc_fifo (pointer to ADC
                                                                 fifo struct)
               _ccurpmfc_adc_fifo_reset_t      reset;
```

```
                  # CCURPMFC_ADC_FIFO_ACTIVE
                  # CCURPMFC_ADC_FIFO_RESET
               _ccurpmfc_adc_fifo_overflow_t   overflow;
                  # CCURPMFC_ADC_FIFO_NO_OVERFLOW
                  # CCURPMFC_ADC_FIFO_OVERFLOW
               _ccurpmfc_adc_fifo_underflow_t  underflow;
                  # CCURPMFC_ADC_FIFO_NO_UNDERFLOW
                  # CCURPMFC_ADC_FIFO_UNDERFLOW
               _ccurpmfc_adc_fifo_full_t       full;
                  # CCURPMFC_ADC_FIFO_NOT_FULL
                  # CCURPMFC_ADC_FIFO_FULL
               _ccurpmfc_adc_fifo_threshold_t  threshold_exceeded;
                  # CCURPMFC_ADC_FIFO_THRESHOLD_NOT_EXCEEDED
                  # CCURPMFC_ADC_FIFO_THRESHOLD_EXCEEDED
               _ccurpmfc_adc_fifo_empty_t      empty;
                  # CCURPMFC_ADC_FIFO_NOT_EMPTY
                  # CCURPMFC_ADC_FIFO_EMPTY
               uint                            data_counter;
               uint                            threshold;
               uint                            max_threshold;
               uint                            driver_threshold;
               _ccurpmfc_adc_channel_mask_t    channel_select_mask;
                  # CCURPMFC_ADC_CHANNEL_MASK_0
                  # CCURPMFC_ADC_CHANNEL_MASK_1
                  # CCURPMFC_ADC_CHANNEL_MASK_2
                  # CCURPMFC_ADC_CHANNEL_MASK_3
                  # CCURPMFC_ADC_CHANNEL_MASK_4
                  # CCURPMFC_ADC_CHANNEL_MASK_5
                  # CCURPMFC_ADC_CHANNEL_MASK_6
                  # CCURPMFC_ADC_CHANNEL_MASK_7
                  # CCURPMFC_ADC_CHANNEL_MASK_8
                  # CCURPMFC_ADC_CHANNEL_MASK_9
                  # CCURPMFC_ADC_CHANNEL_MASK_10
                  # CCURPMFC_ADC_CHANNEL_MASK_11
                  # CCURPMFC_ADC_CHANNEL_MASK_12
                  # CCURPMFC_ADC_CHANNEL_MASK_13
                  # CCURPMFC_ADC_CHANNEL_MASK_14
                  # CCURPMFC_ADC_CHANNEL_MASK_15
                  # CCURPMFC_ALL_ADC_CHANNELS_MASK
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
 ************************************************************************/
```

2.2.7 ccurPMFC_ADC_Get_Fifo_Threshold()

This call returns the ADC Fifo threshold information.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Get_Fifo_Threshold(void *Handle,
                                   uint *adc_threshold)

   Description: ADC Get Fifo Threshold

   Input:   void                    *Handle        (handle pointer)
   Output:  uint                    *adc_threshold (ADC fifo threshold)
```

```
  Return:  _ccurpmfc_lib_error_number_t
              # CCURPMFC_LIB_NO_ERROR             (successful)
              # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
              # CCURPMFC_LIB_NOT_OPEN             (device not open)
              # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
              # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
              # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE    (ADC is not active)
 ***************************************************************************/
```

2.2.8 ccurPMFC_ADC_Get_Negative_Cal()

This call returns the ADC negative calibration information for all the channels.

```
/***************************************************************************
  _ccurpmfc_lib_error_number_t
  ccurPMFC_ADC_Get_Negative_Cal(void               *Handle,
                                ccurpmfc_adc_cal_t *cal)

  Description: Get the ADC Negative Calibration data.

  Input:   void                       *Handle  (handle pointer)
  Output:  ccurpmfc_adc_cal_t         *cal     (pointer to board cal)
              uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
              double  Float[CCURPMFC_MAX_ADC_CHANNELS];
  Return:  _ccurpmfc_lib_error_number_t
              # CCURPMFC_LIB_NO_ERROR          (successful)
              # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
              # CCURPMFC_LIB_NOT_OPEN          (device not open)
              # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
              # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
              # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
 ***************************************************************************/
```

2.2.9 ccurPMFC_ADC_Get_Offset_Cal()

This call returns the ADC offset calibration information for all the channels.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Get_Offset_Cal(void               *Handle,
                               ccurpmfc_adc_cal_t *cal)

   Description: Get the ADC Offset Calibration data.

   Input:   void                       *Handle  (handle pointer)
   Output:  ccurpmfc_adc_cal_t         *cal     (pointer to board cal)
               uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
               double  Float[CCURPMFC_MAX_ADC_CHANNELS];
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
 ***************************************************************************/
```

2.2.10 ccurPMFC_ADC_Get_Positive_Cal()

This call returns the ADC positive calibration information for all the channels.

```
/***************************************************************************
```

```
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Get_Positive_Cal(void               *Handle,
                                 ccurpmfc_adc_cal_t *cal)

   Description: Get the ADC Positive Calibration data.

   Input:   void                      *Handle  (handle pointer)
   Output:  ccurpmfc_adc_cal_t        *cal     (pointer to board cal)
               uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
               double  Float[CCURPMFC_MAX_ADC_CHANNELS];
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
 ************************************************************************/
```

2.2.11 ccurPMFC_ADC_Perform_Auto_Calibration()

This single call performs a full ADC calibration of all the channels using the internal reference voltages.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Perform_Auto_Calibration(void *Handle)

   Description: Perform ADC Auto Calibration

   Input:   void   *Handle                          (handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE           (no free PLL available)
               # CCURPMFC_LIB_IO_ERROR              (read error)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
 ************************************************************************/
```

2.2.12 ccurPMFC_ADC_Perform_External_Negative_Calibration()

Use this call to perform an external negative calibration. Prior to calling this function, the ADC inputs must be provided with a negative signal close to -10 Volts, otherwise this call will fail. Additionally, the user can specify a range of channels.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Perform_External_Negative_Calibration(void      *Handle,
                                      _ccurpmfc_adc_channel_t chan_start,
                                      _ccurpmfc_adc_channel_t chan_end,
                                      double                  ReferenceVoltage)

   Description: Perform ADC External Negative Calibration

   Input:    void                    *Handle          (handle pointer)
             _ccurpmfc_adc_channel_t chan_start       (start channel)
             _ccurpmfc_adc_channel_t chan_end         (end channel)
             double                  ReferenceVoltage (Reference Voltage)
```

```
   Output:   none
   Return:   _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (library not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE            (no free PLL available)
               # CCURPMFC_LIB_IO_ERROR               (read error)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE      (ADC is not active)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
 ***************************************************************************/
```

2.2.13 ccurPMFC_ADC_Perform_External_Offset_Calibration()

Use this call to perform an external offset calibration. Prior to calling this function, the ADC inputs must be provided with a offset signal close to 0 Volts, otherwise this call will fail. Additionally, the user can specify a range of channels. Once this call is executed, the user will need to perform external negative and external positive calibrations as this call resets these gains to 1.0 prior to calibration.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Perform_External_Offset_Calibration(void          *Handle,
                                       _ccurpmfc_adc_channel_t chan_start,
                                       _ccurpmfc_adc_channel_t chan_end)

   Description: Perform ADC External Offset Calibration

   Input:   void                    *Handle         (handle pointer)
            _ccurpmfc_adc_channel_t chan_start      (start channel)
            _ccurpmfc_adc_channel_t chan_end        (end channel)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (library not open)
               # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE           (no free PLL available)
               # CCURPMFC_LIB_IO_ERROR              (read error)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
 ***************************************************************************/
```

2.2.14 ccurPMFC_ADC_Perform_External_Positive_Calibration()

Use this call to perform an external positive calibration. Prior to calling this function, the ADC inputs must be provided with a positive signal close to +10 Volts, otherwise this call will fail. Additionally, the user can specify a range of channels.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
  ccurPMFC_ADC_Perform_External_Positive_Calibration(void       *Handle,
                                      _ccurpmfc_adc_channel_t chan_start,
                                      _ccurpmfc_adc_channel_t chan_end,
                                      double                  ReferenceVoltage)

   Description: Perform ADC External Positive Calibration

   Input:   void                    *Handle          (handle pointer)
            _ccurpmfc_adc_channel_t chan_start       (start channel)
```

```
          _ccurpmfc_adc_channel_t chan_end         (end channel)
          double                  ReferenceVoltage (Reference Voltage)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (library not open)
               # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE           (no free PLL available)
               # CCURPMFC_LIB_IO_ERROR              (read error)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
 ***************************************************************************/
```

2.2.15 ccurPMFC_ADC_Perform_Negative_Calibration()

This call performs a negative calibration using the internal reference voltage.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Perform_Negative_Calibration(void *Handle)

   Description: Perform ADC Negative Calibration

   Input:   void   *Handle                          (handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE           (no free PLL available)
               # CCURPMFC_LIB_IO_ERROR              (read error)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
 ***************************************************************************/
```

2.2.16 ccurPMFC_ADC_Perform_Offset_Calibration()

This call performs an offset calibration using the internal reference voltage. Once this call is executed, the user will need to perform negative and positive calibrations as this call resets these gains to 1.0 prior to calibration.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Perform_Offset_Calibration(void *Handle)

   Description: Perform ADC Offset Calibration

   Input:   void   *Handle                          (handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE           (no free PLL available)
               # CCURPMFC_LIB_IO_ERROR              (read error)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
```

```
 ****************************************************************************/
```

2.2.17 ccurPMFC_ADC_Perform_Positive_Calibration()

This call performs a positive calibration using the internal reference voltage.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Perform_Positive_Calibration(void *Handle)

   Description: Perform ADC Positive Calibration

   Input:   void   *Handle                          (handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE           (no free PLL available)
               # CCURPMFC_LIB_IO_ERROR              (read error)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
 ****************************************************************************/
```

2.2.18 ccurPMFC_ADC_Read_Channels ()

This call provides the user an easy method of reading the ADC channels. User can supply a channel mask. If pointer to *adc_csr* is NULL, then the routine itself computes the current ADC configuration. For performance, the user should get the current ADC configuration using the *ccurPMFC_ADC_Get_CSR()* call to get the current settings and pass it to this routine. Hence, if the configuration is not changed, the user can continuously invoke *ccurPMFC_ADC_Read_Channels()* routine without incurring the additional overhead of routine calling the *ccurPMFC_ADC_Get_CSR()* call.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Read_Channels(void                          *Handle,
                              _ccurpmfc_adc_channel_mask_t ChanMask,
                              _ccurpmfc_adc_csr_t          *adc_csr,
                              ccurpmfc_adc_volts_t         *adc_volts)

   Description: Read ADC Channels

   Input:   void                            *Handle       (Handle pointer)
            _ccurpmfc_adc_channel_mask_t    ChanMask      (specify channel mask)
               # CCURPMFC_ADC_CHANNEL_MASK_0
               # CCURPMFC_ADC_CHANNEL_MASK_1
               # CCURPMFC_ADC_CHANNEL_MASK_2
               # CCURPMFC_ADC_CHANNEL_MASK_3
               # CCURPMFC_ADC_CHANNEL_MASK_4
               # CCURPMFC_ADC_CHANNEL_MASK_5
               # CCURPMFC_ADC_CHANNEL_MASK_6
               # CCURPMFC_ADC_CHANNEL_MASK_7
               # CCURPMFC_ADC_CHANNEL_MASK_8
               # CCURPMFC_ADC_CHANNEL_MASK_9
               # CCURPMFC_ADC_CHANNEL_MASK_10
               # CCURPMFC_ADC_CHANNEL_MASK_11
               # CCURPMFC_ADC_CHANNEL_MASK_12
               # CCURPMFC_ADC_CHANNEL_MASK_13
               # CCURPMFC_ADC_CHANNEL_MASK_14
               # CCURPMFC_ADC_CHANNEL_MASK_15
```

```
               # CCURPMFC_ALL_ADC_CHANNELS_MASK
            _ccurpmfc_adc_csr_t             *adc_csr (pointer to ADC csr)
               _ccurpmfc_adccsr_update_clock_t      adc_update_clock
                  # CCURPMFC_ADC_UPDATE_CLOCK_NONE
                  # CCURPMFC_ADC_UPDATE_CLOCK_0
                  # CCURPMFC_ADC_UPDATE_CLOCK_1
                  # CCURPMFC_ADC_UPDATE_CLOCK_2
                  # CCURPMFC_ADC_UPDATE_CLOCK_3
                  # CCURPMFC_ADC_UPDATE_CLOCK_4
                  # CCURPMFC_ADC_UPDATE_CLOCK_5
                  # CCURPMFC_ADC_UPDATE_CLOCK_6
               _ccurpmfc_adccsr_input_signal_t      adc_input_signal
                  # CCURPMFC_ADC_EXTERNAL_SIGNAL
                  # CCURPMFC_ADC_CALIBRATION_BUS
               _ccurpmfc_adccsr_data_format_t       adc_data_format
                  # CCURPMFC_ADC_OFFSET_BINARY
                  # CCURPMFC_ADC_TWOS_COMPLEMENT
               _ccurpmfc_adccsr_input_range_t       adc_input_range
                  # CCURPMFC_ADC_BIPOLAR_10V
                  # CCURPMFC_ADC_BIPOLAR_5V
   Output:  ccurpmfc_adc_volts_t                *adc_volts (pointer to ADC volts)
                  uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
                  double  Float[CCURPMFC_MAX_ADC_CHANNELS];
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (library not open)
               # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
 ************************************************************************/
```

2.2.19 ccurPMFC_ADC_Read_Channels_Calibration()

This routine reads the ADC channel calibration registers and dumps all of them to the user specified file. If the file name specified is NULL, then information is written to *stdout.*

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Read_Channels_Calibration(void *Handle,
                                          char *filename)

   Description: Read ADC Channels Calibration

   Input:    void   *Handle                       (handle pointer)
   Output:   char   *filename                     (pointer to filename)
   Return:   _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (local region not present)
               # CCURPMFC_LIB_CANNOT_OPEN_FILE    (cannot open calib. file)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE   (ADC is not active)
 ************************************************************************/
```

e.g.

```
#Date           : Fri Dec 14 11:46:29 2018

#Chan   Negative               Offset                 Positive
#====   ========               ======                 ========
 ch00:  0.99749580910429358482  0.00061035156250000000  0.99747039563953876495
 ch01:  0.99804047681391239166  0.00000000000000000000  0.99805101659148931503
```

```
ch02:  0.99755537323653697968 -0.00030517578125000000  0.99758055899292230606
ch03:  0.99775091651827096939 -0.00030517578125000000  0.99780559260398149490
ch04:  0.99806733522564172745  0.00030517578125000000  0.99810541700571775436
ch05:  0.99849707912653684616  0.00030517578125000000  0.99846695689484477043
ch06:  0.99810661701485514641 -0.00030517578125000000  0.99812920438125729561
ch07:  0.99777368875220417976  0.00000000000000000000  0.99782193917781114578
ch08:  0.99798910086974501610  0.00030517578125000000  0.99797881394624710083
ch09:  0.99798714602366089821 -0.00030517578125000000  0.99796623829752206802
ch10:  0.99733591498807072639 -0.00030517578125000000  0.99737271666526794434
ch11:  0.99738420499488711357  0.00030517578125000000  0.99731908272951841354
ch12:  0.99858933826908469200  0.00000000000000000000  0.99866439774632453918
ch13:  0.99918560683727264404 -0.00030517578125000000  0.99927572812885046005
ch14:  0.99825186049565672874 -0.00061035156250000000  0.99833006644621491432
ch15:  0.99864477431401610374  0.00000000000000000000  0.99868657672777771950
```

2.2.20 ccurPMFC_ADC_Reset_Calibration()

This call resets the ADC calibration values on the card.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Reset_Calibration(void *Handle)

   Description: ADC Reset Calibration

   Input:   void                        *Handle  (handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR           (successful)
               # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN           (device not open)
               # CCURPMFC_LIB_INVALID_ARG        (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION    (local region not present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE  (ADC is not active)
 *************************************************************************/
```

2.2.21 ccurPMFC_ADC_Reset_Fifo()

This call provides the ability to reset the ADC Fifo to the power-on state. User can elect to activate the FIFO after a reset in order for data collection to resume immediately.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Reset_Fifo(void                        *Handle,
                           _ccurpmfc_adc_fifo_reset_t  activate)

   Description: ADC Reset Fifo

   Input:   void                        *Handle  (handle pointer)
            _ccurpmfc_adc_fifo_reset_t  activate (activate converter)
               # CCURPMFC_ADC_FIFO_ACTIVATE
               # CCURPMFC_ADC_FIFO_RESET
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR           (successful)
               # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN           (device not open)
               # CCURPMFC_LIB_INVALID_ARG        (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION    (local region not present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE  (ADC is not active)
 *************************************************************************/
```

2.2.22 ccurPMFC_ADC_Set_CSR()

This call sets the ADC control registers for the selected channel group.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Set_CSR (void                  *Handle,
                         _ccurpmfc_adc_mask_t  adc_mask,
                         ccurpmfc_adc_csr_t    *adc_csr)

   Description: Set ADC Control and Status information

   Input:   void                                    *Handle  (Handle pointer)
            _ccurpmfc_adc_mask_t                    adc_mask (selected ADC mask)
               # CCURPMFC_ADC_MASK_0_7
               # CCURPMFC_ADC_MASK_8_15
               # CCURPMFC_ALL_ADC_MASK
            ccurpmfc_adc_csr_t                      *adc_csr (pointer to ADC csr)
               _ccurpmfc_adccsr_update_clock_t      adc_update_clock;
                  # CCURPMFC_ADC_UPDATE_CLOCK_NONE
                  # CCURPMFC_ADC_UPDATE_CLOCK_0
                  # CCURPMFC_ADC_UPDATE_CLOCK_1
                  # CCURPMFC_ADC_UPDATE_CLOCK_2
                  # CCURPMFC_ADC_UPDATE_CLOCK_3
                  # CCURPMFC_ADC_UPDATE_CLOCK_4
                  # CCURPMFC_ADC_UPDATE_CLOCK_5
                  # CCURPMFC_ADC_UPDATE_CLOCK_6
                  # CCURPMFC_ADC_UPDATE_CLOCK_DO_NOT_CHANGE
               _ccurpmfc_adccsr_input_signal_t      adc_input_signal;
                  # CCURPMFC_ADC_EXTERNAL_SIGNAL
                  # CCURPMFC_ADC_CALIBRATION_BUS
                  # CCURPMFC_ADC_INPUT_SIGNAL_DO_NOT_CHANGE
               _ccurpmfc_adccsr_data_format_t       adc_data_format;
                  # CCURPMFC_ADC_OFFSET_BINARY
                  # CCURPMFC_ADC_TWOS_COMPLEMENT
                  # CCURPMFC_ADC_DATA_FORMAT_DO_NOT_CHANGE
               _ccurpmfc_adccsr_input_range_t       adc_input_range;
                  # CCURPMFC_ADC_BIPOLAR_10V
                  # CCURPMFC_ADC_BIPOLAR_5V
                  # CCURPMFC_ADC_INPUT_RANGE_DO_NOT_CHANGE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (library not open)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (device not open)
               # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION          (local region not
                                                        present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE        (ADC is not active)
*************************************************************************/
```

2.2.23 ccurPMFC_ADC_Set_Driver_Read_Mode()

This call sets the current driver ADC read mode. When a *read(2)* system call is issued, it is this mode that determines the type of read being performed by the driver. Refer to the *read(2)* system call under *Direct Driver Access* section for more information on the various modes.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
```

```
   ccurPMFC_ADC_Set_Driver_Read_Mode (void                                *Handle,
                                      _ccurpmfc_driver_ADC_read_mode_t   mode)

   Description: Select Driver ADC Read Mode

   Input:   void                              *Handle (Handle pointer)
            _ccurpmfc_driver_ADC_read_mode_t mode    (select ADC read mode)
               # CCURPMFC_ADC_PIO_CHANNEL
               # CCURPMFC_ADC_PIO_FIFO
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (device not open)
               # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
 ***************************************************************************/
```

2.2.24 ccurPMFC_ADC_Set_Fifo_Channel_Select()

This call allows the user to select a set of channels that need to be captured in the ADC Fifo.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Set_Fifo_Channel_Select(void        *Handle,
                                        _ccurpmfc_adc_channel_mask_t
                                                    adc_fifo_channel_select_mask)

   Description: ADC Set Fifo Channel Selection

   Input:   void                          *Handle          (handle pointer)
            _ccurpmfc_adc_channel_mask_t adc_fifo_channel_select_mask
                                                          (channel select mask)
               # CCURPMFC_ADC_CHANNEL_MASK_0
               # CCURPMFC_ADC_CHANNEL_MASK_1
               # CCURPMFC_ADC_CHANNEL_MASK_2
               # CCURPMFC_ADC_CHANNEL_MASK_3
               # CCURPMFC_ADC_CHANNEL_MASK_4
               # CCURPMFC_ADC_CHANNEL_MASK_5
               # CCURPMFC_ADC_CHANNEL_MASK_6
               # CCURPMFC_ADC_CHANNEL_MASK_7
               # CCURPMFC_ADC_CHANNEL_MASK_8
               # CCURPMFC_ADC_CHANNEL_MASK_9
               # CCURPMFC_ADC_CHANNEL_MASK_10
               # CCURPMFC_ADC_CHANNEL_MASK_11
               # CCURPMFC_ADC_CHANNEL_MASK_12
               # CCURPMFC_ADC_CHANNEL_MASK_13
               # CCURPMFC_ADC_CHANNEL_MASK_14
               # CCURPMFC_ADC_CHANNEL_MASK_15
               # CCURPMFC_ALL_ADC_CHANNELS_MASK
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE    (ADC is not active)
 ***************************************************************************/
```

2.2.25 ccurPMFC_ADC_Set_Fifo_Threshold()

This call allows the user to set the ADC Fifo Threshold.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Set_Fifo_Threshold(void *Handle,
                                   uint adc_threshold)

   Description: ADC Set Fifo Threshold

   Input:   void                    *Handle        (handle pointer)
            uint                    adc_threshold  (ADC fifo threshold)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE    (ADC is not active)
 *************************************************************************/
```

2.2.26 ccurPMFC_ADC_Set_Negative_Cal()

This call allows the user to set the negative calibration data for all the channels by supplying floating point *Float* gains to the call. Users can supply CCURPMFC_DO_NOT_CHANGE as a gain for any channel that should not be changed. Additionally, this call will return the *RAW* value of the gain supplied that is written to the board.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Set_Negative_Cal(void               *Handle,
                                 ccurpmfc_adc_cal_t *cal)

   Description: Set the ADC Negative Calibration data.

   Input:    void                       *Handle       (handle pointer)
             ccurpmfc_adc_cal_t         *cal          (pointer to board cal)
               uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
               double  Float[CCURPMFC_MAX_ADC_CHANNELS];
   Output:   none
   Return:   _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR              (successful)
                # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN              (library not open)
                # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
                # CCURPMFC_LIB_NO_RESOURCE           (no free PLL available)
                # CCURPMFC_LIB_IO_ERROR              (read error)
                # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
 *************************************************************************/
```

2.2.27 ccurPMFC_ADC_Set_Offset_Cal()

This call allows the user to set the offset calibration data for all the channels by supplying floating point *Float* offset to the call. Users can supply CCURPMFC_DO_NOT_CHANGE as a gain for any channel that should not be changed. Additionally, this call will return the *Raw* value of the offset supplied that is written to the board.

```
/*************************************************************************
```

```
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Set_Offset_Cal(void               *Handle,
                               ccurpmfc_adc_cal_t *cal)

   Description: Set the ADC Offset Calibration data.

   Input:    void                       *Handle       (handle pointer)
             ccurpmfc_adc_cal_t         *cal          (pointer to board cal)
               uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
               double  Float[CCURPMFC_MAX_ADC_CHANNELS];
   Output:   none
   Return:   _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR              (successful)
                # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN              (library not open)
                # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
                # CCURPMFC_LIB_NO_RESOURCE           (no free PLL available)
                # CCURPMFC_LIB_IO_ERROR              (read error)
                # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
 ****************************************************************************/
```

2.2.28 ccurPMFC_ADC_Set_Positive_Cal()

This call allows the user to set the positive calibration data for all the channels by supplying floating point *Float* gains to the call. Users can supply CCURPMFC_DO_NOT_CHANGE as a gain for any channel that should not be changed. Additionally, this call will return the *Raw* value of the gain supplied that is written to the board.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Set_Positive_Cal(void               *Handle,
                                 ccurpmfc_adc_cal_t *cal)

   Description: Set the ADC Positive Calibration data.

   Input:    void                       *Handle       (handle pointer)
             ccurpmfc_adc_cal_t         *cal          (pointer to board cal)
               uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
               double  Float[CCURPMFC_MAX_ADC_CHANNELS];
   Output:   none
   Return:   _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR              (successful)
                # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN              (library not open)
                # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
                # CCURPMFC_LIB_NO_RESOURCE           (no free PLL available)
                # CCURPMFC_LIB_IO_ERROR              (read error)
                # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
 ****************************************************************************/
```

2.2.29 ccurPMFC_ADC_Write_Channels_Calibration()

This call allows the user to write the calibration registers from a user supplied calibration file. The format of the file is similar to that generated by the *ccurPMFC_ADC_Read_Channels_Calibration()* call. File can contain comments if they start with '#', '*', or empty lines. Additionally, users need only specify those channels that they wish to calibrate and the order of specifying channels in not important, however, the format of each channel entry needs to be adhered to. E.g. <chxx:> <negative> <offset> <positive>

```
/****************************************************************************
```

```
   _ccurpmfc_lib_error_number_t
   ccurPMFC_ADC_Write_Channels_Calibration(void *Handle,
                                           char *filename)

   Description: Write Channels Calibration

   Input:    void   *Handle                       (handle pointer)
   Output:   char   *filename                     (pointer to filename)
   Return:   _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR           (successful)
               # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN           (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION    (local region not present)
               # CCURPMFC_LIB_CANNOT_OPEN_FILE   (cannot open calib. file)
               # CCURPMFC_LIB_INVALID_ARG        (invalid argument)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE  (ADC is not active)
 ***************************************************************************/
```

e.g.

```
#Date           : Fri Dec 14 11:46:29 2018

#Chan   Negative               Offset                 Positive
#====   ========               ======                 ========
 ch00:  0.99749580910429358482  0.00061035156250000000  0.99747039563953876495
 ch01:  0.99804047681391239166  0.00000000000000000000  0.99805101659148931503
 ch02:  0.99755537323653697968 -0.00030517578125000000  0.99758055899292230606
 ch03:  0.99775091651827096939 -0.00030517578125000000  0.99780559260398149490
 ch04:  0.99806733522564172745  0.00030517578125000000  0.99810541700571775436
 ch05:  0.99849707912653684616  0.00030517578125000000  0.99846695689484477043
 ch06:  0.99810661701485514641 -0.00030517578125000000  0.99812920438125729561
 ch07:  0.99777368875220417976  0.00000000000000000000  0.99782193917781114578
 ch08:  0.99798910086974501610  0.00030517578125000000  0.99797881394624710083
 ch09:  0.99798714602366089821 -0.00030517578125000000  0.99796623829752206802
 ch10:  0.99733591498807072639 -0.00030517578125000000  0.99737271666526794434
 ch11:  0.99738420499488711357  0.00030517578125000000  0.99731908272951841354
 ch12:  0.99858933826908469200  0.00000000000000000000  0.99866439774632453918
 ch13:  0.99918560683727264404 -0.00030517578125000000  0.99927572812885046005
 ch14:  0.99825186049565672874 -0.00061035156250000000  0.99833006644621491432
 ch15:  0.99864477431401610374  0.00000000000000000000  0.99868657672777771950
```

2.2.30 ccurPMFC_Add_Irq()

This call will add the driver interrupt handler if it has not been added. Normally, the user should not use this call unless they want to disable the interrupt handler and then re-enable it.

```
/***************************************************************************
   int ccurPMFC_Add_Irq(void *Handle)

   Description: By default, the driver assigns an interrupt handler to handle
                device interrupts. If the interrupt handler was removed using
                the ccurPMFC_Remove_Irq(), then this call adds it back.

   Input:   void *Handle                        (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (library not open)
               # CCURPMFC_LIB_IOCTL_FAILED      (driver ioctl call failed)
***************************************************************************/
```

2.2.31 ccurPMFC_BoardExpirationTimeRemaining()

This call provides useful information about the expiration date of the card if it has restricted licensing.

```
/*****************************************************************************
_ccurpmfc_lib_error_number_t
ccurPMFC_BoardExpirationTimeRemaining(void                      *Handle,
                                      time_t                    *SecondsToExpire,
                                      ccurpmfc_date_string_t    *GmtDateTimeString,
                                      ccurpmfc_date_string_t    *LocalDateTimeString,
                                      _ccurpmfc_firmware_state  *FirmwareState)

   Description: Number of seconds to expire on a restricted card

   Input:   void                        *Handle              (Handle pointer)
   Output:  time_t                      *SecondsToExpire     (seconds to expire)
            ccurpmfc_date_string_t      *GmtDateTimeString   (GMT date/time
                                                              string)
               char  date[CCURPMFC_DATE_TIME_STRING_SIZE]
            ccurpmfc_date_string_t      *LocalDateTimeString (Local date/time
                                                              string)
               char  date[CCURPMFC_DATE_TIME_STRING_SIZE]
            _ccurpmfc_firmware_state    *FirmwareState       (Firmware State)
               # CCURPMFC_FIRMWARE_STATE_UNRESTRICTED
               # CCURPMFC_FIRMWARE_STATE_RESTRICTED
               # CCURPMFC_FIRMWARE_STATE_EXPIRED
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (library not open)
 *****************************************************************************/
```

Mandatory arguments to the call are **Handle* and **SecondsToExpire.* Rest of the arguments are optional and be set to *NULL.*

*SecondsToExpire – If the board has an expiration date, this call will return the number of seconds this card can be used before it expires. ***Once the card has expired, this call will not be reached as the device open will fail with an authorization error.***

It the board has no expiration date, this call will return zero as the number of seconds.

*GmtDateTimeString – If the board has an expiration date, this ascii GMT date representation of the expiration date is available in this variable if it is not NULL

*LocalDateTimeString – If the board has an expiration date, this ascii Local date representation of the expiration date is available in this variable if it is not NULL

*FirmwareState – This returns the current state of the installed firmware. I can be one of:

- CCURPMFC_FIRMWARE_STATE_UNRESTRICTED. This firmware has no restrictions.
- CCURPMFC_FIRMWARE_STATE_RESTRICTED. This firmware has restrictions. It is possible that and expiration date restriction is not present.
- CCURPMFC_FIRMWARE_STATE_EXPIRED. This firmware has restrictions. One of the restrictions is the expiration date which has expired. Typically, you may not see this state as the utility will fail during the open with an authentication error.

2.2.32 ccurPMFC_Clear_Driver_Error()

This call resets the last driver error that was maintained internally by the driver to *CCURPMFC_SUCCESS*.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t ccurPMFC_Clear_Driver_Error(void *Handle)

   Description: Clear any previously generated driver related error.

   Input:   void *Handle                        (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_IOCTL_FAILED      (driver ioctl call failed)
*****************************************************************************/
```

2.2.33 ccurPMFC_Clear_Lib_Error()

This call resets the last library error that was maintained internally by the API.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t ccurPMFC_Clear_Lib_Error(void *Handle)

   Description: Clear any previously generated library related error.

   Input:   void *Handle                        (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
*****************************************************************************
```

2.2.34 ccurPMFC_Clock_Generator_Soft_Reset()

Perform a soft clock reset on all the output clocks.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t ccurPMFC_Clock_Generator_Soft_Reset(void *Handle)

   Description: Perform Soft Reset to Clock Generator

   Input:   void *Handle                         (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
              # CCURPMFC_LIB_NO_ERROR            (successful)
              # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
              # CCURPMFC_LIB_NOT_OPEN            (device not open)
              # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
  *****************************************************************************/
```

2.2.35 ccurPMFC_Clock_Get_Generator_CSR()

Return the clock generator control and status register.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_CSR (void                  *Handle,
                                     ccurpmfc_clkgen_csr_t *CgCsr)

   Description: Get Generator Control and Status information

   Input:   void                                    *Handle     (Handle pointer)
   Output:  ccurpmfc_clkgen_csr_t                   *CgCsr      (pointer to clock
```

```
                                                                generator csr)
            _ccurpmfc_clkgen_interface_t             interface
               # CCURPMFC_CLOCK_GENERATOR_INTERFACE_IDLE
               # CCURPMFC_CLOCK_GENERATOR_INTERFACE_BUSY
            _ccurpmfc_clkgen_output_t                output
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_DISABLE
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_ENABLE
            _ccurpmfc_clkgen_state_t                 state
               # CCURPMFC_CLOCK_GENERATOR_ACTIVE
               # CCURPMFC_CLOCK_GENERATOR_RESET
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                        (successful)
               # CCURPMFC_LIB_BAD_HANDLE                      (no/bad handler
                                                               supplied)
               # CCURPMFC_LIB_NOT_OPEN                        (device not open)
               # CCURPMFC_LIB_INVALID_ARG                     (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION                 (local region not
                                                               present)
 ****************************************************************************/
```

2.2.36 ccurPMFC_Clock_Get_Generator_Info()

This call returns the clock generator information for the selected output.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Info (void                          *Handle,
                              _ccurpmfc_clock_generator_output_t   WhichOutput,
                              ccurpmfc_clock_generator_info_t      *CgInfo)

   Description: Get Clock Generator Information

   Input:   void                                    *Handle     (Handle pointer)
            _ccurpmfc_clock_generator_output_t      WhichOutput (select output)
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
   Output:  ccurpmfc_clock_generator_info_t   *CgInfo  (pointer to clock
                                                       generator info)
               __u64                                      M_divider_num
               __u32                                      M_divider_den
               __u64                                      N_divider_num
               __u32                                      N_divider_den
               __u32                                      R_divider_value
               __u32                                      R_divider
               _ccurpmfc_cg_zero_delay_t                  ZeroDelay
                  # CCURPMFC_CG_ZERO_DELAY_MODE
                  # CCURPMFC_CG_NORMAL_MODE
               _ccurpmfc_cg_stat_ctrl_voltsel_t           Voltage_select
                  # CCURPMFC_CG_VOLTAGE_SELECT_1_8V
                  # CCURPMFC_CG_VOLTAGE_SELECT_3_3V
               _ccurpmfc_cg_input_xaxb_extclk_sel_t     Input_xaxb_selection
                  # CCURPMFC_CG_INPUT_XAXB_USE_CRYSTAL
                  # CCURPMFC_CG_INPUT_XAXB_USE_EXTCLK_SOURCE
```

```
_ccurpmfc_cg_xaxb_power_down_t                  Input_xaxb_power
   # CCURPMFC_CG_XAXB_POWER_DOWN
   # CCURPMFC_CG_XAXB_DO_NOT_POWER_DOWN
ccurpmfc_clkgen_csr_t                           Clkcsr
   _ccurpmfc_clkgen_interface_t                 interface
      # CCURPMFC_CLOCK_GENERATOR_INTERFACE_IDLE
      # CCURPMFC_CLOCK_GENERATOR_INTERFACE_BUSY
   _ccurpmfc_clkgen_output_t                    output
      # CCURPMFC_CLOCK_GENERATOR_OUTPUT_DISABLE
      # CCURPMFC_CLOCK_GENERATOR_OUTPUT_ENABLE
   _ccurpmfc_clkgen_state_t                     state
      # CCURPMFC_CLOCK_GENERATOR_ACTIVE
      # CCURPMFC_CLOCK_GENERATOR_RESET
ccurpmfc_clkgen_output_config_t                 Config
   _ccurpmfc_cg_outcfg_force_rdiv2_t            force_rdiv2
      # CCURPMFC_CG_OUTPUT_CONFIG_DONT_FORCE_RDIV2
      # CCURPMFC_CG_OUTPUT_CONFIG_FORCE_RDIV2
   _ccurpmfc_cg_outcfg_enable_t                 enable
      # CCURPMFC_CG_OUTPUT_CONFIG_DISABLE
      # CCURPMFC_CG_OUTPUT_CONFIG_ENABLE
   _ccurpmfc_cg_outcfg_shutdown_t               shutdown
      # CCURPMFC_CG_OUTPUT_CONFIG_POWER_UP
      # CCURPMFC_CG_OUTPUT_CONFIG_SHUTDOWN
ccurpmfc_clkgen_output_format_t                 Format
   _ccurpmfc_cg_outfmt_cmos_drive_t             cmos_drive
      # CCURPMFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_LVDS
      # CCURPMFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_CMOS
   _ccurpmfc_cg_outfmt_disable_state_t          disable_state
      # CCURPMFC_CG_OUTPUT_FORMAT_DISABLE_LOW
      # CCURPMFC_CG_OUTPUT_FORMAT_DISABLE_HIGH
   _ccurpmfc_cg_outfmt_sync_t                   sync
      # CCURPMFC_CG_OUTPUT_FORMAT_SYNC_DISABLE
      # CCURPMFC_CG_OUTPUT_FORMAT_SYNC_ENABLE
   _ccurpmfc_cg_outfmt_format_t                 format
      # CCURPMFC_CG_OUTPUT_FORMAT_FORMAT_LVDS
      # CCURPMFC_CG_OUTPUT_FORMAT_FORMAT_CMOS
ccurpmfc_clkgen_output_mode_t                   Mode
   _ccurpmfc_cg_outmode_amplitude_t             amplitude
      # CCURPMFC_CG_OUTPUT_AMPLITUDE_CMOS
      # CCURPMFC_CG_OUTPUT_AMPLITUDE_LVDS
   _ccurpmfc_cg_outmode_common_t                common
      # CCURPMFC_CG_OUTPUT_COMMON_CMOS
      # CCURPMFC_CG_OUTPUT_COMMON_LVDS
      # CCURPMFC_CG_OUTPUT_COMMON_LVPECL
ccurpmfc_clkgen_output_mux_t                    Mux
   _ccurpmfc_cg_outmux_inversion_t              inversion
      # CCURPMFC_CG_OUTPUT_MUX_COMPLEMENTARY
      # CCURPMFC_CG_OUTPUT_MUX_IN_PHASE
      # CCURPMFC_CG_OUTPUT_MUX_INVERTED
      # CCURPMFC_CG_OUTPUT_MUX_OUT_OF_PHASE
   _ccurpmfc_cg_outmux_ndiv_select_t            ndiv_mux
      # CCURPMFC_CG_OUTPUT_MUX_NDIV_0
      # CCURPMFC_CG_OUTPUT_MUX_NDIV_1
      # CCURPMFC_CG_OUTPUT_MUX_NDIV_2
      # CCURPMFC_CG_OUTPUT_MUX_NDIV_3
      # CCURPMFC_CG_OUTPUT_MUX_NDIV_4
ccurpmfc_clkgen_input_clock_enable_t            Input_clock_enable
   _ccurpmfc_cg_input_clock_enable_t            input_0_clock
      # CCURPMFC_CG_INPUT_CLOCK_DISABLE
      # CCURPMFC_CG_INPUT_CLOCK_ENABLE
   _ccurpmfc_cg_input_clock_enable_t            input_1_clock
```

```
      # CCURPMFC_CG_INPUT_CLOCK_DISABLE
      # CCURPMFC_CG_INPUT_CLOCK_ENABLE
   _ccurpmfc_cg_input_clock_enable_t            input_2_clock
      # CCURPMFC_CG_INPUT_CLOCK_DISABLE
      # CCURPMFC_CG_INPUT_CLOCK_ENABLE
   _ccurpmfc_cg_input_clock_enable_t            input_fb_clock
      # CCURPMFC_CG_INPUT_CLOCK_DISABLE
      # CCURPMFC_CG_INPUT_CLOCK_ENABLE
ccurpmfc_clkgen_input_clock_select_t            Input_clock_select
   _ccurpmfc_cg_input_clock_select_control_t    control
      # CCURPMFC_CG_INPUT_CLOCK_SELECT_PIN_CONTROL
      # CCURPMFC_CG_INPUT_CLOCK_SELECT_REG_CONTROL
   _ccurpmfc_cg_input_clock_select_register_t   select
      # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN0
      # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN1
      # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN2
      # CCURPMFC_CG_INPUT_CLOCK_SELECT_INXAXB
ccurpmfc_pdiv_all_info_t                        Pdiv_info
   __u64                                        Pfb_divider
   ccurpmfc_pdiv_info_t                         P0
      __u64                                     Divider
      _ccurpmfc_cg_pdiv_enable_t                Enable
         # CCURPMFC_CG_PDIV_DISABLE
         # CCURPMFC_CG_PDIV_ENABLE
      _ccurpmfc_cg_pdiv_input_state_t           State
         # CCURPMFC_CG_PDIV_INPUT_UNUSED
         # CCURPMFC_CG_PDIV_INPUT_DISABLED
         # CCURPMFC_CG_PDIV_INPUT_SELECTED
   ccurpmfc_pdiv_info_t                         P1
      __u64                                     Divider
      _ccurpmfc_cg_pdiv_enable_t                Enable
         # CCURPMFC_CG_PDIV_DISABLE
         # CCURPMFC_CG_PDIV_ENABLE
      _ccurpmfc_cg_pdiv_input_state_t           State
         # CCURPMFC_CG_PDIV_INPUT_UNUSED
         # CCURPMFC_CG_PDIV_INPUT_DISABLED
         # CCURPMFC_CG_PDIV_INPUT_SELECTED
   ccurpmfc_pdiv_info_t                         P2
      __u64                                     Divider
      _ccurpmfc_cg_pdiv_enable_t                Enable
         # CCURPMFC_CG_PDIV_DISABLE
         # CCURPMFC_CG_PDIV_ENABLE
      _ccurpmfc_cg_pdiv_input_state_t           State
         # CCURPMFC_CG_PDIV_INPUT_UNUSED
         # CCURPMFC_CG_PDIV_INPUT_DISABLED
         # CCURPMFC_CG_PDIV_INPUT_SELECTED
   ccurpmfc_pdiv_info_t                         Pxaxb
      __u64                                     Divider
      _ccurpmfc_cg_pdiv_enable_t                Enable
         # CCURPMFC_CG_PDIV_DISABLE
         # CCURPMFC_CG_PDIV_ENABLE
      _ccurpmfc_cg_pdiv_input_state_t           State
         # CCURPMFC_CG_PDIV_INPUT_UNUSED
         # CCURPMFC_CG_PDIV_INPUT_DISABLED
         # CCURPMFC_CG_PDIV_INPUT_SELECTED
   int                                  Which_Pdiv_Selected
   int                                  P_Divider
   long double                          OutputClockFrequency;
      # <valid positive output clock frequency>
      # CCURPMFC_CLOCK_ERROR_INVALID_P_DIVIDER
      # CCURPMFC_CLOCK_ERROR_VCO_CLOCK_NOT_IN_RANGE
```

```
                  # CCURPMFC_CLOCK_ERROR_N_DIVIDER_NOT_IN_RANGE
                  # CCURPMFC_CLOCK_ERROR_P_DIVIDER_NOT_IN_RANGE
                  # CCURPMFC_CLOCK_ERROR_R_DIVIDER_NOT_IN_RANGE
                  # CCURPMFC_CLOCK_ERROR_INVALID_CLOCK_FREQUENCY
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
*****************************************************************************/
```

2.2.37 ccurPMFC_Clock_Get_Generator_Input_Clock_Enable()

This call returns the status of all the input clocks.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Input_Clock_Enable (void *Handle,
                  ccurpmfc_clkgen_input_clock_enable_t   *InputClockEnable)

   Description: Return the Clock Generator Input Clock Enable

   Input:   void                                  *Handle        (Handle pointer)
   Output:  ccurpmfc_clkgen_input_clock_enable_t  *InputClockEnable (pointer to
                                                              input clock enable)
               _ccurpmfc_cg_input_clock_enable_t  input_0_clock
                  # CCURPMFC_CG_INPUT_CLOCK_DISABLE
                  # CCURPMFC_CG_INPUT_CLOCK_ENABLE
               _ccurpmfc_cg_input_clock_enable_t  input_1_clock
                  # CCURPMFC_CG_INPUT_CLOCK_DISABLE
                  # CCURPMFC_CG_INPUT_CLOCK_ENABLE
               _ccurpmfc_cg_input_clock_enable_t  input_2_clock
                  # CCURPMFC_CG_INPUT_CLOCK_DISABLE
                  # CCURPMFC_CG_INPUT_CLOCK_ENABLE
               _ccurpmfc_cg_input_clock_enable_t  input_fb_clock
                  # CCURPMFC_CG_INPUT_CLOCK_DISABLE
                  # CCURPMFC_CG_INPUT_CLOCK_ENABLE
   Return: _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE  (Clock is not active)
 *****************************************************************************/
```

2.2.38 ccurPMFC_Clock_Get_Generator_Input_Clock_Select()

This call returns the input clock selection.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Input_Clock_Select (void *Handle,
                             ccurpmfc_clkgen_input_clock_select_t   *ClkSel)

   Description: Get Input Clock Selection

   Input:   void                                    *Handle  (Handle pointer)
   Output:  ccurpmfc_clkgen_input_clock_select_t    *ClkSel  (pointer to
```

```
                                                        input clock selection)
            _ccurpmfc_cg_input_clock_select_control_t    control;
               # CCURPMFC_CG_INPUT_CLOCK_SELECT_PIN_CONTROL
               # CCURPMFC_CG_INPUT_CLOCK_SELECT_REG_CONTROL
            _ccurpmfc_cg_input_clock_select_register_t   select;
               # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN0
               # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN1
               # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN2
               # CCURPMFC_CG_INPUT_CLOCK_SELECT_INXAXB
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (device not open)
               # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION          (local region error)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE      (Clock is not active)
 ***************************************************************************/
```

2.2.39 ccurPMFC_Clock_Get_Generator_Input_Clock_Status()

The call returns the input clock status.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Input_Clock_Status (void *Handle,
                              ccurpmfc_clkgen_input_clock_status_t   *ClkStatus)

   Description: Get Input Clock Status

   Input:   void                                  *Handle    (Handle pointer)
   Output:  ccurpmfc_clkgen_input_clock_status_t  *ClkSatus  (pointer to input
                                                             clock status)
               _ccurpmfc_cg_calibration_status_t            calstat
                  # CCURPMFC_CG_STATUS_DEVICE_IS_NOT_CALIBRATING
                  # CCURPMFC_CG_STATUS_DEVICE_IS_CALIBRATING
               _ccurpmfc_cg_lol_pll_locked_t                PLL_locked
                  # CCURPMFC_CG_STATUS_LOL_PLL_LOCKED
                  # CCURPMFC_CG_STATUS_LOL_PLL_NOT_LOCKED
               _ccurpmfc_cg_smbus_timeout_error_t           SMBUS_timeout
                  # CCURPMFC_CG_STATUS_LOL_SMBUS_NOT_TIMEDOUT
                  # CCURPMFC_CG_STATUS_LOL_SMBUS_TIMEDOUT
               _ccurpmfc_cg_los_signal_present_t            input_signal
                  # CCURPMFC_CG_STATUS_LOS_SIGNAL_PRESENT
                  # CCURPMFC_CG_STATUS_LOS_SIGNAL_NOT_PRESENT
               _ccurpmfc_cg_los_alarm_t                     input_0_clock
                  # CCURPMFC_CG_LOS_INPUT_CLOCK_PRESENT
                  # CCURPMFC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
               _ccurpmfc_cg_los_alarm_t                     input_1_clock
                  # CCURPMFC_CG_LOS_INPUT_CLOCK_PRESENT
                  # CCURPMFC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
               _ccurpmfc_cg_los_alarm_t                     input_2_clock
                  # CCURPMFC_CG_LOS_INPUT_CLOCK_PRESENT
                  # CCURPMFC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
               _ccurpmfc_cg_los_alarm_t                     input_fb_clock
                  # CCURPMFC_CG_LOS_INPUT_CLOCK_PRESENT
                  # CCURPMFC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
               _ccurpmfc_cg_losxaxb_signal_present_t        input_xaxb_clock
                  # CCURPMFC_CG_LOS_INPUT_CLOCK_PRESENT
                  # CCURPMFC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
```

```
            # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN               (device not open)
            # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
            # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
 ************************************************************************/
```

2.2.40 ccurPMFC_Clock_Get_Generator_M_Divider()

This call returns the M-Divider numerator, denominator and value.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_M_Divider (void        *Handle,
                                           __u64       *Numerator,
                                           __u32       *Denominator,
                                           long double *Value)

   Description: Return Clock Generator M-Divider Numerator and Denominator

   Input:   void              *Handle             (Handle pointer)
   Output:  __u64             *Numerator          (pointer to Numerator)
            __u32             *Denominator        (pointer to Denominator)
            long double       *Value              (pointer to Value)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (device not open)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
 ************************************************************************/
```

2.2.41 ccurPMFC_Clock_Get_Generator_N_Divider()

This call returns the N-Divider numerator, denominator and value for the selected divider.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_N_Divider (void                       *Handle,
                              _ccurpmfc_clock_generator_divider_t WhichDivider,
                              __u64                               *Numerator,
                              __u32                               *Denominator,
                              long double                         *Value)

   Description: Return Clock Generator N-Divider Numerator and Denominator

   Input:   void                                *Handle         (Handle pointer)
            _ccurpmfc_clock_generator_divider_t  WhichDivider    (select divider)
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N0
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N1
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N2
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N3
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N4
   Output:  __u64                        *Numerator    (pointer to Numerator)
            __u32                        *Denominator  (pointer to Denominator)
            long double                  *Value        (pointer to Value)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
```

```
            # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
            # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
 ***************************************************************************/
```

2.2.42 ccurPMFC_Clock_Get_Generator_Output_Config()

Return the clock generator output configuration for the selected output.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Output_Config (void               *Handle,
                            _ccurpmfc_clock_generator_output_t  WhichOutput,
                            ccurpmfc_clkgen_output_config_t     *OutCfg)

   Description: Return Clock Generator Output Configuration

   Input:   void                                  *Handle     (Handle pointer)
            _ccurpmfc_clock_generator_output_t    WhichOutput (select output)
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
   Output:  ccurpmfc_clkgen_output_config_t   *OutCfg (pointer to output config)
               _ccurpmfc_cg_outcfg_force_rdiv2_t        force_rdiv2
                  # CCURPMFC_CG_OUTPUT_CONFIG_DONT_FORCE_RDIV2
                  # CCURPMFC_CG_OUTPUT_CONFIG_FORCE_RDIV2
               _ccurpmfc_cg_outcfg_enable_t             enable
                  # CCURPMFC_CG_OUTPUT_CONFIG_DISABLE
                  # CCURPMFC_CG_OUTPUT_CONFIG_ENABLE
               _ccurpmfc_cg_outcfg_shutdown_t           shutdown
                  # CCURPMFC_CG_OUTPUT_CONFIG_POWER_UP
                  # CCURPMFC_CG_OUTPUT_CONFIG_SHUTDOWN
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
 ***************************************************************************/
```

2.2.43 ccurPMFC_Clock_Get_Generator_Output_Format()

Return the clock generator output format for the selected output.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Output_Format (void                *Handle,
                             _ccurpmfc_clock_generator_output_t  WhichOutput,
                             ccurpmfc_clkgen_output_format_t     *OutFmt)

   Description: Return Clock Generator Output Format

   Input:   void                                 *Handle     (Handle pointer)
            _ccurpmfc_clock_generator_output_t   WhichOutput (select output)
```

```
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
   Output:  ccurpmfc_clkgen_output_format_t    *OutFmt (pointer to output format)
               _ccurpmfc_cg_outfmt_cmos_drive_t         cmos_drive
                  # CCURPMFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_LVDS
                  # CCURPMFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_CMOS
               _ccurpmfc_cg_outfmt_disable_state_t      disable_state
                  # CCURPMFC_CG_OUTPUT_FORMAT_DISABLE_LOW
                  # CCURPMFC_CG_OUTPUT_FORMAT_DISABLE_HIGH
               _ccurpmfc_cg_outfmt_sync_t               sync
                  # CCURPMFC_CG_OUTPUT_FORMAT_SYNC_DISABLE
                  # CCURPMFC_CG_OUTPUT_FORMAT_SYNC_ENABLE
               _ccurpmfc_cg_outfmt_format_t             format
                  # CCURPMFC_CG_OUTPUT_FORMAT_FORMAT_LVDS
                  # CCURPMFC_CG_OUTPUT_FORMAT_FORMAT_CMOS
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
 ***************************************************************************/
```

2.2.44 ccurPMFC_Clock_Get_Generator_Output_Mode()

Return the clock generator output mode for the selected output.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Output_Mode (void               *Handle,
                        _ccurpmfc_clock_generator_output_t    WhichOutput,
                        ccurpmfc_clkgen_output_mode_t         *OutMode)

   Description: Return Clock Generator Output Mode

   Input:   void                                    *Handle     (Handle pointer)
            _ccurpmfc_clock_generator_output_t      WhichOutput (select output)
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
   Output:  ccurpmfc_clkgen_output_mode_t        *OutMode (pointer to output
                                                           amplitude/common mode)
               _ccurpmfc_cg_outmode_amplitude_t         amplitude
                  # CCURPMFC_CG_OUTPUT_AMPLITUDE_CMOS
                  # CCURPMFC_CG_OUTPUT_AMPLITUDE_LVDS
```

```
             _ccurpmfc_cg_outmode_common_t            common
                # CCURPMFC_CG_OUTPUT_COMMON_CMOS
                # CCURPMFC_CG_OUTPUT_COMMON_LVDS
                # CCURPMFC_CG_OUTPUT_COMMON_LVPECL
   Return:  _ccurpmfc_lib_error_number_t
              # CCURPMFC_LIB_NO_ERROR                 (successful)
              # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
              # CCURPMFC_LIB_NOT_OPEN                 (device not open)
              # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
              # CCURPMFC_LIB_NO_LOCAL_REGION          (local region not present)
              # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE      (Clock is not active)
 ***************************************************************************/
```

2.2.45 ccurPMFC_Clock_Get_Generator_Output_Mux()

Return the clock generator output mux for the selected output.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Output_Mux (void                  *Handle,
                             _ccurpmfc_clock_generator_output_t WhichOutput,
                             ccurpmfc_clkgen_output_mux_t       *OutMux)

   Description: Return Clock Generator Output Mux

   Input:   void                                   *Handle     (Handle pointer)
            _ccurpmfc_clock_generator_output_t     WhichOutput (select output)
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
   Output:  ccurpmfc_clkgen_output_mux_t        *OutMux (pointer to output
                                                         inversion/N-divider mux)
              _ccurpmfc_cg_outmux_inversion_t       inversion
                 # CCURPMFC_CG_OUTPUT_MUX_COMPLEMENTARY
                 # CCURPMFC_CG_OUTPUT_MUX_IN_PHASE
                 # CCURPMFC_CG_OUTPUT_MUX_INVERTED
                 # CCURPMFC_CG_OUTPUT_MUX_OUT_OF_PHASE
              _ccurpmfc_cg_outmux_ndiv_select_t     ndiv_mux
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_0
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_1
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_2
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_3
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_4
   Return:  _ccurpmfc_lib_error_number_t
              # CCURPMFC_LIB_NO_ERROR               (successful)
              # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
              # CCURPMFC_LIB_NOT_OPEN               (device not open)
              # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
              # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
              # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
 ***************************************************************************/
```

2.2.46 ccurPMFC_Clock_Get_Generator_P_Divider()

Return the clock generator P-Divider.

```
/*****************************************************************************
  _ccurpmfc_lib_error_number_t
  ccurPMFC_Clock_Get_Generator_P_Divider (void                        *Handle,
                              _ccurpmfc_clock_generator_divider_t WhichDivider,
                              __u64                               *Divider)

   Description: Return Clock Generator P-Divider

   Input:   void                                  *Handle      (Handle pointer)
            _ccurpmfc_clock_generator_divider_t   WhichDivider (select divider)
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P0
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P1
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P2
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_PFB
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_PXAXB
   Output:  __u64                                 *Divider    (pointer to
                                                               Divider)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR           (successful)
               # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN           (device not open)
               # CCURPMFC_LIB_INVALID_ARG        (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION    (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
 *****************************************************************************/
```

2.2.47 ccurPMFC_Clock_Get_Generator_P_Divider_Enable()

Return the clock generator P-Divider Enable state.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_P_Divider_Enable (void             *Handle,
                              _ccurpmfc_clock_generator_divider_t  WhichDivider,
                              _ccurpmfc_cg_pdiv_enable_t           *Pdiv_Enable)

   Description: Return Clock Generator P-Divider Enable

   Input:   void                                  *Handle      (Handle pointer)
            _ccurpmfc_clock_generator_divider_t   WhichDivider (select divider)
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P0
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P1
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P2
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_PXAXB
   Output:  _ccurpmfc_cg_pdiv_enable_t            *Pdiv_Enable (pointer to enable
                                                                flag)
               # CCURPMFC_CG_PDIV_DISABLE
               # CCURPMFC_CG_PDIV_ENABLE
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR           (successful)
               # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN           (device not open)
               # CCURPMFC_LIB_INVALID_ARG        (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION    (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
 *****************************************************************************/
```

2.2.48 ccurPMFC_Clock_Get_Generator_R_Divider()

Return the clock generator R-Divider for the selected divider.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_R_Divider (void                 *Handle,
                           _ccurpmfc_clock_generator_divider_t WhichDivider,
                           __u32                               *Divider)

   Description: Return Clock Generator R-Divider

   Input:   void                                  *Handle      (Handle pointer)
            _ccurpmfc_clock_generator_divider_t   WhichDivider (select divider)
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R0
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R1
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R2
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R3
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R4
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R5
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R6
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R7
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R8
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R9
   Output:  __u32                                 *Divider   (pointer to Divider)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (device not open)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
 *****************************************************************************/
```

2.2.49 ccurPMFC_Clock_Get_Generator_Revision()

Return the clock generator revision information.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Revision (void                      *Handle,
                                          ccurpmfc_clock_revision_t *Revision)

   Description: Return Clock Generator Revision

   Input:   void                                  *Handle      (Handle pointer)
   Output:  ccurpmfc_clock_revision_t             *Revision    (pointer to Divider)
             _ccurpmfc_cg_die_revision_t           DieRevision
               # CCURPMFC_CG_SILICON_REVISION_A0
               # CCURPMFC_CG_SILICON_REVISION_A1
             _ccurpmfc_convert_base_part_number_t BasePartNumber;
               u_short BPN
               u_char  NChar[2]
             _ccurpmfc_cg_clock_speed_grade_t     ClockSpeedGrade;
               # CCURPMFC_CG_CLOCK_SPEED_GRADE_A
               # CCURPMFC_CG_CLOCK_SPEED_GRADE_B
               # CCURPMFC_CG_CLOCK_SPEED_GRADE_C
               # CCURPMFC_CG_CLOCK_SPEED_GRADE_D
             _ccurpmfc_cg_clock_revision_t        ClockRevision;
               # CCURPMFC_CG_CLOCK_REVISION_A
               # CCURPMFC_CG_CLOCK_REVISION_B
               # CCURPMFC_CG_CLOCK_REVISION_C
               # CCURPMFC_CG_CLOCK_REVISION_D
   Return:  _ccurpmfc_lib_error_number_t
```

```
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (device not open)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*************************************************************************/
```

2.2.50 ccurPMFC_Clock_Get_Generator_Value()

This is a generic call that can return the value of a valid clock generator address.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Value (void    *Handle,
                                       int     address,
                                       u_char  *value)

   Description: Return the value of the specified Clock Generator register.

   Input:   void              *Handle             (Handle pointer)
            int               address             (clock gen address to display)
   Output:  u_char            *value;             (pointer to value)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (device not open)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
 *************************************************************************/
```

2.2.51 ccurPMFC_Clock_Get_Generator_Voltage_Select()

Return the clock generator Voltage Selection.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Get_Generator_Voltage_Select (void                  *Handle,
                                _ccurpmfc_cg_stat_ctrl_voltsel_t   *VoltSel)

   Description: Return the Clock Generator Voltage Selection

   Input:   void                            *Handle  (Handle pointer)
   Output:  _ccurpmfc_cg_stat_ctrl_voltsel_t *VoltSel (pointer to voltage select)
               # CCURPMFC_CG_VOLTAGE_SELECT_1_8V
               # CCURPMFC_CG_VOLTAGE_SELECT_3_3V
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
 *************************************************************************/
```

2.2.52 ccurPMFC_Clock_Get_Generator_Zero_Delay()

Return the clock generator Zero Delay status.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
```

```
   ccurPMFC_Clock_Get_Generator_Zero_Delay (void                    *Handle,
                                           _ccurpmfc_cg_zero_delay_t  *ZeroDelay)

   Description: Return the Clock Generator Zero Delay setting.

   Input:   void                        *Handle     (Handle pointer)
   Output:  _ccurpmfc_cg_zero_delay_t   *ZeroDelay  (pointer to zero delay)
               # CCURPMFC_CG_ZERO_DELAY_MODE
               # CCURPMFC_CG_NORMAL_MODE
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (device not open)
               # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
 ***************************************************************************/
```

2.2.53 ccurPMFC_Clock_ReturnOutputFrequency()

This call does not return the actual programmed frequency but instead returns the expected output frequency that would be generated if the specified user input parameters are supplied.

```
/***************************************************************************
   long double
   ccurPMFC_Clock_ReturnOutputFrequency(double       InputClock,
                                        long double  Mdiv_value,
                                        long double  Ndiv_value,
                                        double       Pdiv_value,
                                        double       Rdiv_value)

   Description: Return output frequency

   Input:   double             InputClock  (input clock frequency in Hz)
            long double        Mdiv_value  (M-Divider value)
            long double        Ndiv_value  (N-Divider value)
            double             Pdiv_value  (P-Divider value)
            double             Rdiv_value  (R-Divider value)
   Output:  none
   Return:  long double        returned frequency
 ***************************************************************************/
```

2.2.54 ccurPMFC_Clock_Set_Generator_CSR()

This call sets the clock generator control and status register.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_CSR (void                      *Handle,
                                     ccurpmfc_clkgen_csr_t     *CgCsr)

   Description: Set Clock Generator Control and Status information

   Input:   void                   *Handle      (Handle pointer)
            ccurpmfc_clkgen_csr_t  *CgCsr       (pointer to clock generator csr)
               _ccurpmfc_clkgen_output_t                    output
                  # CCURPMFC_CLOCK_GENERATOR_OUTPUT_DISABLE
                  # CCURPMFC_CLOCK_GENERATOR_OUTPUT_ENABLE
                  # CCURPMFC_CLOCK_GENERATOR_OUTPUT_DO_NOT_CHANGE
               _ccurpmfc_clkgen_state_t                     state
                  # CCURPMFC_CLOCK_GENERATOR_ACTIVE
```

```
                # CCURPMFC_CLOCK_GENERATOR_RESET
                # CCURPMFC_CLOCK_GENERATOR_STATE_DO_NOT_CHANGE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
 ***************************************************************************/
```

2.2.55 ccurPMFC_Clock_Set_Generator_Input_Clock_Enable()

This call sets the input clock status for the input clocks. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_Input_Clock_Enable (void       *Handle,
                       ccurpmfc_clkgen_input_clock_enable_t   *InputClockEnable)

   Description: Set Clock Generator Input Clock Enable

   Input:   void                                  *Handle            (Handle
                                                                      pointer)
            ccurpmfc_clkgen_input_clock_enable_t *InputClockEnable  (pointer to
                                                                 input clock enable)
               _ccurpmfc_cg_input_clock_enable_t    input_0_clock
                  # CCURPMFC_CG_INPUT_CLOCK_DISABLE
                  # CCURPMFC_CG_INPUT_CLOCK_ENABLE
                  # CCURPMFC_CG_INPUT_CLOCK_DO_NOT_CHANGE
               _ccurpmfc_cg_input_clock_enable_t    input_1_clock
                  # CCURPMFC_CG_INPUT_CLOCK_DISABLE
                  # CCURPMFC_CG_INPUT_CLOCK_ENABLE
                  # CCURPMFC_CG_INPUT_CLOCK_DO_NOT_CHANGE
               _ccurpmfc_cg_input_clock_enable_t    input_2_clock
                  # CCURPMFC_CG_INPUT_CLOCK_DISABLE
                  # CCURPMFC_CG_INPUT_CLOCK_ENABLE
                  # CCURPMFC_CG_INPUT_CLOCK_DO_NOT_CHANGE
               _ccurpmfc_cg_input_clock_enable_t    input_fb_clock
                  # CCURPMFC_CG_INPUT_CLOCK_DISABLE
                  # CCURPMFC_CG_INPUT_CLOCK_ENABLE
                  # CCURPMFC_CG_INPUT_CLOCK_DO_NOT_CHANGE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
 ***************************************************************************/
```

2.2.56 ccurPMFC_Clock_Set_Generator_Input_Clock_Select()

This call sets the input clock selection. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/***************************************************************************
```

```
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_Input_Clock_Select (void          *Handle,
                           ccurpmfc_clkgen_input_clock_select_t   *ClkSel)

   Description: Set Clock Generator Input Clock Selection

   Input:   void                                    *Handle (Handle pointer)
            ccurpmfc_clkgen_input_clock_select_t    *ClkSel (pointer to input
                                                             clock select)
               _ccurpmfc_cg_input_clock_select_control_t    control;
                  # CCURPMFC_CG_INPUT_CLOCK_SELECT_PIN_CONTROL
                  # CCURPMFC_CG_INPUT_CLOCK_SELECT_REG_CONTROL
                  # CCURPMFC_CG_INPUT_CLOCK_SELECT_CONTROL_DO_NOT_CHANGE
               _ccurpmfc_cg_input_clock_select_register_t   select;
                  # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN0
                  # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN1
                  # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN2
                  # CCURPMFC_CG_INPUT_CLOCK_SELECT_INXAXB
                  # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN_DO_NOT_CHANGE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
 ***************************************************************************/
```

2.2.57 ccurPMFC_Clock_Set_Generator_M_Divider()

This call sets the clock generator M-Divider to the user specified Numerator and Denominator. If the Update flag is set, then the change will take place after the divider has been written to. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_M_Divider (void    *Handle,
                                           __u64   Numerator,
                                           __u32   Denominator,
                                           int     Update)

   Description: Set Clock Generator M-Divider Numerator and Denominator

   Input:   void              *Handle             (Handle pointer)
            __u64             Numerator           (Numerator)
            __u32             Denominator         (Denominator)
            int               Update              (True=Update)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (device not open)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
 ***************************************************************************/
```

2.2.58 ccurPMFC_Clock_Set_Generator_N_Divider()

This call sets the clock generator selected N-Divider to the user specified Numerator and Denominator. If the Update flag is set, then the change will take place after the divider has been written to. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/*****************************************************************************
   ccurPMFC_Clock_Set_Generator_N_Divider()
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_N_Divider (void            *Handle,
                   _ccurpmfc_clock_generator_divider_t     WhichDivider,
                   __u64                                   Numerator,
                   __u32                                   Denominator,
                   int                                     Update)

   Description: Set Clock Generator N-Divider Numerator and Denominator

   Input:   void                                *Handle         (Handle pointer)
            _ccurpmfc_clock_generator_divider_t  WhichDivider    (select divider)
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N0
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N1
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N2
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N3
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N4
            __u64                                Numerator       (Numerator)
            __u32                                Denominator     (Denominator)
            int                                  Update          (True=Update)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
 *****************************************************************************/
```

2.2.59 ccurPMFC_Clock_Set_Generator_Output_Config()

This call sets the clock generator Output Configuration for the selected output. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_Output_Config (void              *Handle,
                           _ccurpmfc_clock_generator_output_t  WhichOutput,
                           ccurpmfc_clkgen_output_config_t     *OutCfg)

   Description: Set Clock Generator Output Configuration

   Input:   void                                *Handle         (Handle pointer)
            _ccurpmfc_clock_generator_output_t  WhichOutput     (select output)
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
```

```
            # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
            # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
            # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
         ccurpmfc_clkgen_output_config_t   *OutCfg  (pointer to output config)
            _ccurpmfc_cg_outcfg_force_rdiv2_t        force_rdiv2
               # CCURPMFC_CG_OUTPUT_CONFIG_DONT_FORCE_RDIV2
               # CCURPMFC_CG_OUTPUT_CONFIG_FORCE_RDIV2
               # CCURPMFC_CG_OUTPUT_CONFIG_FORCE_DO_NOT_CHANGE
            _ccurpmfc_cg_outcfg_enable_t             enable
               # CCURPMFC_CG_OUTPUT_CONFIG_DISABLE
               # CCURPMFC_CG_OUTPUT_CONFIG_ENABLE
               # CCURPMFC_CG_OUTPUT_CONFIG_ENABLE_DO_NOT_CHANGE
            _ccurpmfc_cg_outcfg_shutdown_t           shutdown
               # CCURPMFC_CG_OUTPUT_CONFIG_POWER_UP
               # CCURPMFC_CG_OUTPUT_CONFIG_SHUTDOWN
               # CCURPMFC_CG_OUTPUT_CONFIG_SHUTDOWN_DO_NOT_CHANGE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
 ***************************************************************************/
```

2.2.60 ccurPMFC_Clock_Set_Generator_Output_Format()

This call sets the clock generator Output Format for the selected output. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_Output_Format (void                  *Handle,
                              _ccurpmfc_clock_generator_output_t  WhichOutput,
                              ccurpmfc_clkgen_output_format_t     *OutFmt)

   Description: Set Clock Generator Output Format

   Input:   void                                    *Handle     (Handle pointer)
            _ccurpmfc_clock_generator_output_t      WhichOutput (select output)
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
            ccurpmfc_clkgen_output_format_t         *OutFmt     (pointer to
                                                                 output format)
               _ccurpmfc_cg_outfmt_cmos_drive_t         cmos_drive
                  # CCURPMFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_LVDS
                  # CCURPMFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_CMOS
                  # CCURPMFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_DO_NOT_CHANGE
               _ccurpmfc_cg_outfmt_disable_state_t      disable_state
                  # CCURPMFC_CG_OUTPUT_FORMAT_DISABLE_LOW
                  # CCURPMFC_CG_OUTPUT_FORMAT_DISABLE_HIGH
```

```
               # CCURPMFC_CG_OUTPUT_FORMAT_DISABLE_DO_NOT_CHANGE
            _ccurpmfc_cg_outfmt_sync_t              sync
               # CCURPMFC_CG_OUTPUT_FORMAT_SYNC_DISABLE
               # CCURPMFC_CG_OUTPUT_FORMAT_SYNC_ENABLE
               # CCURPMFC_CG_OUTPUT_FORMAT_SYNC_DO_NOT_CHANGE
            _ccurpmfc_cg_outfmt_format_t            format
               # CCURPMFC_CG_OUTPUT_FORMAT_FORMAT_LVDS
               # CCURPMFC_CG_OUTPUT_FORMAT_FORMAT_CMOS
               # CCURPMFC_CG_OUTPUT_FORMAT_FORMAT_DO_NOT_CHANGE
      Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
 ***************************************************************************/
```

2.2.61 ccurPMFC_Clock_Set_Generator_Output_Mode()

This call sets the clock generator Output Mode for the selected output. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_Output_Mode (void                  *Handle,
                           _ccurpmfc_clock_generator_output_t     WhichOutput,
                           ccurpmfc_clkgen_output_mode_t          *OutMode)

   Description: Set Clock Generator Output Mode

   Input:   void                                   *Handle     (Handle pointer)
            _ccurpmfc_clock_generator_output_t     WhichOutput (select output)
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
            ccurpmfc_clkgen_output_mode_t          *OutMode    (pointer to
                                                                output mode)
               _ccurpmfc_cg_outmode_amplitude_t         amplitude
                  # CCURPMFC_CG_OUTPUT_AMPLITUDE_CMOS
                  # CCURPMFC_CG_OUTPUT_AMPLITUDE_LVDS
                  # CCURPMFC_CG_OUTPUT_AMPLITUDE_DO_NOT_CHANGE
               _ccurpmfc_cg_outmode_common_t            common
                  # CCURPMFC_CG_OUTPUT_COMMON_CMOS
                  # CCURPMFC_CG_OUTPUT_COMMON_LVDS
                  # CCURPMFC_CG_OUTPUT_COMMON_LVPECL
                  # CCURPMFC_CG_OUTPUT_COMMON_DO_NOT_CHANGE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
```

```
            # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION          (local region not present)
            # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE      (Clock is not active)
 ************************************************************************/
```

2.2.62 ccurPMFC_Clock_Set_Generator_Output_Mux()

This call sets the clock generator Output Mux for the selected output. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_Output_Mux (void                  *Handle,
                         _ccurpmfc_clock_generator_output_t     WhichOutput,
                         ccurpmfc_clkgen_output_mux_t           *OutMux)

   Description: Set Clock Generator Output Mux

   Input:   void                                    *Handle     (Handle pointer)
            _ccurpmfc_clock_generator_output_t      WhichOutput (select output)
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
               # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
            ccurpmfc_clkgen_output_mux_t       *OutMux  (pointer to output
                                                         inversion/N-divider mux)
              _ccurpmfc_cg_outmux_inversion_t       inversion
                 # CCURPMFC_CG_OUTPUT_MUX_COMPLEMENTARY
                 # CCURPMFC_CG_OUTPUT_MUX_IN_PHASE
                 # CCURPMFC_CG_OUTPUT_MUX_INVERTED
                 # CCURPMFC_CG_OUTPUT_MUX_OUT_OF_PHASE
                 # CCURPMFC_CG_OUTPUT_MUX_INVERSION_DO_NOT_CHANGE
              _ccurpmfc_cg_outmux_ndiv_select_t     ndiv_mux
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_0
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_1
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_2
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_3
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_4
                 # CCURPMFC_CG_OUTPUT_MUX_NDIV_DO_NOT_CHANGE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (device not open)
               # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION          (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE      (Clock is not active)
 ************************************************************************/
```

2.2.63 ccurPMFC_Clock_Set_Generator_P_Divider()

This call sets the clock generator selected P-Divider to the user specified value. If the Update flag is set, then the change will take place after the divider has been written to. *Normally, this call should not be used. It is*

recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_P_Divider (void                    *Handle,
                         _ccurpmfc_clock_generator_divider_t      WhichDivider,
                         __u64                                    Divider,
                         int                                      Update)

   Description: Set Clock Generator R-Divider

   Input:   void                                  *Handle       (Handle pointer)
            _ccurpmfc_clock_generator_divider_t   WhichDivider  (select divider)
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P0
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P1
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P2
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_PFB
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_PXAXB
            __u64                      Divider         (Divider)
            int                        Update          (True=Update)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (device not open)
               # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION          (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE      (Clock is not active)
 ************************************************************************/
```

2.2.64 ccurPMFC_Clock_Set_Generator_P_Divider_Enable()

This call sets the state of the clock generator P-Divider. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_P_Divider_Enable (void            *Handle,
                          _ccurpmfc_clock_generator_divider_t   WhichDivider,
                          _ccurpmfc_cg_pdiv_enable_t            Pdiv_Enable)

   Description: Set Clock Generator P-Divider Enable

   Input:   void                                  *Handle         (Handle pointer)
            _ccurpmfc_clock_generator_divider_t   WhichDivider    (select divider)
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P0
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P1
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P2
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_PXAXB
            _ccurpmfc_cg_pdiv_enable_t            Pdiv_Enable     (enable flag)
               # CCURPMFC_CG_PDIV_DISABLE
               # CCURPMFC_CG_PDIV_ENABLE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (device not open)
               # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION          (local region not present)
```

```
            # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE      (Clock is not active)
 ***************************************************************************/
```

2.2.65 ccurPMFC_Clock_Set_Generator_R_Divider()

This call sets the clock generator selected R-Divider to the user specified value. If the output clock is running, the new clock frequency will take affect immediately or on the next clock cycle depending on the output configuration. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_R_Divider (void                    *Handle,
                        _ccurpmfc_clock_generator_divider_t     WhichDivider,
                        __u32                                   Divider)

   Description: Set Clock Generator R-Divider

   Input:   void                                *Handle        (Handle pointer)
            _ccurpmfc_clock_generator_divider_t  WhichDivider   (select divider)
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R0
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R1
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R2
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R3
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R4
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R5
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R6
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R7
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R8
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_R9
            __u32                                 Divider         (Divider)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
 ***************************************************************************/
```

2.2.66 ccurPMFC_Clock_Set_Generator_Value()

This is a generic call that can program a valid clock generator address to a desired value. User must be intimately familiar with the hardware before programming the values. In-correct programming could result in unpredictable results. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_Value (void    *Handle,
                                       int     address,
                                       u_char  value)

   Description: Set the value of the specified Clock Generator register.

   Input:   void             *Handle             (Handle pointer)
            int              address             (clock gen address to set)
            u_char           value;              (value to write)
```

```
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR           (successful)
               # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN           (device not open)
               # CCURPMFC_LIB_INVALID_ARG        (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION    (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
 ***************************************************************************/
```

2.2.67 ccurPMFC_Clock_Set_Generator_Voltage_Select()

Program the clock generator voltage selection. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_Voltage_Select (void              *Handle,
                               _ccurpmfc_cg_stat_ctrl_voltsel_t   VoltSel)

   Description: Set Clock Generator voltage selection

   Input:   void                              *Handle   (Handle pointer)
            _ccurpmfc_cg_stat_ctrl_voltsel_t VoltSel   (voltage selection)
               # CCURPMFC_CG_VOLTAGE_SELECT_1_8V
               # CCURPMFC_CG_VOLTAGE_SELECT_3_3V
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)

 ***************************************************************************/
```

2.2.68 ccurPMFC_Clock_Set_Generator_Zero_Delay()

Program the clock generator zero delay. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Clock_Set_Generator_Zero_Delay (void                    *Handle,
                                      _ccurpmfc_cg_zero_delay_t  ZeroDelay)

   Description: Set Clock Generator Zero Delay selection

   Input:   void                             *Handle    (Handle pointer)
            _ccurpmfc_cg_zero_delay_t        ZeroDelay  (zero delay selection)
               # CCURPMFC_CG_ZERO_DELAY_MODE
               # CCURPMFC_CG_NORMAL_MODE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
```

```
            # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
            # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
 ***************************************************************************/
```

2.2.69 ccurPMFC_Close()

This call is used to close an already opened device using the *ccurPMFC_Open()* call.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t ccurPMFC_Close(void *Handle)

   Description: Close a previously opened device.

   Input:   void *Handle                   (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR      (successful)
               # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN      (device not open)
 ***************************************************************************/
```

2.2.70 ccurPMFC_Compute_All_Output_Clocks()

This call does not program the clock outputs but instead returns to the user whether the board can be programmed with the user selected output clock frequencies. Additionally, useful information is returned to the user in a structure for each clock that was computed.

```
/***************************************************************************

   ccurPMFC_Compute_All_Output_Clocks()

   Description: Compute All Output Clocks

   Input:   void                        *Handle             (Handle pointer)
            double                      InputClockFrequency (Input clock
                                                             frequency)
            ccurpmfc_compute_all_output_clocks_t *AllClocks (Pointer to all
                                                             output clocks info)
               ccurpmfc_compute_single_output_clock_t       *Clock
                  long double                               DesiredFrequency
                  double                                    DesiredTolerancePPT
   Output:  ccurpmfc_compute_all_output_clocks_t            *AllClocks
               (Pointer to returned output clocks info)
               __u32                                        NumberOfNdividers
               ccurpmfc_compute_single_output_clock_t       *Clock
                  _ccurpmfc_clock_generator_output_t        OutputClock
                     # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
                     # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
                     # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
                     # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
                     # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
                     # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
                     # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
                     # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
                     # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
                     # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
                  double                                    InputClockFrequency
                  long double                               FrequencyDeviation
                  int                                       FrequencyFound
                  long double                               ActualFrequency
                  double                                    ActualTolerancePPT
```

```
                __u64                                       Mdiv_Numerator
                __u32                                       Mdiv_Denominator
                __u64                                       Ndiv_Numerator
                __u32                                       Ndiv_Denominator
                _ccurpmfc_cg_outmux_ndiv_select_t           Ndiv_ToUse
                   # CCURPMFC_CG_OUTPUT_MUX_NDIV_0
                   # CCURPMFC_CG_OUTPUT_MUX_NDIV_1
                   # CCURPMFC_CG_OUTPUT_MUX_NDIV_2
                   # CCURPMFC_CG_OUTPUT_MUX_NDIV_3
                   # CCURPMFC_CG_OUTPUT_MUX_NDIV_4
                __u32                                       Rdiv_value
                __u32                                       Rdivider
                __u32                                       Pdivider
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                     (successful)
               # CCURPMFC_LIB_BAD_HANDLE                   (no/bad handler
                                                            supplied)
               # CCURPMFC_LIB_NOT_OPEN                     (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION              (local region error)
               # CCURPMFC_LIB_IO_ERROR                     (device not ready)
               # CCURPMFC_LIB_N_DIVIDERS_EXCEEDED          (number of N-Dividers
                                                            exceeded)
               # CCURPMFC_LIB_CANNOT_COMPUTE_OUTPUT_FREQ   (cannot compute
                                                            output freq)
               # CCURPMFC_LIB_INVALID_ARG                  (invalid argument)
 ***************************************************************************/
```

2.2.71 ccurPMFC_Convert_Physmem2avmm_Address()

This call is used to supply the user with an Avalon equivalent Address for the supplied Physical DMA memory. This Avalon equivalent address can then be supplied to the DMA engine to perform DMA operations.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Convert_Physmem2avmm_Address(void        *Handle,
                                         uint        *PhysDmaMemPtr,
                                         uint        *AvalonAddress)

   Description: Get the converted value of Physical DMA memory to Avalon address
                to be supplied as address for DMA operations.

   Input:   void                    *Handle           (Handle pointer)
            uint                    *PhysDmaMemPtr    (pointer to physical DMA
                                                       memory
   Output:  uint                    *AvalonAddress    (pointer to Avalon
                                                       Address).
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (library not open)
               # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
               # CCURPMFC_LIB_AVALON_TRANSLATION_TABLE (avalon translation table
                                                       error)
               # CCURPMFC_LIB_ADDRESS_RANGE_ERROR      (address range error)
 ***************************************************************************/
```

2.2.72 ccurPMFC_Create_UserDioCosInterruptHandler()

This call provides the ability for a user to get notification when a DIO change-of-state interrupt occurs. Prior to invoking this call, the user needs to create an *interrupt callback* function which is supplied to this call as

one of its inputs. Additionally, the user selects a set of DIO COS wakeup masks to enter the user supplied callback when a corresponding interrupt occurs. On successful completion of this call, a real-time high priority thread is created and blocked waiting for DIO COS interrupts. When a DIO COS interrupt occurs, the driver will wake up this thread which in turn execute the user supplied *interrupt callback* function. Various DIO COS statistics will be returned to the user as an argument *driver_dio_cos_int* supplied to the *interrupt callback* routine everytime a wakeup occurs. The user needs to ensure that the processing within this *interrupt callback* should be completed in as short a time as possible for the thread to be ready in time to accept the next DIO COS interrupt. Failure to do so will result in missed change-of-state interrupts.

If the interrupt handler has already been created for a device, then the user will be unable to create another one as only one interrupt handler is assigned to each device. User will need to destroy the interrupt handler with the *ccurPMFC_Destroy_UserDioCosInterruptHandler()* call prior to creating a new one.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Create_UserDioCosInterruptHandler(void     *Handle,
                                              void     CallBack(),
                                              u_int    DioCosWakeupInterruptMask)

   Description: Create a User DIO COS Interrupt Handler

   Input:   void            *Handle                     (Handle pointer)
            void            CallBack()                  (user callback function)
            u_int           DioCosWakeupInterruptMask   (wakeup interrupt mask)
               # CCURPMFC_DIO_GROUP0_INTMASK
               # CCURPMFC_DIO_GROUP1_INTMASK
               # CCURPMFC_DIO_GROUP2_INTMASK
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (device not open)
               # CCURPMFC_LIB_INTHDLR_CREATE_FAILURE   (failed to create
                                                        interrupt handler)
               # CCURPMFC_LIB_INTHDLR_ALREADY_RUNNING  (interrupt hdlr already
                                                        running)
               # CCURPMFC_LIB_IOCTL_FAILED             (ioctl failed)
               # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
 *************************************************************************/
```

// User interrupt callback()

```
void DioCosUserCallback(void                          *Handle,
                        ccurpmfc_driver_dio_cos_int_t *driver_dio_cos_int)
{
    // User supplied code for handling interrupt
}

// Interrupt Counters
typedef struct
{
    long long unsigned  dio_interrupt_count;
    long long unsigned  dma_count[CCURPMFC_DMA_MAX_ENGINES];
    long long unsigned  DIO_COS_ChannelsCount[CCURPMFC_DIO_MAX_REGISTERS];
    long long unsigned  DIO_COS_ChannelsOverflowCount[CCURPMFC_DIO_MAX_REGISTERS];
} ccurpmfc_interrupt_dio_cos_counters_t;

typedef struct {
    u_int32_t   chan_00_31;
    u_int32_t   chan_32_63;
```

```
    u_int32_t   chan_64_95;
} ccurpmfc_dio_channel_t;

typedef u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS];

// DIO COS Interrupt
typedef struct
{
    union {
        uint           status;                 // Obsolete
        uint           InterruptsOccurredMask;  // use this name
    };
    union {
        uint           mask;                   // Obsolete
        uint           WakeupInterruptMask;     // use this name
    };

    // DIO information
    union {
        ccurpmfc_dio_channels_t DIO_COS_ChannelsStatus;
        ccurpmfc_dio_channel_t  DIO_COS_ChannelsStatusX;
    };
    union {
        ccurpmfc_dio_channels_t DIO_COS_ChannelsOverflow;
        ccurpmfc_dio_channel_t  DIO_COS_ChannelsOverflowX;
    };

    ccurpmfc_interrupt_dio_cos_counters_t   counters;
} ccurpmfc_driver_dio_cos_int_t;
```

2.2.73 ccurPMFC_Create_UserProcess()

Typically reads from h/w take a finite time to complete. If the user has a process that is time critical and needs to read the latest data faster, they may use a new approach called Hyper-Drive. In this case, the user defines a thread with this call, which continuously reads the data from the board and holds the latest values. The user process can then access this latest data at substantially faster rates. The two drawbacks to this approach is that the excessive bus assess is made and dedicated CPUs are required.

This call is used to create this User Process looping thread which can be controlled by the user via the returned handle. *(This is an experimental API for debugging and testing).*

```
/*************************************************************************
  _ccurpmfc_lib_error_number_t
  ccurPMFC_Create_UserProcess(void                        *Handle,
                              _ccurpmfc_UserFunction_t    *UFunc,
                              _ccurpmfc_UserFunction_t    **UFuncHandle)

   Description: Create a User Process for user defined processing

   Input:   void                       *Handle         (Handle pointer)
            _ccurpmfc_UserFunction_t   *UFunc          (pointer to user
                                                        information structure)
   Output:  _ccurpmfc_UserFunction_t   **UFuncHandle   (pointer to user function
                                                        struct handle)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (device not open)
               # CCURPMFC_LIB_NO_RESOURCE              (cannot allocate memory)
               # CCURPMFC_LIB_INTERNAL_ERROR           (pthread attr failed)
```

```
              # CCURPMFC_LIB_THREAD_CREATE_FAILED      (failed to create thread)
*****************************************************************************/

typedef struct
{
    int Magic;
    void                            (*UserFunction) (void *hdl);
    pthread_t                       UserFunction_Thread_id;
    pid_t                           Pid;
    pthread_mutex_t                 lock;               /* lock this structure */
    pthread_cond_t                  wait;               /* wait for command */
    pthread_mutex_t                 cmd_lock;           /* lock this structure */
    pthread_cond_t                  cmd_wait;           /* wait for command */
    pthread_mutex_t                 user_lock;          /* lock this structure */
    pthread_cond_t                  user_wait;          /* wait for command */
    pthread_mutex_t                 user_mem_lock;      /* lock this structure */
    pthread_cond_t                  user_mem_wait;      /* wait for command */
    volatile int                    cpuAffinity;        /* CPU on which Thread
                                                           will run */
    volatile int                    cpuCount;           /* no. of cpus to run on
                                                           starting at base */
    volatile void                   *Handle;
    volatile void                   **Args;
    volatile int                    SchedulePolicy;
    volatile int                    SchedulePriority;
    volatile int                    ScheduleSelf;       /* 1=(Use
                                                           SchedulePriority-
                                                           1),0=no change */
    volatile ccurpmfc_uf_action_t   Action;
    volatile ccurpmfc_uf_state_t    State;
    volatile int                    CommandPending;
    volatile void                   *Next_UserFunction;
    volatile unsigned int long long RunCount;
    volatile int                    Pause;
} _ccurpmfc_UserFunction_t;
```

2.2.74 ccurPMFC_DAC_Activate()

This call must be the first call to activate the DAC. Without activation, all other calls to the DAC will fail. The user can also use this call to return the current state of the DAC without any change by specifying a pointer to *current_state* and setting *activate* to *CCURPMFC_DAC_ALL_ENABLE_DO_NOT_CHANGE*. If the DAC is already active and the user issues a *CCURPMFC_DAC_ALL_ENABLE*, no additional activation will be performed. To cause the DAC to go through a full reset, the user needs to issue the *CCURPMFC_DAC_ALL_RESET* which will cause the DAC to disable and then re-enable, setting all its DAC values to a default state. DAC calibration data will also be reset.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Activate (void                        *Handle,
                          _ccurpmfc_dac_all_enable_t  activate,
                          _ccurpmfc_dac_all_enable_t  *current_state)

   Description: Activate/DeActivate DAC module

   Input:   void                            *Handle         (Handle pointer)
            _ccurpmfc_dac_all_enable_t      activate        (activate/deactivate)
               # CCURPMFC_DAC_ALL_DISABLE
               # CCURPMFC_DAC_ALL_ENABLE
               # CCURPMFC_DAC_ALL_RESET     (disable followed by enable)
               # CCURPMFC_DAC_ALL_ENABLE_DO_NOT_CHANGE
   Output:  _ccurpmfc_dac_all_enable_t     *current_state  (active/deactive)
```

```
            # CCURPMFC_DAC_ALL_DISABLE
            # CCURPMFC_DAC_ALL_ENABLE
   Return:  _ccurpmfc_lib_error_number_t
            # CCURPMFC_LIB_NO_ERROR               (successful)
            # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN               (device not open)
            # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
*************************************************************************/
```

2.2.75 ccurPMFC_DAC_Get_CSR()

This call returns information from the DAC registers for the selected channel group.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Get_CSR (void                  *Handle,
                         _ccurpmfc_dac_mask_t  dac_mask,
                         ccurpmfc_dac_csr_t    *dac_csr)

   Description: Get DAC Control and Status information

   Input:   void                                  *Handle  (Handle pointer)
            _ccurpmfc_dac_mask_t                  dac_mask (selected DAC mask)
               # CCURPMFC_DAC_MASK_0_3
               # CCURPMFC_DAC_MASK_4_7
               # CCURPMFC_DAC_MASK_8_11
               # CCURPMFC_DAC_MASK_12_15
   Output:  ccurpmfc_dac_csr_t                    *dac_csr (pointer to DAC csr)
               _ccurpmfc_daccsr_busy_t            dac_interface_busy
                  # CCURPMFC_DAC_IDLE
                  # CCURPMFC_DAC_BUSY
               _ccurpmfc_daccsr_powerdown_t       dac_powerdown
                  # CCURPMFC_DAC_OPERATIONAL
                  # CCURPMFC_DAC_POWERDOWN
               _ccurpmfc_daccsr_updmode_t         dac_update_mode
                  # CCURPMFC_DAC_MODE_IMMEDIATE
                  # CCURPMFC_DAC_MODE_SYNCHRONIZED
               _ccurpmfc_daccsr_data_format_t     dac_data_format
                  # CCURPMFC_DAC_OFFSET_BINARY
                  # CCURPMFC_DAC_TWOS_COMPLEMENT
               _ccurpmfc_daccsr_output_select_t   dac_output_select
                  # CCURPMFC_DAC_SINGLE_ENDED
                  # CCURPMFC_DAC_DIFFERENTIAL
               _ccurpmfc_daccsr_output_range_t    dac_output_range
                    # CCURPMFC_DAC_SINGLE_ENDED_UNIPOLAR_10V
                    # CCURPMFC_DAC_SINGLE_ENDED_BIPOLAR_5V
                    # CCURPMFC_DAC_SINGLE_ENDED_BIPOLAR_10V
                    # CCURPMFC_DAC_SINGLE_ENDED_UNIPOLAR_20V

                    # CCURPMFC_DAC_DIFFERENTIAL_UNIPOLAR_10V
                    # CCURPMFC_DAC_DIFFERENTIAL_BIPOLAR_10V
                    # CCURPMFC_DAC_DIFFERENTIAL_BIPOLAR_20V
                    # CCURPMFC_DAC_DIFFERENTIAL_UNIPOLAR_20V

   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
```

```
            # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE       (DAC is not active)
 ****************************************************************************/
```

2.2.76 ccurPMFC_DAC_Get_Driver_Write_Mode()

This call returns the current driver DAC write mode. When a *write(2)* system call is issued, it is this mode that determines the type of write being performed by the driver.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Get_Driver_Write_Mode (void                          *Handle,
                                  _ccurpmfc_driver_DAC_write_mode_t  *mode)

   Description: Get current DAC write mode that will be selected by the 'write()'
                                                                       call

   Input:   void                               *Handle (Handle pointer)
   Output:  _ccurpmfc_driver_DAC_write_mode_t  *mode   (select DAC write mode)
               # CCURPMFC_DAC_PIO_CHANNEL
               # CCURPMFC_DAC_PIO_FIFO
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
               # CCURPMFC_LIB_IOCTL_FAILED           (driver ioctl call failed)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
 ****************************************************************************/
```

2.2.77 ccurPMFC_DAC_Get_Fifo_Channel_Select()

This call returns the current Fifo Channel selection mask. Only samples for these selected channels are placed in the FIFO during sample generation. Unlike the ADC Fifo channel select option, this DAC FIFO channel select option also restricts DAC channel register writes to those selected by this option. The advantage for implementing this option for DAC channel registers in addition to DAC FIFO is that the user can perform DMA operations to generate samples on selected channels without affecting the output of those channels that have not been included in the channel selection.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Get_Fifo_Channel_Select(void       *Handle,
                                  _ccurpmfc_dac_channel_mask_t
                                                   *dac_fifo_channel_select_mask)

   Description: DAC Get Fifo Channel Selection

   Input:   void                         *Handle            (handle pointer)
   Output:  _ccurpmfc_dac_channel_mask_t *dac_fifo_channel_select_mask
                                                            (channel select mask)
               # CCURPMFC_DAC_CHANNEL_MASK_0
               # CCURPMFC_DAC_CHANNEL_MASK_1
               # CCURPMFC_DAC_CHANNEL_MASK_2
               # CCURPMFC_DAC_CHANNEL_MASK_3
               # CCURPMFC_DAC_CHANNEL_MASK_4
               # CCURPMFC_DAC_CHANNEL_MASK_5
               # CCURPMFC_DAC_CHANNEL_MASK_6
               # CCURPMFC_DAC_CHANNEL_MASK_7
               # CCURPMFC_DAC_CHANNEL_MASK_8
               # CCURPMFC_DAC_CHANNEL_MASK_9
               # CCURPMFC_DAC_CHANNEL_MASK_10
               # CCURPMFC_DAC_CHANNEL_MASK_11
```

```
               # CCURPMFC_DAC_CHANNEL_MASK_12
               # CCURPMFC_DAC_CHANNEL_MASK_13
               # CCURPMFC_DAC_CHANNEL_MASK_14
               # CCURPMFC_DAC_CHANNEL_MASK_15
               # CCURPMFC_ALL_DAC_CHANNELS_MASK
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
 ***************************************************************************/
```

2.2.78 ccurPMFC_DAC_Get_Fifo_Info()

This call returns DAC FIFO information to the user.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Get_Fifo_Info (void                     *Handle,
                               ccurpmfc_dac_fifo_info_t *dac_fifo)

   Description: Get DAC FIFO control and Status information

   Input:   void                            *Handle   (Handle pointer)
   Output:  ccurpmfc_dac_fifo_info_t        *dac_fifo (pointer to DAC fifo
                                                       struct)
               _ccurpmfc_dac_fifo_reset_t      reset;
                  # CCURPMFC_DAC_FIFO_ACTIVE
                  # CCURPMFC_DAC_FIFO_RESET
               _ccurpmfc_dac_fifo_overflow_t   overflow;
                  # CCURPMFC_DAC_FIFO_NO_OVERFLOW
                  # CCURPMFC_DAC_FIFO_OVERFLOW
               _ccurpmfc_dac_fifo_underflow_t  underflow;
                  # CCURPMFC_DAC_FIFO_NO_UNDERFLOW
                  # CCURPMFC_DAC_FIFO_UNDERFLOW
               _ccurpmfc_dac_fifo_full_t       full;
                  # CCURPMFC_DAC_FIFO_NOT_FULL
                  # CCURPMFC_DAC_FIFO_FULL
               _ccurpmfc_dac_fifo_threshold_t  threshold_exceeded;
                  # CCURPMFC_DAC_FIFO_THRESHOLD_NOT_EXCEEDED
                  # CCURPMFC_DAC_FIFO_THRESHOLD_EXCEEDED
               _ccurpmfc_dac_fifo_empty_t      empty;
                  # CCURPMFC_DAC_FIFO_NOT_EMPTY
                  # CCURPMFC_DAC_FIFO_EMPTY
               uint                             data_counter;
               uint                             threshold;
               uint                             max_threshold;
               uint                             driver_threshold;
               uint                             write_count;
               dac_fifo_channel_select_mask     channel_select_mask;
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
***************************************************************************/
```

2.2.79 ccurPMFC_DAC_Get_Fifo_Threshold()

This call returns the DAC Fifo threshold information.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Get_Fifo_Threshold(void *Handle,
                                   uint *dac_threshold)

   Description: DAC Get Fifo Threshold

   Input:   void                    *Handle        (handle pointer)
   Output:  uint                    *dac_threshold (DAC fifo threshold)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
*****************************************************************************/
```

2.2.80 ccurPMFC_DAC_Get_Fifo_Write_Count()

This call returns the count of the DAC FIFO. It is updated anytime the FIFO is written to. This is only used for debug.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Get_Fifo_Write_Count(void *Handle,
                                     uint *dac_write_count)

   Description: DAC Get Fifo Write Count

   Input:   void                    *Handle          (handle pointer)
   Output:  uint                    *dac_write_count (DAC fifo write count)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
*****************************************************************************/
```

2.2.81 ccurPMFC_DAC_Get_Gain_Cal()

This call returns the DAC calibration gain values.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Get_Gain_Cal(void                          *Handle,
                             _ccurpmfc_dac_channel_mask_t ChanMask
                             ccurpmfc_dac_cal_t           *cal)

   Description: Get the DAC Gain Calibration data.

   Input:   void                          *Handle  (handle pointer)
             _ccurpmfc_dac_channel_mask_t ChanMask (channel selection mask)
               # CCURPMFC_DAC_CHANNEL_MASK_0
               # CCURPMFC_DAC_CHANNEL_MASK_1
               # CCURPMFC_DAC_CHANNEL_MASK_2
```

```
               # CCURPMFC_DAC_CHANNEL_MASK_3
               # CCURPMFC_DAC_CHANNEL_MASK_4
               # CCURPMFC_DAC_CHANNEL_MASK_5
               # CCURPMFC_DAC_CHANNEL_MASK_6
               # CCURPMFC_DAC_CHANNEL_MASK_7
               # CCURPMFC_DAC_CHANNEL_MASK_8
               # CCURPMFC_DAC_CHANNEL_MASK_9
               # CCURPMFC_DAC_CHANNEL_MASK_10
               # CCURPMFC_DAC_CHANNEL_MASK_11
               # CCURPMFC_DAC_CHANNEL_MASK_12
               # CCURPMFC_DAC_CHANNEL_MASK_13
               # CCURPMFC_DAC_CHANNEL_MASK_14
               # CCURPMFC_DAC_CHANNEL_MASK_15
               # CCURPMFC_ALL_DAC_CHANNELS_MASK
   Output:  ccurpmfc_dac_cal_t           *cal     (pointer to board cal)
               uint    Raw[CCURPMFC_MAX_DAC_CHANNELS]
               double  Float[CCURPMFC_MAX_DAC_CHANNELS]
               double  CalibrationReferenceVoltage[CCURPMFC_MAX_DAC_CHANNELS]
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR           (successful)
               # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN           (device not open)
               # CCURPMFC_LIB_INVALID_ARG        (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION    (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE  (DAC is not active)
*************************************************************************/
```

2.2.82 ccurPMFC_DAC_Get_Offset_Cal()

This call returns the DAC calibration offset values.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Get_Offset_Cal(void                       *Handle,
                               _ccurpmfc_dac_channel_mask_t ChanMask
                                 ccurpmfc_dac_cal_t         *cal)

   Description: Get the DAC Offset Calibration data.

   Input:   void                          *Handle  (handle pointer)
             _ccurpmfc_dac_channel_mask_t ChanMask (channel selection mask)
               # CCURPMFC_DAC_CHANNEL_MASK_0
               # CCURPMFC_DAC_CHANNEL_MASK_1
               # CCURPMFC_DAC_CHANNEL_MASK_2
               # CCURPMFC_DAC_CHANNEL_MASK_3
               # CCURPMFC_DAC_CHANNEL_MASK_4
               # CCURPMFC_DAC_CHANNEL_MASK_5
               # CCURPMFC_DAC_CHANNEL_MASK_6
               # CCURPMFC_DAC_CHANNEL_MASK_7
               # CCURPMFC_DAC_CHANNEL_MASK_8
               # CCURPMFC_DAC_CHANNEL_MASK_9
               # CCURPMFC_DAC_CHANNEL_MASK_10
               # CCURPMFC_DAC_CHANNEL_MASK_11
               # CCURPMFC_DAC_CHANNEL_MASK_12
               # CCURPMFC_DAC_CHANNEL_MASK_13
               # CCURPMFC_DAC_CHANNEL_MASK_14
               # CCURPMFC_DAC_CHANNEL_MASK_15
               # CCURPMFC_ALL_DAC_CHANNELS_MASK
   Output:  ccurpmfc_dac_cal_t           *cal     (pointer to board cal)
               uint    Raw[CCURPMFC_MAX_DAC_CHANNELS]
               double  Float[CCURPMFC_MAX_DAC_CHANNELS]
```

```
            double  CalibrationReferenceVoltage[CCURPMFC_MAX_DAC_CHANNELS]
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE    (DAC is not active)
 ***************************************************************************/
```

2.2.83 ccurPMFC_DAC_Get_Update_Source_Select()

This call allows the user to return the selected DAC update source.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Get_Update_Source_Select (void                    *Handle,
                                          _ccurpmfc_daccsr_update_source_t
                                                                  *update_source)

   Description: Get DAC Update Source Select

   Input:   void                               *Handle        (Handle pointer)
   Output:  _ccurpmfc_daccsr_update_source_t  *update_source (pointer to update
                                                              source)
               # CCURPMFC_DAC_UPDATE_SOFTWARE
               # CCURPMFC_DAC_UPDATE_CLOCK_0
               # CCURPMFC_DAC_UPDATE_CLOCK_1
               # CCURPMFC_DAC_UPDATE_CLOCK_2
               # CCURPMFC_DAC_UPDATE_CLOCK_3
               # CCURPMFC_DAC_UPDATE_CLOCK_4
               # CCURPMFC_DAC_UPDATE_CLOCK_5
               # CCURPMFC_DAC_UPDATE_CLOCK_6
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
 ***************************************************************************/
```

2.2.84 ccurPMFC_DAC_Perform_Auto_Calibration()

This single call performs a full DAC calibration of all the channels using the ADC. The ADC needs to be first calibrated prior to issuing this call.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Perform_Auto_Calibration(void                          *Handle,
                                         _ccurpmfc_dac_channel_mask_t   ChanMask)

   Description: Perform DAC Auto Calibration for selected channels

   Input:    void                          *Handle  (handle pointer)
             _ccurpmfc_dac_channel_mask_t  ChanMask (channel selection mask)
               # CCURPMFC_DAC_CHANNEL_MASK_0
               # CCURPMFC_DAC_CHANNEL_MASK_1
               # CCURPMFC_DAC_CHANNEL_MASK_2
               # CCURPMFC_DAC_CHANNEL_MASK_3
               # CCURPMFC_DAC_CHANNEL_MASK_4
```

```
              # CCURPMFC_DAC_CHANNEL_MASK_5
              # CCURPMFC_DAC_CHANNEL_MASK_6
              # CCURPMFC_DAC_CHANNEL_MASK_7
              # CCURPMFC_DAC_CHANNEL_MASK_8
              # CCURPMFC_DAC_CHANNEL_MASK_9
              # CCURPMFC_DAC_CHANNEL_MASK_10
              # CCURPMFC_DAC_CHANNEL_MASK_11
              # CCURPMFC_DAC_CHANNEL_MASK_12
              # CCURPMFC_DAC_CHANNEL_MASK_13
              # CCURPMFC_DAC_CHANNEL_MASK_14
              # CCURPMFC_DAC_CHANNEL_MASK_15
              # CCURPMFC_ALL_DAC_CHANNELS_MASK
   Output:    none
   Return:    _ccurpmfc_lib_error_number_t
                 # CCURPMFC_LIB_NO_ERROR               (successful)
                 # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
                 # CCURPMFC_LIB_NOT_OPEN               (library not open)
                 # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
                 # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE      (DAC is not active)
                 # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE      (ADC is not active)
                 # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
                 # CCURPMFC_LIB_CALIBRATION_RANGE_ERROR (range error)
 ************************************************************************/
```

2.2.85 ccurPMFC_DAC_Perform_Gain_Calibration()

This call performs a *gain* calibration using the ADC. The ADC needs to be first calibrated prior to issuing this call.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Perform_Gain_Calibration(void                          *Handle,
                                         _ccurpmfc_dac_channel_mask_t   ChanMask)

   Description: Perform DAC Gain Calibration for selected channels

   Input:     void                          *Handle  (handle pointer)
              _ccurpmfc_dac_channel_mask_t  ChanMask (channel selection mask)
                # CCURPMFC_DAC_CHANNEL_MASK_0
                # CCURPMFC_DAC_CHANNEL_MASK_1
                # CCURPMFC_DAC_CHANNEL_MASK_2
                # CCURPMFC_DAC_CHANNEL_MASK_3
                # CCURPMFC_DAC_CHANNEL_MASK_4
                # CCURPMFC_DAC_CHANNEL_MASK_5
                # CCURPMFC_DAC_CHANNEL_MASK_6
                # CCURPMFC_DAC_CHANNEL_MASK_7
                # CCURPMFC_DAC_CHANNEL_MASK_8
                # CCURPMFC_DAC_CHANNEL_MASK_9
                # CCURPMFC_DAC_CHANNEL_MASK_10
                # CCURPMFC_DAC_CHANNEL_MASK_11
                # CCURPMFC_DAC_CHANNEL_MASK_12
                # CCURPMFC_DAC_CHANNEL_MASK_13
                # CCURPMFC_DAC_CHANNEL_MASK_14
                # CCURPMFC_DAC_CHANNEL_MASK_15
                # CCURPMFC_ALL_DAC_CHANNELS_MASK
   Output:    none
   Return:    _ccurpmfc_lib_error_number_t
                 # CCURPMFC_LIB_NO_ERROR               (successful)
                 # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
                 # CCURPMFC_LIB_NOT_OPEN               (library not open)
                 # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
```

```
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE       (DAC is not active)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE       (ADC is not active)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
               # CCURPMFC_LIB_CALIBRATION_RANGE_ERROR (range error)
 ***************************************************************************/
```

2.2.86 ccurPMFC_DAC_Perform_Offset_Calibration()

This call performs a *offset* calibration using the ADC. The ADC needs to be first calibrated prior to issuing this call.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Perform_Offset_Calibration(void                          *Handle,
                                      _ccurpmfc_dac_channel_mask_t   ChanMask)

   Description: Perform DAC Offset Calibration for selected channels

   Input:    void                          *Handle  (handle pointer)
             _ccurpmfc_dac_channel_mask_t  ChanMask (channel selection mask)
               # CCURPMFC_DAC_CHANNEL_MASK_0
               # CCURPMFC_DAC_CHANNEL_MASK_1
               # CCURPMFC_DAC_CHANNEL_MASK_2
               # CCURPMFC_DAC_CHANNEL_MASK_3
               # CCURPMFC_DAC_CHANNEL_MASK_4
               # CCURPMFC_DAC_CHANNEL_MASK_5
               # CCURPMFC_DAC_CHANNEL_MASK_6
               # CCURPMFC_DAC_CHANNEL_MASK_7
               # CCURPMFC_DAC_CHANNEL_MASK_8
               # CCURPMFC_DAC_CHANNEL_MASK_9
               # CCURPMFC_DAC_CHANNEL_MASK_10
               # CCURPMFC_DAC_CHANNEL_MASK_11
               # CCURPMFC_DAC_CHANNEL_MASK_12
               # CCURPMFC_DAC_CHANNEL_MASK_13
               # CCURPMFC_DAC_CHANNEL_MASK_14
               # CCURPMFC_DAC_CHANNEL_MASK_15
               # CCURPMFC_ALL_DAC_CHANNELS_MASK
   Output:   none
   Return:   _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (library not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE      (DAC is not active)
               # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE      (ADC is not active)
               # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
               # CCURPMFC_LIB_CALIBRATION_RANGE_ERROR (range error)
 ***************************************************************************/
```

2.2.87 ccurPMFC_DAC_Read_Channels_Calibration()

This routine reads the DAC channel calibration registers and dumps all of them to the user specified file. If the file name specified is NULL, then information is written to *stdout.*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Read_Channels_Calibration(void *Handle,
                                      char *filename)
```

```
   Description: Read DAC Channels Calibration

   Input:    void   *Handle                      (handle pointer)
   Output:   char   *filename                    (pointer to filename)
   Return:   _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR          (successful)
                # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN          (library not open)
                # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
                # CCURPMFC_LIB_CANNOT_OPEN_FILE  (cannot open calib. file)
                # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (ADC is not active)
 ***************************************************************************/
```

e.g.

```
#Date           : Fri Dec 14 11:52:02 2018

#Chan   Gain                    Offset
#====   ====                    ======
 ch00:  0.0028991699218750  0.0002288818359375
 ch01:  0.0030517578125000  0.0001525878906250
 ch02:  0.0024414062500000  0.0001716613769531
 ch03:  0.0024414062500000  0.0001525878906250
 ch04:  0.0030517578125000  0.0002288818359375
 ch05:  0.0030517578125000  0.0001907348632812
 ch06:  0.0028991699218750  0.0002098083496094
 ch07:  0.0027465820312500  0.0001907348632812
 ch08:  0.0028991699218750  0.0001525878906250
 ch09:  0.0030517578125000  0.0001907348632812
 ch10:  0.0032043457031250  0.0001716613769531
 ch11:  0.0032043457031250  0.0000190734863281
 ch12:  0.0022888183593750  0.0002479553222656
 ch13:  0.0022888183593750  0.0002288818359375
 ch14:  0.0027465820312500  0.0002288818359375
 ch15:  0.0025939941406250  0.0001907348632812
```

2.2.88 ccurPMFC_DAC_ReadBack_Channels()

This call is more of debug purpose. It causes the DAC channels output to be connected to a user specified ADC channel and then returns the current reading of the ADC channel. Hence, we have read back the DAC channels selected.

If the ADC configuration *adc_csr* is *NULL*, then the call reads the current ADC configuration for the selected ADC channel *ADCChan*, however, the user can instead supply a pointer to the ADC configuration *adc_csr* where they have already configured the ADC as the following:

adc_data_format = CCURPMFC_ADC_TWOS_COMPLEMENT
adc_input_range = CCURPMFC_ADC_BIPOLAR_10V
adc_input_signal = CCURPMFC_ADC_CALIBRATION_BUS
adc_update_clock = CCURPMFC_ADC_UPDATE_CLOCK_0

Note that the *adc_update_clock* should be set to the currently active clock. Failure to set the above values will result in the call failing with invalid argument *CCURPMFC_LIB_INVALID_ARG*.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_ReadBack_Channels(void                        *Handle,
                                  _ccurpmfc_dac_channel_mask_t DACChanMask,
                                  _ccurpmfc_adc_channel_t      ADCChan,
                                  _ccurpmfc_adc_csr_t          *adc_csr,
                                  ccurpmfc_dac_volts_t         *dac_volts)
```

```
  Description: Read Back DAC Channels using ADC

  Input:   void                          *Handle       (Handle pointer)
           _ccurpmfc_dac_channel_mask_t  DACChanMask   (specify DAC channel mask)
              # CCURPMFC_DAC_CHANNEL_MASK_0
              # CCURPMFC_DAC_CHANNEL_MASK_1
              # CCURPMFC_DAC_CHANNEL_MASK_2
              # CCURPMFC_DAC_CHANNEL_MASK_3
              # CCURPMFC_DAC_CHANNEL_MASK_4
              # CCURPMFC_DAC_CHANNEL_MASK_5
              # CCURPMFC_DAC_CHANNEL_MASK_6
              # CCURPMFC_DAC_CHANNEL_MASK_7
              # CCURPMFC_DAC_CHANNEL_MASK_8
              # CCURPMFC_DAC_CHANNEL_MASK_9
              # CCURPMFC_DAC_CHANNEL_MASK_10
              # CCURPMFC_DAC_CHANNEL_MASK_11
              # CCURPMFC_DAC_CHANNEL_MASK_12
              # CCURPMFC_DAC_CHANNEL_MASK_13
              # CCURPMFC_DAC_CHANNEL_MASK_14
              # CCURPMFC_DAC_CHANNEL_MASK_15
              # CCURPMFC_ALL_DAC_CHANNELS_MASK
           _ccurpmfc_adc_channel_t                ADCChan  (ADC channel to
                                                            read)
           _ccurpmfc_adc_csr_t                    *adc_csr (pointer to ADC csr)
              _ccurpmfc_adccsr_update_clock_t     adc_update_clock
                 # CCURPMFC_ADC_UPDATE_CLOCK_NONE
                 # CCURPMFC_ADC_UPDATE_CLOCK_0
                 # CCURPMFC_ADC_UPDATE_CLOCK_1
                 # CCURPMFC_ADC_UPDATE_CLOCK_2
                 # CCURPMFC_ADC_UPDATE_CLOCK_3
                 # CCURPMFC_ADC_UPDATE_CLOCK_4
                 # CCURPMFC_ADC_UPDATE_CLOCK_5
                 # CCURPMFC_ADC_UPDATE_CLOCK_6
              _ccurpmfc_adccsr_input_signal_t     adc_input_signal
                 # CCURPMFC_ADC_EXTERNAL_SIGNAL
                 # CCURPMFC_ADC_CALIBRATION_BUS
              _ccurpmfc_adccsr_data_format_t      adc_data_format
                 # CCURPMFC_ADC_OFFSET_BINARY
                 # CCURPMFC_ADC_TWOS_COMPLEMENT
              _ccurpmfc_adccsr_input_range_t      adc_input_range
                 # CCURPMFC_ADC_BIPOLAR_10V
                 # CCURPMFC_ADC_BIPOLAR_5V
  Output:  ccurpmfc_dac_volts_t                  *dac_volts (pointer to DAC
                                                            volts)
                 uint    Raw[CCURPMFC_MAX_DAC_CHANNELS];
                 double  Float[CCURPMFC_MAX_DAC_CHANNELS];
  Return:  _ccurpmfc_lib_error_number_t
              # CCURPMFC_LIB_NO_ERROR             (no error)
              # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
              # CCURPMFC_LIB_NOT_OPEN             (library not open)
              # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
              # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
              # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE    (DAC is not active)
              # CCURPMFC_LIB_ADC_IS_NOT_ACTIVE    (ADC is not active)
              # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE  (Clock is not active)
*************************************************************************/
```

2.2.89 ccurPMFC_DAC_Read_Channels()

This call provides the user an easy method of reading the DAC channels. User can supply a channel mask. If pointer to *dac_csr* is NULL, then the routine itself computes the current DAC configuration. For performance, the user should get the current DAC configuration using the *ccurPMFC_DAC_CSR()* call to get the current settings and pass it to this routine. Hence, if the configuration is not changed, the user can continuously invoke *ccurPMFC_DAC_Read_Channels()* routine without incurring the additional overhead of routine calling the *ccurPMFC_DAC_CSR()* call.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Read_Channels(void                          *Handle,
                              _ccurpmfc_dac_channel_mask_t ChanMask,
                              _ccurpmfc_dac_csr_t          *dac_csr,
                              ccurpmfc_dac_volts_t         *dac_volts)

   Description: Read DAC Channels

   Input:   void                             *Handle       (Handle pointer)
            _ccurpmfc_dac_channel_mask_t     ChanMask     (specify channel mask)
               # CCURPMFC_DAC_CHANNEL_MASK_0
               # CCURPMFC_DAC_CHANNEL_MASK_1
               # CCURPMFC_DAC_CHANNEL_MASK_2
               # CCURPMFC_DAC_CHANNEL_MASK_3
               # CCURPMFC_DAC_CHANNEL_MASK_4
               # CCURPMFC_DAC_CHANNEL_MASK_5
               # CCURPMFC_DAC_CHANNEL_MASK_6
               # CCURPMFC_DAC_CHANNEL_MASK_7
               # CCURPMFC_DAC_CHANNEL_MASK_8
               # CCURPMFC_DAC_CHANNEL_MASK_9
               # CCURPMFC_DAC_CHANNEL_MASK_10
               # CCURPMFC_DAC_CHANNEL_MASK_11
               # CCURPMFC_DAC_CHANNEL_MASK_12
               # CCURPMFC_DAC_CHANNEL_MASK_13
               # CCURPMFC_DAC_CHANNEL_MASK_14
               # CCURPMFC_DAC_CHANNEL_MASK_15
               # CCURPMFC_ALL_DAC_CHANNELS_MASK
            _ccurpmfc_dac_csr_t                      *dac_csr (pointer to DAC csr)
               _ccurpmfc_daccsr_busy_t               dac_interface_busy
                  # CCURPMFC_DAC_IDLE
                  # CCURPMFC_DAC_BUSY
               _ccurpmfc_daccsr_powerdown_t          dac_powerdown
                  # CCURPMFC_DAC_OPERATIONAL
                  # CCURPMFC_DAC_POWERDOWN
               _ccurpmfc_daccsr_updmode_t            dac_update_mode
                  # CCURPMFC_DAC_MODE_IMMEDIATE
                  # CCURPMFC_DAC_MODE_SYNCHRONIZED
               _ccurpmfc_daccsr_data_format_t        dac_data_format
                  # CCURPMFC_DAC_OFFSET_BINARY
                  # CCURPMFC_DAC_TWOS_COMPLEMENT
               _ccurpmfc_daccsr_output_select_t      dac_output_select
                  # CCURPMFC_DAC_SINGLE_ENDED
                  # CCURPMFC_DAC_DIFFERENTIAL
               _ccurpmfc_daccsr_output_range_t       dac_output_range
                    # CCURPMFC_DAC_SINGLE_ENDED_UNIPOLAR_10V
                    # CCURPMFC_DAC_SINGLE_ENDED_BIPOLAR_5V
                    # CCURPMFC_DAC_SINGLE_ENDED_BIPOLAR_10V
                    # CCURPMFC_DAC_SINGLE_ENDED_UNIPOLAR_20V

                    # CCURPMFC_DAC_DIFFERENTIAL_UNIPOLAR_10V
                    # CCURPMFC_DAC_DIFFERENTIAL_BIPOLAR_10V
```

```
                  # CCURPMFC_DAC_DIFFERENTIAL_BIPOLAR_20V
                  # CCURPMFC_DAC_DIFFERENTIAL_UNIPOLAR_20V
   Output:  ccurpmfc_dac_volts_t               *dac_volts (pointer to DAC volts)
                 uint    Raw[CCURPMFC_MAX_DAC_CHANNELS];
                 double  Float[CCURPMFC_MAX_DAC_CHANNELS];
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (no error)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE    (DAC is not active)
 ************************************************************************/
```

2.2.90 ccurPMFC_DAC_Reset_Calibration()

This call resets the ADC calibration values on the card.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Reset_Calibration(void *Handle)

   Description: DAC Reset Calibration

   Input:   void                       *Handle  (handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
 ************************************************************************/
```

2.2.91 ccurPMFC_DAC_Reset_Fifo()

This call resets the DAC fifo to the power-on state. User can elect to activate the FIFO after a reset.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Reset_Fifo(void                        *Handle,
                           _ccurpmfc_dac_fifo_reset_t  activate)

   Description: DAC Reset Fifo

   Input:   void                       *Handle  (handle pointer)
            _ccurpmfc_dac_fifo_reset_t  activate (activate converter)
               # CCURPMFC_DAC_FIFO_ACTIVATE
               # CCURPMFC_DAC_FIFO_RESET
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
 ************************************************************************/
```

2.2.92 ccurPMFC_DAC_Set_CSR()

This call sets the DAC control registers for the selected channel group.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Set_CSR (void                  *Handle,
                         _ccurpmfc_dac_mask_t  dac_mask,
                         ccurpmfc_dac_csr_t    *dac_csr)

   Description: Set DAC Control and Status information

   Input:   void                                    *Handle  (Handle pointer)
            _ccurpmfc_dac_mask_t                    dac_mask (selected DAC mask)
               # CCURPMFC_DAC_MASK_0_3
               # CCURPMFC_DAC_MASK_4_7
               # CCURPMFC_DAC_MASK_8_11
               # CCURPMFC_DAC_MASK_12_15
            ccurpmfc_dac_csr_t                      *dac_csr (pointer to DAC csr)
               _ccurpmfc_daccsr_powerdown_t         dac_powerdown
                  # CCURPMFC_DAC_OPERATIONAL
                  # CCURPMFC_DAC_POWERDOWN
                  # CCURPMFC_DAC_POWERDOWN_DO_NOT_CHANGE
               _ccurpmfc_daccsr_updmode_t           dac_update_mode
                  # CCURPMFC_DAC_MODE_IMMEDIATE
                  # CCURPMFC_DAC_MODE_SYNCHRONIZED
                  # CCURPMFC_DAC_MODE_DO_NOT_CHANGE
               _ccurpmfc_daccsr_data_format_t       dac_data_format
                  # CCURPMFC_DAC_OFFSET_BINARY
                  # CCURPMFC_DAC_TWOS_COMPLEMENT
                  # CCURPMFC_DAC_DATA_FORMAT_DO_NOT_CHANGE
               _ccurpmfc_daccsr_output_select_t     dac_output_select
                  # CCURPMFC_DAC_SINGLE_ENDED
                  # CCURPMFC_DAC_DIFFERENTIAL
                  # CCURPMFC_DAC_OUTPUT_SELECT_DO_NOT_CHANGE
               _ccurpmfc_daccsr_output_range_t      dac_output_range
                    # CCURPMFC_DAC_SINGLE_ENDED_UNIPOLAR_10V
                    # CCURPMFC_DAC_SINGLE_ENDED_BIPOLAR_5V
                    # CCURPMFC_DAC_SINGLE_ENDED_BIPOLAR_10V
                    # CCURPMFC_DAC_SINGLE_ENDED_UNIPOLAR_20V

                    # CCURPMFC_DAC_DIFFERENTIAL_UNIPOLAR_10V
                    # CCURPMFC_DAC_DIFFERENTIAL_BIPOLAR_10V
                    # CCURPMFC_DAC_DIFFERENTIAL_BIPOLAR_20V
                    # CCURPMFC_DAC_DIFFERENTIAL_UNIPOLAR_20V

                    # CCURPMFC_DAC_OUTPUT_RANGE_DO_NOT_CHANGE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE       (DAC is not active)
************************************************************************/
```

2.2.93 ccurPMFC_DAC_Set_Driver_Write_Mode()

This call sets the current driver write mode. When a *write(2)* system call is issued, it is this mode that determines the type of DAC write being performed by the driver. Refer to the *write(2)* system call under *Direct Driver Access* section for more information on the various modes.

```
/*************************************************************************
   ccurPMFC_DAC_Set_Driver_Write_Mode()
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Select_Driver_DAC_Write_Mode (void                          *Handle,
                                  _ccurpmfc_driver_DAC_write_mode_t   mode)

   Description: Select Driver DAC_Write Mode

   Input:   void                              *Handle (Handle pointer)
            _ccurpmfc_driver_DAC_write_mode_t mode    (select write mode)
               # CCURPMFC_DAC_PIO_CHANNEL
               # CCURPMFC_DAC_PIO_FIFO
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR         (successful)
               # CCURPMFC_LIB_BAD_HANDLE       (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN         (device not open)
               # CCURPMFC_LIB_INVALID_ARG      (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION  (local region not present)
 *************************************************************************/
```

2.2.94 ccurPMFC_DAC_Set_Fifo_Channel_Select()

This call allows the user to select a set of channels that need to be provided in the DAC FIFO or the DAC channel registers. Only samples for these selected channels are placed in the FIFO during sample generation. Unlike the ADC Fifo channel select option, this DAC FIFO channel select option also restricts DAC channel register writes to those selected by this option. The advantage for implementing this option for DAC channel registers in addition to DAC FIFO is that the user can perform DMA operations to generate samples on selected channels without affecting the output of those channels that have not been included in the channel selection.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Set_Fifo_Channel_Select(void       *Handle,
                                        _ccurpmfc_dac_channel_mask_t
                                                   dac_fifo_channel_select_mask)

   Description: DAC Set Fifo Channel Selection

   Input:   void                    *Handle                        (handle pointer)
                                    _ccurpmfc_dac_channel_mask_t
                                dac_fifo_channel_select_mask  (channel select mask)
               # CCURPMFC_DAC_CHANNEL_MASK_0
               # CCURPMFC_DAC_CHANNEL_MASK_1
               # CCURPMFC_DAC_CHANNEL_MASK_2
               # CCURPMFC_DAC_CHANNEL_MASK_3
               # CCURPMFC_DAC_CHANNEL_MASK_4
               # CCURPMFC_DAC_CHANNEL_MASK_5
               # CCURPMFC_DAC_CHANNEL_MASK_6
               # CCURPMFC_DAC_CHANNEL_MASK_7
               # CCURPMFC_DAC_CHANNEL_MASK_8
               # CCURPMFC_DAC_CHANNEL_MASK_9
               # CCURPMFC_DAC_CHANNEL_MASK_10
               # CCURPMFC_DAC_CHANNEL_MASK_11
```

```
               # CCURPMFC_DAC_CHANNEL_MASK_12
               # CCURPMFC_DAC_CHANNEL_MASK_13
               # CCURPMFC_DAC_CHANNEL_MASK_14
               # CCURPMFC_DAC_CHANNEL_MASK_15
               # CCURPMFC_ALL_DAC_CHANNELS_MASK
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
 ***************************************************************************/
```

2.2.95 ccurPMFC_DAC_Set_Fifo_Threshold()

This call allows the user to set the DAC FIFO threshold.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Set_Fifo_Threshold(void *Handle,
                                   uint dac_threshold)

   Description: DAC Set Fifo Threshold

   Input:   void                    *Handle        (handle pointer)
            uint                    dac_threshold  (DAC fifo threshold)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE    (DAC is not active)
****************************************************************************/
```

2.2.96 ccurPMFC_DAC_Set_Fifo_Write_Count()

This call allows the user the set the DAC FIFO write count. This is only used for debug.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Set_Fifo_Write_Count(void *Handle,
                                   uint dac_write_count)

   Description: DAC Set Fifo Write Count

   Input:   void                    *Handle          (handle pointer)
            uint                    dac_write_count  (DAC fifo write count)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (device not open)
               # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE     (DAC is not active)
 ***************************************************************************/
```

2.2.97 ccurPMFC_DAC_Set_Gain_Cal()

This call is used to set the DAC gain calibration for all the channels.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Set_Gain_Cal(void                          *Handle,
                             _ccurpmfc_dac_channel_mask_t ChanMask,
                             ccurpmfc_dac_cal_t           *cal)

   Description: Set the DAC Offset Calibration data.

   Input:   void                       *Handle    (handle pointer)
            _ccurpmfc_dac_channel_mask_t ChanMask (channel selection mask)
               # CCURPMFC_DAC_CHANNEL_MASK_0
               # CCURPMFC_DAC_CHANNEL_MASK_1
               # CCURPMFC_DAC_CHANNEL_MASK_2
               # CCURPMFC_DAC_CHANNEL_MASK_3
               # CCURPMFC_DAC_CHANNEL_MASK_4
               # CCURPMFC_DAC_CHANNEL_MASK_5
               # CCURPMFC_DAC_CHANNEL_MASK_6
               # CCURPMFC_DAC_CHANNEL_MASK_7
               # CCURPMFC_DAC_CHANNEL_MASK_8
               # CCURPMFC_DAC_CHANNEL_MASK_9
               # CCURPMFC_DAC_CHANNEL_MASK_10
               # CCURPMFC_DAC_CHANNEL_MASK_11
               # CCURPMFC_DAC_CHANNEL_MASK_12
               # CCURPMFC_DAC_CHANNEL_MASK_13
               # CCURPMFC_DAC_CHANNEL_MASK_14
               # CCURPMFC_DAC_CHANNEL_MASK_15
               # CCURPMFC_ALL_DAC_CHANNELS_MASK
            ccurpmfc_dac_cal_t         *cal       (pointer to board cal)
               uint    Raw[CCURPMFC_MAX_DAC_CHANNELS]
               double  Float[CCURPMFC_MAX_DAC_CHANNELS]
               double  CalibrationReferenceVoltage[CCURPMFC_MAX_DAC_CHANNELS]
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (device not open)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (local region not present)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE   (DAC is not active)
 ************************************************************************/
```

2.2.98 ccurPMFC_DAC_Set_Offset_Cal()

This call is used to set the DAC offset calibration for all the channels.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Set_Offset_Cal(void                          *Handle,
                               _ccurpmfc_dac_channel_mask_t ChanMask,
                               ccurpmfc_dac_cal_t           *cal)

   Description: Set the DAC Offset Calibration data.

   Input:   void                         *Handle  (handle pointer)
            _ccurpmfc_dac_channel_mask_t ChanMask (channel selection mask)
               # CCURPMFC_DAC_CHANNEL_MASK_0
               # CCURPMFC_DAC_CHANNEL_MASK_1
               # CCURPMFC_DAC_CHANNEL_MASK_2
```

```
            # CCURPMFC_DAC_CHANNEL_MASK_3
            # CCURPMFC_DAC_CHANNEL_MASK_4
            # CCURPMFC_DAC_CHANNEL_MASK_5
            # CCURPMFC_DAC_CHANNEL_MASK_6
            # CCURPMFC_DAC_CHANNEL_MASK_7
            # CCURPMFC_DAC_CHANNEL_MASK_8
            # CCURPMFC_DAC_CHANNEL_MASK_9
            # CCURPMFC_DAC_CHANNEL_MASK_10
            # CCURPMFC_DAC_CHANNEL_MASK_11
            # CCURPMFC_DAC_CHANNEL_MASK_12
            # CCURPMFC_DAC_CHANNEL_MASK_13
            # CCURPMFC_DAC_CHANNEL_MASK_14
            # CCURPMFC_DAC_CHANNEL_MASK_15
            # CCURPMFC_ALL_DAC_CHANNELS_MASK
         ccurpmfc_dac_cal_t          *cal     (pointer to board cal)
            uint    Raw[CCURPMFC_MAX_DAC_CHANNELS]
            double  Float[CCURPMFC_MAX_DAC_CHANNELS]
            double  CalibrationReferenceVoltage[CCURPMFC_MAX_DAC_CHANNELS]
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
            # CCURPMFC_LIB_NO_ERROR           (successful)
            # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN           (device not open)
            # CCURPMFC_LIB_INVALID_ARG        (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION    (local region not present)
            # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE  (DAC is not active)
 ************************************************************************/
```

2.2.99 ccurPMFC_DAC_Set_Update_Source_Select()

This call allows the user to set the DAC update source. Users can select either one of the defined clock generators or software update to cause the samples in the FIFO to be sent out.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Set_Update_Source_Select (void                   *Handle,
                                          _ccurpmfc_daccsr_update_source_t
                                                                 update_source)

   Description: Set DAC Update Source Select

   Input:   void                                *Handle        (Handle pointer)
            _ccurpmfc_daccsr_update_source_t    update_source  (pointer to update
                                                                        source)
            # CCURPMFC_DAC_UPDATE_SOFTWARE
            # CCURPMFC_DAC_UPDATE_CLOCK_0
            # CCURPMFC_DAC_UPDATE_CLOCK_1
            # CCURPMFC_DAC_UPDATE_CLOCK_2
            # CCURPMFC_DAC_UPDATE_CLOCK_3
            # CCURPMFC_DAC_UPDATE_CLOCK_4
            # CCURPMFC_DAC_UPDATE_CLOCK_5
            # CCURPMFC_DAC_UPDATE_CLOCK_6
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
            # CCURPMFC_LIB_NO_ERROR          (successful)
            # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN          (device not open)
            # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
            # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
 ************************************************************************/
```

2.2.100 ccurPMFC_DAC_Wait_For_Channel_Idle()

This call is used when writing to channel registers to ensure that we are not writing too fast. The DAC must be idle prior to the corresponding channel write, otherwise, the data would not be sent out.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Wait_For_Channel_Idle (void                    *Handle,
                                       _ccurpmfc_dac_channel_t dac_channel)

   Description: Wait for DAC Channel to go idle

   Input:   void                       *Handle         (Handle pointer)
            _ccurpmfc_dac_channel_t    dac_channel     (dac channel number)
               # CCURPMFC_DAC_CHANNEL_0
               # CCURPMFC_DAC_CHANNEL_1
               # CCURPMFC_DAC_CHANNEL_2
               # CCURPMFC_DAC_CHANNEL_3
               # CCURPMFC_DAC_CHANNEL_4
               # CCURPMFC_DAC_CHANNEL_5
               # CCURPMFC_DAC_CHANNEL_6
               # CCURPMFC_DAC_CHANNEL_7
               # CCURPMFC_DAC_CHANNEL_8
               # CCURPMFC_DAC_CHANNEL_9
               # CCURPMFC_DAC_CHANNEL_10
               # CCURPMFC_DAC_CHANNEL_11
               # CCURPMFC_DAC_CHANNEL_12
               # CCURPMFC_DAC_CHANNEL_13
               # CCURPMFC_DAC_CHANNEL_14
               # CCURPMFC_DAC_CHANNEL_15
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (device not open)
               # CCURPMFC_LIB_IOCTL_FAILED             (driver ioctl call failed)
               # CCURPMFC_LIB_INVALID_ARG              (Invalid argument)
               # CCURPMFC_LIB_DAC_FIFO_UNDERFLOW       (DAC Fifo underflow)
               # CCURPMFC_LIB_DAC_IS_BUSY              (DAC is busy)
 *************************************************************************/
```

2.2.101 ccurPMFC_DAC_Wait_For_Fifo_To_Drain()

This call is used prior to writing to the DAC fifo to ensure that there is enough sample space available to complete the write without getting an overflow condition. User needs to specify the threshold level to wait for the FIFO to reach, before returning to the caller. Note that if the threshold is too low, it is possible that the FIFO could experience an underflow condition prior to adding more samples to the FIFO.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Wait_For_Fifo_To_Drain (void   *Handle,
                                        uint   fifo_threshold)

   Description: Wait for DAC Fifo to drain

   Input:   void                       *Handle         (Handle pointer)
            uint                       fifo_threshold  (fifo threshold)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
```

```
            # CCURPMFC_LIB_NOT_OPEN                (device not open)
            # CCURPMFC_LIB_IOCTL_FAILED            (driver ioctl call
                                                                 failed)
            # CCURPMFC_LIB_INVALID_ARG             (Invalid argument)
            # CCURPMFC_LIB_DAC_FIFO_UNDERFLOW      (DAC Fifo underflow)
 ************************************************************************/
```

2.2.102 ccurPMFC_DAC_Write_Channels()

This call provides the user an easy method of writing the DAC channels. User can supply a channel mask. If pointer to *dac_csr* is NULL, then the routine itself computes the current DAC configuration. For performance, the user should get the current DAC configuration using the *ccurPMFC_DAC_CSR()* call to get the current settings and pass it to this routine. Hence, if the configuration is not changed, the user can continuously invoke *ccurPMFC_DAC_Write_Channels()* routine without incurring the additional overhead of routine calling the *ccurPMFC_DAC_CSR()* call.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DAC_Write_Channels(void                          *Handle,
                               _ccurpmfc_dac_channel_mask_t ChanMask,
                               _ccurpmfc_dac_csr_t          *dac_csr,
                               ccurpmfc_dac_volts_t         *dac_volts)

   Description: Read DAC Channels

   Input:   void                            *Handle       (Handle pointer)
            _ccurpmfc_dac_channel_mask_t    ChanMask     (specify channel mask)
               # CCURPMFC_DAC_CHANNEL_MASK_0
               # CCURPMFC_DAC_CHANNEL_MASK_1
               # CCURPMFC_DAC_CHANNEL_MASK_2
               # CCURPMFC_DAC_CHANNEL_MASK_3
               # CCURPMFC_DAC_CHANNEL_MASK_4
               # CCURPMFC_DAC_CHANNEL_MASK_5
               # CCURPMFC_DAC_CHANNEL_MASK_6
               # CCURPMFC_DAC_CHANNEL_MASK_7
               # CCURPMFC_DAC_CHANNEL_MASK_8
               # CCURPMFC_DAC_CHANNEL_MASK_9
               # CCURPMFC_DAC_CHANNEL_MASK_10
               # CCURPMFC_DAC_CHANNEL_MASK_11
               # CCURPMFC_DAC_CHANNEL_MASK_12
               # CCURPMFC_DAC_CHANNEL_MASK_13
               # CCURPMFC_DAC_CHANNEL_MASK_14
               # CCURPMFC_DAC_CHANNEL_MASK_15
               # CCURPMFC_ALL_DAC_CHANNELS_MASK
            _ccurpmfc_dac_csr_t                     *dac_csr (pointer to DAC csr)
               _ccurpmfc_daccsr_busy_t              dac_interface_busy
                  # CCURPMFC_DAC_IDLE
                  # CCURPMFC_DAC_BUSY
               _ccurpmfc_daccsr_powerdown_t         dac_powerdown
                  # CCURPMFC_DAC_OPERATIONAL
                  # CCURPMFC_DAC_POWERDOWN
               _ccurpmfc_daccsr_updmode_t           dac_update_mode
                  # CCURPMFC_DAC_MODE_IMMEDIATE
                  # CCURPMFC_DAC_MODE_SYNCHRONIZED
               _ccurpmfc_daccsr_data_format_t       dac_data_format
                  # CCURPMFC_DAC_OFFSET_BINARY
                  # CCURPMFC_DAC_TWOS_COMPLEMENT
               _ccurpmfc_daccsr_output_select_t     dac_output_select
                  # CCURPMFC_DAC_SINGLE_ENDED
                  # CCURPMFC_DAC_DIFFERENTIAL
               _ccurpmfc_daccsr_output_range_t      dac_output_range
```

```
                  # CCURPMFC_DAC_SINGLE_ENDED_UNIPOLAR_10V
                  # CCURPMFC_DAC_SINGLE_ENDED_BIPOLAR_5V
                  # CCURPMFC_DAC_SINGLE_ENDED_BIPOLAR_10V
                  # CCURPMFC_DAC_SINGLE_ENDED_UNIPOLAR_20V

                  # CCURPMFC_DAC_DIFFERENTIAL_UNIPOLAR_10V
                  # CCURPMFC_DAC_DIFFERENTIAL_BIPOLAR_10V
                  # CCURPMFC_DAC_DIFFERENTIAL_BIPOLAR_20V
                  # CCURPMFC_DAC_DIFFERENTIAL_UNIPOLAR_20V
            ccurpmfc_dac_volts_t               *dac_volts (pointer to DAC volts)
                  uint    Raw[CCURPMFC_MAX_DAC_CHANNELS];
                  double  Float[CCURPMFC_MAX_DAC_CHANNELS];
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (no error)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_VOLTAGE_NOT_IN_RANGE (voltage not in range)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE    (DAC is not active)
 ************************************************************************/
```

2.2.103 ccurPMFC_DAC_Write_Channels_Calibration()

This call allows the user to write the calibration registers from a user supplied calibration file. The format of the file is similar to that generated by the *ccurPMFC_DAC_Read_Channels_Calibration()* call. File can contain comments if they start with '#', '*', or empty lines. Additionally, users need only specify those channels that they wish to calibrate and the order of specifying channels in not important, however, the format of each channel entry needs to be adhered to. E.g. <chxx:> <gain> <offset>

```
/************************************************************************
   ccurPMFC_DAC_Write_Channels_Calibration()

   Description: Write Channels Calibration

   Input:    void   *Handle                     (handle pointer)
   Output:   char   *filename                   (pointer to filename)
   Return:   _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (library not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_CANNOT_OPEN_FILE  (cannot open calib. file)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
************************************************************************/
```

e.g.

```
#Date           : Fri Dec 14 11:52:02 2018

#Chan   Gain                    Offset
#====   ====                    ======
 ch00:  0.0028991699218750  0.0002288818359375
 ch01:  0.0030517578125000  0.0001525878906250
 ch02:  0.0024414062500000  0.0001716613769531
 ch03:  0.0024414062500000  0.0001525878906250
 ch04:  0.0030517578125000  0.0002288818359375
 ch05:  0.0030517578125000  0.0001907348632812
 ch06:  0.0028991699218750  0.0002098083496094
 ch07:  0.0027465820312500  0.0001907348632812
```

```
ch08:  0.0028991699218750  0.0001525878906250
ch09:  0.0030517578125000  0.0001907348632812
ch10:  0.0032043457031250  0.0001716613769531
ch11:  0.0032043457031250  0.0000190734863281
ch12:  0.0022888183593750  0.0002479553222656
ch13:  0.0022888183593750  0.0002288818359375
ch14:  0.0027465820312500  0.0002288818359375
ch15:  0.0025939941406250  0.0001907348632812
```

2.2.104 ccurPMFC_DataToVolts()

This routine takes a raw analog input data value and converts it to a floating point voltage based on the supplied format. Format can be *CCURPMFC_TWOS_COMPLEMENT* or *CCURPMFC_OFFSET_BINARY*. The data supplied in *us_data* must not be greater than the hardware resolution bits *CCURPMFC_ADC_RESOLUTION_BITS* supported by the board. Data greater than this will be masked out.

```
/*************************************************************************
   double ccurPMFC_DataToVolts(int us_data, ccurpmfc_volt_convert_t *conv)

   Description: Convert Data to volts

   Input:   int                                 us_data       (data to convert)
            ccurpmfc_volt_convert_t             *conv         (pointer to
                                                               conversion struct)
               double                           VoltageRange  (maximum voltage
                                                               range)
               _ccurpmfc_csr_dataformat_t       Format        (format)
                  # CCURPMFC_OFFSET_BINARY
                  # CCURPMFC_TWOS_COMPLEMENT
               ccurpmfc_bool                    BiPolar       (bi-polar)
                  # CCURPMFC_TRUE
                  # CCURPMFC_FALSE
               int                              ResolutionBits (Number of
                                                                resolution bits)
   Output:  none
   Return:  double                              volts          (returned volts)
 *************************************************************************/
```

2.2.105 ccurPMFC_Destroy_AllUserProcess()

The purpose of this call is to destroy all User Processes that have been previously created by the *ccurPMFC_Create_UserProcess()* command. *(This is an experimental API for debugging and testing).*

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t ccurPMFC_Destroy_AllUserProcess(void *Handle)

   Description: Destroy all created  user processes

   Input:   void                        *Handle      (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
 *************************************************************************/
```

2.2.106 ccurPMFC_Destroy_UserDioCosInterruptHandler()

The purpose of this call is to destroy the User DIO COS Interrupt handler that was created earlier with the *ccurPMFC_Create_UserDioCosInterruptHandler()* call.

```
/*************************************************************************
```

```
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Destroy_UserDioCosInterruptHandler(void *Handle)

   Description: Destroy a previously created User DIO COS Interrupt Handler

   Input:   void                  *Handle              (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_IOCTL_FAILED            (ioctl failed)
               # CCURPMFC_LIB_IO_ERROR                (failed to terminate
                                                       handler)
 ****************************************************************************/
```

2.2.107 ccurPMFC_Destroy_UserProcess()

The purpose of this call is to destroy the User Process that have been previously created by the *ccurPMFC_Create_UserProcess()* call. *(This is an experimental API for debugging and testing).*

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t ccurPMFC_Destroy_UserProcess(void *Handle,
                                  _ccurpmfc_UserFunction_t **UFuncHandle)

   Description: Destroy an already created  user process

   Input:   void                       *Handle          (Handle pointer)
            _ccurpmfc_UserFunction_t   **UFuncHandle    (pointer to user handle)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
 ****************************************************************************/
```

2.2.108 ccurPMFC_DIO_Activate()

This call must be the first call to activate the DIO. Without activation, all other calls to the DIO will fail. The user can also use this call to return the current state of the DIO without any change by specifying a pointer to *current_state* and setting *activate* to *CCURPMFC_DIO_ALL_ENABLE_DO_NOT_CHANGE*. If the DIO is already active and the user issues a *CCURPMFC_DIO_ALL_ENABLE*, no additional activation will be performed. To cause the DIO to go through a full reset, the user needs to issue the *CCURPMFC_DIO_ALL_RESET* which will cause the DIO to disable and then re-enable, setting all its DIO values to a default state.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Activate (void                       *Handle,
                          _ccurpmfc_dio_all_enable_t  activate,
                          _ccurpmfc_dio_all_enable_t  *current_state)

   Description: Activate/DeActivate DIO module

   Input:   void                            *Handle         (Handle pointer)
            _ccurpmfc_dio_all_enable_t      activate        (activate/deactivate)
               # CCURPMFC_DIO_ALL_DISABLE
               # CCURPMFC_DIO_ALL_ENABLE
               # CCURPMFC_DIO_ALL_RESET     (disable followed by enable)
               # CCURPMFC_DIO_ALL_ENABLE_DO_NOT_CHANGE
   Output:  _ccurpmfc_dio_all_enable_t      *current_state  (active/deactive)
               # CCURPMFC_DIO_ALL_DISABLE
```

```
            # CCURPMFC_DIO_ALL_ENABLE
  Return:  _ccurpmfc_lib_error_number_t
            # CCURPMFC_LIB_NO_ERROR                (successful)
            # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN                (device not open)
            # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
 ***************************************************************************/
```

2.2.109 ccurPMFC_DIO_Get_Channels_Polarity()

This call allows the user to get the polarity for the DIO channels. The *ChannelSelectMask* is used to retrieve polarity settings for selected channels.

For input channels, a value of *CCURPMFC_DIO_INPUT_LOW_TRUE* or *'0'* for polarity indicates low true, while a value of *CCURPMFC_DIO_INPUT_HIGH_TRUE* or *'1'* for polarity indicates high true.

For output channels, a value of *CCURPMFC_DIO_OUTPUT_LOW* or *'0'* for polarity indicates low or 0 volts, while a value of *CCURPMFC_DIO_OUTPUT_HIGH* or *'1'* for polarity indicates high or +5 volts.

```
/***************************************************************************
  _ccurpmfc_lib_error_number_t
  ccurPMFC_DIO_Get_Channels_Polarity(void                   *Handle,
                                  ccurpmfc_dio_channels_t  DIO_ChannelsPolarity,
                                  ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Get Channels Polarity

   Input:   void                    *Handle            (handle pointer)
            ccurpmfc_dio_channels_t  ChannelSelectMask (channel selection)
               # NULL                                  (select all channels)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
                   # CCURPMFC_DIO_CHANNEL_MASK_18
                   # CCURPMFC_DIO_CHANNEL_MASK_19
                   # CCURPMFC_DIO_CHANNEL_MASK_20
                   # CCURPMFC_DIO_CHANNEL_MASK_21
                   # CCURPMFC_DIO_CHANNEL_MASK_22
                   # CCURPMFC_DIO_CHANNEL_MASK_23
                   # CCURPMFC_DIO_CHANNEL_MASK_24
                   # CCURPMFC_DIO_CHANNEL_MASK_25
                   # CCURPMFC_DIO_CHANNEL_MASK_26
                   # CCURPMFC_DIO_CHANNEL_MASK_27
                   # CCURPMFC_DIO_CHANNEL_MASK_28
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
  Output:  ccurpmfc_dio_channels_t DIO_ChannelsPolarity (channels polarity
                                                          registers)
                # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                    # CCURPMFC_DIO_CHANNEL_MASK_0
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
  Return:  _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR          (successful)
                # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN          (device not open)
                # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
                # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.110 ccurPMFC_DIO_Get_COS_Channels_Edge_Sense()

This call returns to the user the settings for the change-of-state to sense the rising or falling edge of the signal on input for all the channels. The *ChannelSelectMask* is used to retrieve edge sense settings for selected

channels. A value of *CCURPMFC_DIO_COS_FALLING_EDGE* or '0' represents sensing of falling edge of input signal while a value of *CCURPMFC_DIO_COS_RISING_EDGE* or '1' represents sensing of rising edge of input signal.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Get_COS_Channels_Edge_Sense(void       *Handle,
                            ccurpmfc_dio_channels_t  DIO_COS_ChannelsEdgeSense,
                            ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Get COS Channels Edge Sense

   Input:   void                    *Handle                 (handle pointer)
            ccurpmfc_dio_channels_t ChannelSelectMask       (channel selection)
               # NULL                                       (select all channels)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
                   # CCURPMFC_DIO_CHANNEL_MASK_18
                   # CCURPMFC_DIO_CHANNEL_MASK_19
                   # CCURPMFC_DIO_CHANNEL_MASK_20
                   # CCURPMFC_DIO_CHANNEL_MASK_21
                   # CCURPMFC_DIO_CHANNEL_MASK_22
                   # CCURPMFC_DIO_CHANNEL_MASK_23
                   # CCURPMFC_DIO_CHANNEL_MASK_24
                   # CCURPMFC_DIO_CHANNEL_MASK_25
                   # CCURPMFC_DIO_CHANNEL_MASK_26
                   # CCURPMFC_DIO_CHANNEL_MASK_27
                   # CCURPMFC_DIO_CHANNEL_MASK_28
                   # CCURPMFC_DIO_CHANNEL_MASK_29
                   # CCURPMFC_DIO_CHANNEL_MASK_30
                   # CCURPMFC_DIO_CHANNEL_MASK_31
                   # CCURPMFC_DIO_ALL_CHANNELS_MASK
               CCURPMFC_DIO_MAX_REGISTERS can be one of:
                   # CCURPMFC_DIO_CHAN_00_31
                   # CCURPMFC_DIO_CHAN_32_63
                   # CCURPMFC_DIO_CHAN_64_95
   Output:  ccurpmfc_dio_channels_t   DIO_COS_ChannelsEdgeSense (COS channels
                                                               edge sense registers)
                # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
   Return:  _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR          (successful)
                # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN          (device not open)
                # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
                # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ***************************************************************************/
```

2.2.111 ccurPMFC_DIO_Get_COS_Channels_Enable()

This call returns to the user the settings for the change-of-state enable registers all the channels. The *ChannelSelectMask* is used to retrieve enable settings for selected channels. A value of *CCURPMFC_DIO_COS_IGNORE* or '0' ignores change-of-state while a value of *CCURPMFC_DIO_COS_ENABLE* or '1' represents enabling change-of-state for the selected channels.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Get_COS_Channels_Enable(void           *Handle,
                            ccurpmfc_dio_channels_t  DIO_COS_ChannelsEnable,
                            ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Get COS Channels Enable Setting

   Input:   void                    *Handle                (handle pointer)
            ccurpmfc_dio_channels_t ChannelSelectMask      (channel selection)
               # NULL                                      (select all channels)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
Output:  ccurpmfc_dio_channels_t   DIO_COS_ChannelsEnable (COS channels
                                                          enable registers)
              # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                    # CCURPMFC_DIO_CHANNEL_MASK_0
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
```

```
                # CCURPMFC_DIO_CHANNEL_MASK_22
                # CCURPMFC_DIO_CHANNEL_MASK_23
                # CCURPMFC_DIO_CHANNEL_MASK_24
                # CCURPMFC_DIO_CHANNEL_MASK_25
                # CCURPMFC_DIO_CHANNEL_MASK_26
                # CCURPMFC_DIO_CHANNEL_MASK_27
                # CCURPMFC_DIO_CHANNEL_MASK_28
                # CCURPMFC_DIO_CHANNEL_MASK_29
                # CCURPMFC_DIO_CHANNEL_MASK_30
                # CCURPMFC_DIO_CHANNEL_MASK_31
                # CCURPMFC_DIO_ALL_CHANNELS_MASK
             CCURPMFC_DIO_MAX_REGISTERS can be one of:
                # CCURPMFC_DIO_CHAN_00_31
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
   Return:  _ccurpmfc_lib_error_number_t
             # CCURPMFC_LIB_NO_ERROR          (successful)
             # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
             # CCURPMFC_LIB_NOT_OPEN          (device not open)
             # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
             # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
             # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.112 ccurPMFC_DIO_Get_COS_Channels_Mode()

This call returns to the user the settings for the change-of-state mode registers all the channels. The *ChannelSelectMask* is used to retrieve mode settings for selected channels. A value of *CCURPMFC_DIO_COS_ANY_TRANSITION* or '0' detects change-of-state on any edge transition while a value of *CCURPMFC_DIO_COS_RISING_OR_FALLING_TRANSITION* or '1' represents enabling change-of-state for either rising edge or falling edge depending on the channel edge sense setting for the selected channels.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Get_COS_Channels_Mode(void            *Handle,
                              ccurpmfc_dio_channels_t  DIO_COS_ChannelsMode,
                              ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Get COS Channels Mode Setting

   Input:   void                    *Handle                (handle pointer)
            ccurpmfc_dio_channels_t ChannelSelectMask      (channel selection)
               # NULL                                      (select all channels)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
Output:  ccurpmfc_dio_channels_t   DIO_COS_ChannelsMode (COS channels
                                                         Mode registers)
                 # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                    # CCURPMFC_DIO_CHANNEL_MASK_0
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
```

```
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.113 ccurPMFC_DIO_Get_COS_Channels_Overflow()

This call returns to the user the state of the change-of-state overflow registers for all the channels. The *ChannelSelectMask* is used to retrieve overflow settings for selected channels. A value of *CCURPMFC_DIO_COS_ OVERFLOW_DID_NOT_OCCUR* or '0' indicates that no overflow occurred while a value of *CCURPMFC_DIO_COS_OVERFLOW_OCCURRED* or '1' indicates that an overflow condition occurred for the selected channels. An overflow condition is set when a change-of-state condition is detected on a channel that previously detected a change-of-state condition without its status being cleared.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Get_COS_Channels_Overflow(void         *Handle,
                            ccurpmfc_dio_channels_t  DIO_COS_ChannelsOverflow,
                            ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Get COS Channels Overflow State

   Input:   void                    *Handle                (handle pointer)
            ccurpmfc_dio_channels_t ChannelSelectMask      (channel selection)
               # NULL                                      (select all channels)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
                   # CCURPMFC_DIO_CHANNEL_MASK_18
                   # CCURPMFC_DIO_CHANNEL_MASK_19
                   # CCURPMFC_DIO_CHANNEL_MASK_20
                   # CCURPMFC_DIO_CHANNEL_MASK_21
                   # CCURPMFC_DIO_CHANNEL_MASK_22
                   # CCURPMFC_DIO_CHANNEL_MASK_23
                   # CCURPMFC_DIO_CHANNEL_MASK_24
                   # CCURPMFC_DIO_CHANNEL_MASK_25
                   # CCURPMFC_DIO_CHANNEL_MASK_26
                   # CCURPMFC_DIO_CHANNEL_MASK_27
                   # CCURPMFC_DIO_CHANNEL_MASK_28
                   # CCURPMFC_DIO_CHANNEL_MASK_29
                   # CCURPMFC_DIO_CHANNEL_MASK_30
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
  Output:  ccurpmfc_dio_channels_t   DIO_COS_ChannelsOverflow (COS channels
                                                              Overflow registers)
                 # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                    # CCURPMFC_DIO_CHANNEL_MASK_0
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
  Return:  _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR          (successful)
                # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN          (device not open)
                # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
                # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.114 ccurPMFC_DIO_Get_COS_Channels_Status()

This call returns to the user the state of the change-of-state status registers for all the channels. The *ChannelSelectMask* is used to retrieve status settings for selected channels. A value of *CCURPMFC_DIO_COS_DID_NOT_OCCUR* or '0' indicates that no change-of-state occurred while a value of *CCURPMFC_DIO_COS_OCCURRED* or '1' indicates that a change-of-state condition occurred for the

selected channels. A change-of-state status is set when the hardware is enabled to detect a change of input signal transition and an input signal is received with the monitored transition.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Get_COS_Channels_Status(void           *Handle,
                            ccurpmfc_dio_channels_t  DIO_COS_ChannelsOStatus,
                            ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Get COS Channels Status

   Input:   void                    *Handle                (handle pointer)
            ccurpmfc_dio_channels_t ChannelSelectMask      (channel selection)
               # NULL                                      (select all channels)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
                   # CCURPMFC_DIO_CHANNEL_MASK_18
                   # CCURPMFC_DIO_CHANNEL_MASK_19
                   # CCURPMFC_DIO_CHANNEL_MASK_20
                   # CCURPMFC_DIO_CHANNEL_MASK_21
                   # CCURPMFC_DIO_CHANNEL_MASK_22
                   # CCURPMFC_DIO_CHANNEL_MASK_23
                   # CCURPMFC_DIO_CHANNEL_MASK_24
                   # CCURPMFC_DIO_CHANNEL_MASK_25
                   # CCURPMFC_DIO_CHANNEL_MASK_26
                   # CCURPMFC_DIO_CHANNEL_MASK_27
                   # CCURPMFC_DIO_CHANNEL_MASK_28
                   # CCURPMFC_DIO_CHANNEL_MASK_29
                   # CCURPMFC_DIO_CHANNEL_MASK_30
                   # CCURPMFC_DIO_CHANNEL_MASK_31
                   # CCURPMFC_DIO_ALL_CHANNELS_MASK
               CCURPMFC_DIO_MAX_REGISTERS can be one of:
                   # CCURPMFC_DIO_CHAN_00_31
                   # CCURPMFC_DIO_CHAN_32_63
                   # CCURPMFC_DIO_CHAN_64_95
   Output:  ccurpmfc_dio_channels_t   DIO_COS_ChannelsStatus (COS channels
                                                            Status registers)
                # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
   Return:  _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR          (successful)
                # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN          (device not open)
                # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
                # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ***************************************************************************/
```

2.2.115 ccurPMFC_DIO_Get_Input_Channels_Filter()

This call allows the user to get the settings for the DIO channels input filters. The *ChannelSelectMask* is used to retrieve filter settings for selected channels. A value of *CCURPMFC_DIO_INPUT_FILTER_ENABLED* or *'0'* for filter indicates that the 100 nanosecond filter is enabled for the selected channel, while a value of *CCURPMFC_DIO_INPUT_FILTER_DISABLED* or *'1'* indicates that the filter is disabled. On powerup, filter for all channels are enabled.

```
/***************************************************************************
  _ccurpmfc_lib_error_number_t
  ccurPMFC_DIO_Get_Input_Channels_Filter(void                *Handle,
                                  ccurpmfc_dio_channels_t  DIO_InputChannels,
                                  ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Get Input Channel Filters

   Input:   void                    *Handle            (handle pointer)
            ccurpmfc_dio_channels_t  ChannelSelectMask (channel selection)
               # NULL                                  (select all channels)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
Output:  ccurpmfc_dio_channels_t DIO_InputChannels   (input channel registers)
                # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                    # CCURPMFC_DIO_CHANNEL_MASK_0
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
   Return:  _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR          (successful)
                # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN          (device not open)
                # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
                # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.116 ccurPMFC_DIO_Get_Input_Snapshot()

This call returns the Input Snapshot state to the user. The purpose of this snapshot feature is to allow the user to read the input channels without the firmware updating them in the middle of the reads. In this way, they can ensure that all channels data are in sync.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Get_Input_Snapshot(void                              *Handle,
                                   _ccurpmfc_dio_input_snapshot_t  *dio_snapshot)

   Description: Get DIO Input Snapshot

   Input:   void                            *Handle              (handle pointer)
   Output:  _ccurpmfc_dio_input_snapshot_t *dio_snapshot        (dio snapshot)
                # CCURPMFC_DIO_INPUT_OPERATION_CONTINUOUS
                # CCURPMFC_DIO_INPUT_OPERATION_SNAPSHOT
   Return:  _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR          (successful)
                # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN          (device not open)
                # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
                # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
************************************************************************/
```

2.2.117 ccurPMFC_DIO_Get_Mode()

This call returns the current DIO mode.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Get_Mode(void                *Handle,
                         _ccurpmfc_dio_mode_t *dio_mode)

   Description: Get DIO Mode

   Input:   void                           *Handle                (handle pointer)
```

```
   Output:  _ccurpmfc_dio_mode_t         *dio_mode              (dio mode)
               # CCURPMFC_DIO_MODE_CUSTOM
               # CCURPMFC_DIO_MODE_NORMAL
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
*************************************************************************/
```

2.2.118 ccurPMFC_DIO_Get_Output_Sync()

This call returns the current state of the output sync flag. The purpose of the output sync feature is to ensure that the user can safely program all the output channels prior to directing the firmware to send them out simultaneously.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Get_Output_Sync(void                          *Handle,
                                _ccurpmfc_dio_output_sync_t    *dio_sync)

   Description: Get DIO Output Sync

   Input:   void                            *Handle            (handle pointer)
   Output:  _ccurpmfc_dio_output_sync_t     *dio_sync          (dio sync)
               # CCURPMFC_DIO_OUTPUT_OPERATION_CONTINUOUS
               # CCURPMFC_DIO_OUTPUT_OPERATION_SYNC
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
*************************************************************************/
```

2.2.119 ccurPMFC_DIO_Get_Ports_Direction()

This call allows the user to get the direction of the digital channels. There are 24 ports with grouping of 4 channels per port. Direction control is on a port level or a group of 4 channels.

When the direction for channels are set to output, then reading the channels input registers will result in acquiring what was written to the output (loopback). When the direction for the channels are set as inputs, then reading the channels input registers will result in acquiring signals coming into the board from the external digital lines.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Get_Ports_Direction(void                      *Handle,
                                    _ccurpmfc_dio_port_mask_t *dio_direction)

   Description: Get DIO Ports Direction Mask

   Input:   void                          *Handle              (handle pointer)
   Output:  _ccurpmfc_dio_port_mask_t     *dio_direction       (port mask)
               # CCURPMFC_DIO_PORT_MASK_P0
               # CCURPMFC_DIO_PORT_MASK_P1
               # CCURPMFC_DIO_PORT_MASK_P2
               # CCURPMFC_DIO_PORT_MASK_P3
```

```
               # CCURPMFC_DIO_PORT_MASK_P4
               # CCURPMFC_DIO_PORT_MASK_P5
               # CCURPMFC_DIO_PORT_MASK_P6
               # CCURPMFC_DIO_PORT_MASK_P7
               # CCURPMFC_DIO_PORT_MASK_P8
               # CCURPMFC_DIO_PORT_MASK_P9
               # CCURPMFC_DIO_PORT_MASK_P10
               # CCURPMFC_DIO_PORT_MASK_P11
               # CCURPMFC_DIO_PORT_MASK_P12
               # CCURPMFC_DIO_PORT_MASK_P13
               # CCURPMFC_DIO_PORT_MASK_P14
               # CCURPMFC_DIO_PORT_MASK_P15
               # CCURPMFC_DIO_PORT_MASK_P16
               # CCURPMFC_DIO_PORT_MASK_P17
               # CCURPMFC_DIO_PORT_MASK_P18
               # CCURPMFC_DIO_PORT_MASK_P19
               # CCURPMFC_DIO_PORT_MASK_P20
               # CCURPMFC_DIO_PORT_MASK_P21
               # CCURPMFC_DIO_PORT_MASK_P22
               # CCURPMFC_DIO_PORT_MASK_P23
               # CCURPMFC_DIO_ALL_PORTS_MASK
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
*************************************************************************/
```

2.2.120 ccurPMFC_DIO_Read_Custom_Channel_Registers()

This call allows the user to display any DIO channels that have been reserved for custom usage. This is specifically dependant on the firmware being loaded. The user must enable the custom mode for DIO operation before reviewing these reserved channels. If no channels are listed, then no custom DIO channels exist. Any custom DIO channel will not operate in the normal DIO functionality but will behave differently based on the firmware loaded.

```
/*************************************************************************
   ccurPMFC_DIO_Read_Custom_Channel_Registers()
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Read_Custom_Channel_Registers(void           *Handle,
                                   ccurpmfc_dio_channels_t  DIO_CustomChannels,
                                   ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Read DIO Custom Channel Registers

   Input:   void                    *Handle            (handle pointer)
            ccurpmfc_dio_channels_t  ChannelSelectMask (custom channel selection)
               # NULL                                  (select all channels)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
Output:  ccurpmfc_dio_channels_t DIO_CustomChannels (custom channel registers)
                # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                    # CCURPMFC_DIO_CHANNEL_MASK_0
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
```

```
                # CCURPMFC_DIO_ALL_CHANNELS_MASK
              CCURPMFC_DIO_MAX_REGISTERS can be one of:
                # CCURPMFC_DIO_CHAN_00_31
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
   Return:  _ccurpmfc_lib_error_number_t
              # CCURPMFC_LIB_NO_ERROR          (successful)
              # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
              # CCURPMFC_LIB_NOT_OPEN          (device not open)
              # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
              # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
              # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.121 ccurPMFC_DIO_Read_Input_Channel_Registers()

This call reads the contents of the input channel registers and returns to the user. There are two modes of operation for this call. (1) Continuous (2) Snapshot.

When the user selects the *continuous* operation, this call immediately returns to the user whatever is available on the input registers as they are being received by the hardware. There is therefore no synchronizing occurring between the three input channel registers (32 channels/register). For performance improvements with *this* operational mode, it is recommended that the user sets the continuous option using the *ccurPMFC_DIO_Set_Input_Snapshot()* call once and then supply *CCURPMFC_DIO_INPUT_OPERATION_DO_NOT_CHANGE* to this call for more reads. In this way, an additional register access will not occur everytime this call is issued.

When the user decides to use the *snapshot* operation instead, there is no need to issue the *ccurPMFC_DIO_Set_Input_Snapshot()*. All that is required is to supply the *CCURPMFC_DIO_INPUT_OPERATION_SNAPSHOT* option when issuing this call. The result is that all the three input registers will be captured instantaneously *(in sync)* by the firmware and returned to the user.

Obviously, the *snapshot* operation is only meaningful if the user selects channels *(using the channel selection mask)* that reside in at least two different input channel registers.

Input channel register 0: Channels 0 to 31
Input channel register 1: Channels 32 to 63
Input channel register 2: Channels 64 to 95

The *skip_dio_disable_check* (when set to *CCURPMFC_FALSE*) causes the call to test for the DIO being enabled prior to proceeding. If this option is set to *CCURPMFC_TRUE*, then no validation is performed. If the DIO has not been enabled, input reads will be invalid. The only reason for providing this option to disable the check in order to improve the performance of the call. If the user can ensure that the DIO is enabled prior to issuing this call, they can set this option to *CCURPMFC_TRUE* so that no validation is performed and hence, improve performance.

When the direction for channels are set to output, then reading the channels input registers will result in acquiring what was written to the output (loopback). When the direction for the channels are set as inputs, then reading the channels input registers will result in acquiring signals coming into the board from the external digital lines.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Read_Input_Channel_Registers(void        *Handle,
                       _ccurpmfc_dio_input_snapshot_t    dio_snapshot,
                       ccurpmfc_bool                     skip_dio_disabled_check,
                       ccurpmfc_dio_channels_t           DIO_InputChannels,
                       ccurpmfc_dio_channels_t           ChannelSelectMask)
```

```
   Description: Read DIO Input Channel Registers

   Input:   void                            *Handle      (handle pointer)
            _ccurpmfc_dio_input_snapshot_t  dio_snapshot (dio_snapshot operation)
               # CCURPMFC_DIO_INPUT_OPERATION_CONTINUOUS
               # CCURPMFC_DIO_INPUT_OPERATION_SNAPSHOT
               # CCURPMFC_DIO_INPUT_OPERATION_DO_NOT_CHANGE
            ccurpmfc_bool  skip_dio_disabled_check      (skip dio disabled check)
               # CCURPMFC_TRUE
               # CCURPMFC_FALSE
            ccurpmfc_dio_channels_t  ChannelSelectMask  (input channel selection)
               # NULL                                   (select all channels)
# u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
                   # CCURPMFC_DIO_CHANNEL_MASK_18
                   # CCURPMFC_DIO_CHANNEL_MASK_19
                   # CCURPMFC_DIO_CHANNEL_MASK_20
                   # CCURPMFC_DIO_CHANNEL_MASK_21
                   # CCURPMFC_DIO_CHANNEL_MASK_22
                   # CCURPMFC_DIO_CHANNEL_MASK_23
                   # CCURPMFC_DIO_CHANNEL_MASK_24
                   # CCURPMFC_DIO_CHANNEL_MASK_25
                   # CCURPMFC_DIO_CHANNEL_MASK_26
                   # CCURPMFC_DIO_CHANNEL_MASK_27
                   # CCURPMFC_DIO_CHANNEL_MASK_28
                   # CCURPMFC_DIO_CHANNEL_MASK_29
                   # CCURPMFC_DIO_CHANNEL_MASK_30
                   # CCURPMFC_DIO_CHANNEL_MASK_31
                   # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                   # CCURPMFC_DIO_CHAN_00_31
                   # CCURPMFC_DIO_CHAN_32_63
                   # CCURPMFC_DIO_CHAN_64_95
   Output:  ccurpmfc_dio_channels_t  DIO_InputChannels  (input channel registers)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
   Return:  _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR          (successful)
                # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN          (device not open)
                # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
                # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ***************************************************************************/
```

2.2.122 ccurPMFC_DIO_Read_Output_Channel_Registers()

This call reads the contents of the output channel registers and returns to the user. This simply represents the contents of the last write to the output registers.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Read_Output_Channel_Registers(void          *Handle,
                                   ccurpmfc_dio_channels_t DIO_OutputChannels,
                                   ccurpmfc_dio_channels_t ChannelSelectMask)

   Description: Read DIO Output Channel Registers

   Input:   void                        *Handle       (handle pointer)
            ccurpmfc_dio_channels_t  ChannelSelectMask (output channel selection)
               # NULL                                  (select all channels)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
Output:  ccurpmfc_dio_channels_t DIO_OutputChannels (output channel registers)
                # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                    # CCURPMFC_DIO_CHANNEL_MASK_0
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
```

```
                # CCURPMFC_DIO_CHANNEL_MASK_28
                # CCURPMFC_DIO_CHANNEL_MASK_29
                # CCURPMFC_DIO_CHANNEL_MASK_30
                # CCURPMFC_DIO_CHANNEL_MASK_31
                # CCURPMFC_DIO_ALL_CHANNELS_MASK
             CCURPMFC_DIO_MAX_REGISTERS can be one of:
                # CCURPMFC_DIO_CHAN_00_31
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
   Return:  _ccurpmfc_lib_error_number_t
             # CCURPMFC_LIB_NO_ERROR          (successful)
             # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
             # CCURPMFC_LIB_NOT_OPEN          (device not open)
             # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
             # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
             # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
**************************************************************************/
```

2.2.123 ccurPMFC_DIO_Set_Channels_Polarity()

This call allows the user to set the polarity for the DIO channels. The *ChannelSelectMask* is used to retrieve polarity settings for selected channels.

For input channels, a value of *CCURPMFC_DIO_INPUT_LOW_TRUE* or *'0'* for polarity indicates low true, while a value of *CCURPMFC_DIO_INPUT_HIGH_TRUE* or *'1'* for polarity indicates high true.

For output channels, a value of *CCURPMFC_DIO_OUTPUT_LOW* or *'0'* for polarity indicates low or 0 volts, while a value of *CCURPMFC_DIO_OUTPUT_HIGH* or *'1'* for polarity indicates high or +5 volts.

```
/**************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Set_Channels_Polarity(void                *Handle,
                                 ccurpmfc_dio_channels_t  DIO_ChannelsPolarity,
                                 ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Set Input Channel Filters

   Input:   void                    *Handle            (handle pointer)
            ccurpmfc_dio_channels_t DIO_ChannelsPolarity(channels polarity regs)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
                   # CCURPMFC_DIO_CHANNEL_MASK_18
                   # CCURPMFC_DIO_CHANNEL_MASK_19
                   # CCURPMFC_DIO_CHANNEL_MASK_20
```

```
                # CCURPMFC_DIO_CHANNEL_MASK_21
                # CCURPMFC_DIO_CHANNEL_MASK_22
                # CCURPMFC_DIO_CHANNEL_MASK_23
                # CCURPMFC_DIO_CHANNEL_MASK_24
                # CCURPMFC_DIO_CHANNEL_MASK_25
                # CCURPMFC_DIO_CHANNEL_MASK_26
                # CCURPMFC_DIO_CHANNEL_MASK_27
                # CCURPMFC_DIO_CHANNEL_MASK_28
                # CCURPMFC_DIO_CHANNEL_MASK_29
                # CCURPMFC_DIO_CHANNEL_MASK_30
                # CCURPMFC_DIO_CHANNEL_MASK_31
                # CCURPMFC_DIO_ALL_CHANNELS_MASK
             CCURPMFC_DIO_MAX_REGISTERS can be one of:
                # CCURPMFC_DIO_CHAN_00_31
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
          ccurpmfc_dio_channels_t ChannelSelectMask (polarity channel
                                                     selection)
             # NULL                                 (select all channels)
             # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                # CCURPMFC_DIO_CHANNEL_MASK_0
                # CCURPMFC_DIO_CHANNEL_MASK_1
                # CCURPMFC_DIO_CHANNEL_MASK_2
                # CCURPMFC_DIO_CHANNEL_MASK_3
                # CCURPMFC_DIO_CHANNEL_MASK_4
                # CCURPMFC_DIO_CHANNEL_MASK_5
                # CCURPMFC_DIO_CHANNEL_MASK_6
                # CCURPMFC_DIO_CHANNEL_MASK_7
                # CCURPMFC_DIO_CHANNEL_MASK_8
                # CCURPMFC_DIO_CHANNEL_MASK_9
                # CCURPMFC_DIO_CHANNEL_MASK_10
                # CCURPMFC_DIO_CHANNEL_MASK_11
                # CCURPMFC_DIO_CHANNEL_MASK_12
                # CCURPMFC_DIO_CHANNEL_MASK_13
                # CCURPMFC_DIO_CHANNEL_MASK_14
                # CCURPMFC_DIO_CHANNEL_MASK_15
                # CCURPMFC_DIO_CHANNEL_MASK_16
                # CCURPMFC_DIO_CHANNEL_MASK_17
                # CCURPMFC_DIO_CHANNEL_MASK_18
                # CCURPMFC_DIO_CHANNEL_MASK_19
                # CCURPMFC_DIO_CHANNEL_MASK_20
                # CCURPMFC_DIO_CHANNEL_MASK_21
                # CCURPMFC_DIO_CHANNEL_MASK_22
                # CCURPMFC_DIO_CHANNEL_MASK_23
                # CCURPMFC_DIO_CHANNEL_MASK_24
                # CCURPMFC_DIO_CHANNEL_MASK_25
                # CCURPMFC_DIO_CHANNEL_MASK_26
                # CCURPMFC_DIO_CHANNEL_MASK_27
                # CCURPMFC_DIO_CHANNEL_MASK_28
                # CCURPMFC_DIO_CHANNEL_MASK_29
                # CCURPMFC_DIO_CHANNEL_MASK_30
                # CCURPMFC_DIO_CHANNEL_MASK_31
                # CCURPMFC_DIO_ALL_CHANNELS_MASK
             CCURPMFC_DIO_MAX_REGISTERS can be one of:
                # CCURPMFC_DIO_CHAN_00_31
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
Output:  none
Return:  _ccurpmfc_lib_error_number_t
             # CCURPMFC_LIB_NO_ERROR          (successful)
             # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
```

```
            # CCURPMFC_LIB_NOT_OPEN          (device not open)
            # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
            # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.124 ccurPMFC_DIO_Set_COS_Channels_Edge_Sense()

This call sets the change-of-state to sense the rising or falling edge of the signal on input for the channels. The *ChannelSelectMask* is used to set the edge sense settings for selected channels. A value of *CCURPMFC_DIO_COS_FALLING_EDGE* or '0' represents sensing of falling edge of input signal while a value of *CCURPMFC_DIO_COS_RISING_EDGE* or '1' represents sensing of rising edge of input signal.

For edge sensing to occur, the *CCURPMFC_DIO_COS_RISING_OR_FALLING_TRANSITION* bit needs to be set for the corresponding channels using the *ccurPMFC_DIO_Set_COS_Channels_Mode()* call.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Set_COS_Channels_Edge_Sense(void       *Handle,
                            ccurpmfc_dio_channels_t  DIO_COS_ChannelsEdgeSense,
                            ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Set COS Channels Edge Sense

   Input:   void                     *Handle               (handle pointer)
            ccurpmfc_dio_channels_t  DIO_COS_ChannelsEdgeSense
                                        (COS channels edge sense registers)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
                   # CCURPMFC_DIO_CHANNEL_MASK_18
                   # CCURPMFC_DIO_CHANNEL_MASK_19
                   # CCURPMFC_DIO_CHANNEL_MASK_20
                   # CCURPMFC_DIO_CHANNEL_MASK_21
                   # CCURPMFC_DIO_CHANNEL_MASK_22
                   # CCURPMFC_DIO_CHANNEL_MASK_23
                   # CCURPMFC_DIO_CHANNEL_MASK_24
                   # CCURPMFC_DIO_CHANNEL_MASK_25
                   # CCURPMFC_DIO_CHANNEL_MASK_26
                   # CCURPMFC_DIO_CHANNEL_MASK_27
                   # CCURPMFC_DIO_CHANNEL_MASK_28
                   # CCURPMFC_DIO_CHANNEL_MASK_29
                   # CCURPMFC_DIO_CHANNEL_MASK_30
                   # CCURPMFC_DIO_CHANNEL_MASK_31
                   # CCURPMFC_DIO_ALL_CHANNELS_MASK
```

```
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
            ccurpmfc_dio_channels_t  ChannelSelectMask   (channel selection)
                # NULL                                   (select all channels)
                # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                    # CCURPMFC_DIO_CHANNEL_MASK_0
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
    Output:  none
    Return:  _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR          (successful)
                # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN          (device not open)
                # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
                # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
*************************************************************************/
```

2.2.125 ccurPMFC_DIO_Set_COS_Channels_Enable()

If the user wishes to monitor change-of-state for a channel, then need to enable the change-of-state detection for the respective channels using this call. Without the channel being enabled, no change-of-state detection will occur. The *ChannelSelectMask* is used to set enable settings for selected channels. A value of *CCURPMFC_DIO_COS_IGNORE* or '0' ignores change-of-state while a value of *CCURPMFC_DIO_COS_ENABLE* or '1' represents enabling change-of-state for the selected channels.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Set_COS_Channels_Enable(void           *Handle,
                            ccurpmfc_dio_channels_t  DIO_COS_ChannelsEnable,
                            ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Set COS Channels Enable

   Input:   void                     *Handle               (handle pointer)
            ccurpmfc_dio_channels_t  DIO_COS_ChannelsEnable
                                        (COS channels enable registers)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
                   # CCURPMFC_DIO_CHANNEL_MASK_18
                   # CCURPMFC_DIO_CHANNEL_MASK_19
                   # CCURPMFC_DIO_CHANNEL_MASK_20
                   # CCURPMFC_DIO_CHANNEL_MASK_21
                   # CCURPMFC_DIO_CHANNEL_MASK_22
                   # CCURPMFC_DIO_CHANNEL_MASK_23
                   # CCURPMFC_DIO_CHANNEL_MASK_24
                   # CCURPMFC_DIO_CHANNEL_MASK_25
                   # CCURPMFC_DIO_CHANNEL_MASK_26
                   # CCURPMFC_DIO_CHANNEL_MASK_27
                   # CCURPMFC_DIO_CHANNEL_MASK_28
                   # CCURPMFC_DIO_CHANNEL_MASK_29
                   # CCURPMFC_DIO_CHANNEL_MASK_30
                   # CCURPMFC_DIO_CHANNEL_MASK_31
                   # CCURPMFC_DIO_ALL_CHANNELS_MASK
               CCURPMFC_DIO_MAX_REGISTERS can be one of:
                   # CCURPMFC_DIO_CHAN_00_31
                   # CCURPMFC_DIO_CHAN_32_63
                   # CCURPMFC_DIO_CHAN_64_95
            ccurpmfc_dio_channels_t  ChannelSelectMask   (channel selection)
               # NULL                                    (select all channels)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
```

```
                # CCURPMFC_DIO_CHANNEL_MASK_8
                # CCURPMFC_DIO_CHANNEL_MASK_9
                # CCURPMFC_DIO_CHANNEL_MASK_10
                # CCURPMFC_DIO_CHANNEL_MASK_11
                # CCURPMFC_DIO_CHANNEL_MASK_12
                # CCURPMFC_DIO_CHANNEL_MASK_13
                # CCURPMFC_DIO_CHANNEL_MASK_14
                # CCURPMFC_DIO_CHANNEL_MASK_15
                # CCURPMFC_DIO_CHANNEL_MASK_16
                # CCURPMFC_DIO_CHANNEL_MASK_17
                # CCURPMFC_DIO_CHANNEL_MASK_18
                # CCURPMFC_DIO_CHANNEL_MASK_19
                # CCURPMFC_DIO_CHANNEL_MASK_20
                # CCURPMFC_DIO_CHANNEL_MASK_21
                # CCURPMFC_DIO_CHANNEL_MASK_22
                # CCURPMFC_DIO_CHANNEL_MASK_23
                # CCURPMFC_DIO_CHANNEL_MASK_24
                # CCURPMFC_DIO_CHANNEL_MASK_25
                # CCURPMFC_DIO_CHANNEL_MASK_26
                # CCURPMFC_DIO_CHANNEL_MASK_27
                # CCURPMFC_DIO_CHANNEL_MASK_28
                # CCURPMFC_DIO_CHANNEL_MASK_29
                # CCURPMFC_DIO_CHANNEL_MASK_30
                # CCURPMFC_DIO_CHANNEL_MASK_31
                # CCURPMFC_DIO_ALL_CHANNELS_MASK
             CCURPMFC_DIO_MAX_REGISTERS can be one of:
                # CCURPMFC_DIO_CHAN_00_31
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.126 ccurPMFC_DIO_Set_COS_Channels_Mode()

This call sets the change-of-state mode registers for all the channels. The *ChannelSelectMask* is used to set the mode settings for selected channels. A value of *CCURPMFC_DIO_COS_ANY_TRANSITION* or '0' sets change-of-state on any edge transition while a value of *CCURPMFC_DIO_COS_RISING_OR_FALLING_TRANSITION* or '1' represents enabling change-of-state for either rising edge or falling edge depending on the channel edge sense setting for the selected channels.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Set_COS_Channels_Mode(void             *Handle,
                            ccurpmfc_dio_channels_t  DIO_COS_ChannelsMode,
                            ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Set COS Channels Mode

   Input:   void                     *Handle               (handle pointer)
            ccurpmfc_dio_channels_t  DIO_COS_ChannelsMode
                                          (COS channels mode registers)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
```

```
            # CCURPMFC_DIO_CHANNEL_MASK_2
            # CCURPMFC_DIO_CHANNEL_MASK_3
            # CCURPMFC_DIO_CHANNEL_MASK_4
            # CCURPMFC_DIO_CHANNEL_MASK_5
            # CCURPMFC_DIO_CHANNEL_MASK_6
            # CCURPMFC_DIO_CHANNEL_MASK_7
            # CCURPMFC_DIO_CHANNEL_MASK_8
            # CCURPMFC_DIO_CHANNEL_MASK_9
            # CCURPMFC_DIO_CHANNEL_MASK_10
            # CCURPMFC_DIO_CHANNEL_MASK_11
            # CCURPMFC_DIO_CHANNEL_MASK_12
            # CCURPMFC_DIO_CHANNEL_MASK_13
            # CCURPMFC_DIO_CHANNEL_MASK_14
            # CCURPMFC_DIO_CHANNEL_MASK_15
            # CCURPMFC_DIO_CHANNEL_MASK_16
            # CCURPMFC_DIO_CHANNEL_MASK_17
            # CCURPMFC_DIO_CHANNEL_MASK_18
            # CCURPMFC_DIO_CHANNEL_MASK_19
            # CCURPMFC_DIO_CHANNEL_MASK_20
            # CCURPMFC_DIO_CHANNEL_MASK_21
            # CCURPMFC_DIO_CHANNEL_MASK_22
            # CCURPMFC_DIO_CHANNEL_MASK_23
            # CCURPMFC_DIO_CHANNEL_MASK_24
            # CCURPMFC_DIO_CHANNEL_MASK_25
            # CCURPMFC_DIO_CHANNEL_MASK_26
            # CCURPMFC_DIO_CHANNEL_MASK_27
            # CCURPMFC_DIO_CHANNEL_MASK_28
            # CCURPMFC_DIO_CHANNEL_MASK_29
            # CCURPMFC_DIO_CHANNEL_MASK_30
            # CCURPMFC_DIO_CHANNEL_MASK_31
            # CCURPMFC_DIO_ALL_CHANNELS_MASK
        CCURPMFC_DIO_MAX_REGISTERS can be one of:
            # CCURPMFC_DIO_CHAN_00_31
            # CCURPMFC_DIO_CHAN_32_63
            # CCURPMFC_DIO_CHAN_64_95
    ccurpmfc_dio_channels_t  ChannelSelectMask   (channel selection)
        # NULL                                   (select all channels)
        # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
            # CCURPMFC_DIO_CHANNEL_MASK_0
            # CCURPMFC_DIO_CHANNEL_MASK_1
            # CCURPMFC_DIO_CHANNEL_MASK_2
            # CCURPMFC_DIO_CHANNEL_MASK_3
            # CCURPMFC_DIO_CHANNEL_MASK_4
            # CCURPMFC_DIO_CHANNEL_MASK_5
            # CCURPMFC_DIO_CHANNEL_MASK_6
            # CCURPMFC_DIO_CHANNEL_MASK_7
            # CCURPMFC_DIO_CHANNEL_MASK_8
            # CCURPMFC_DIO_CHANNEL_MASK_9
            # CCURPMFC_DIO_CHANNEL_MASK_10
            # CCURPMFC_DIO_CHANNEL_MASK_11
            # CCURPMFC_DIO_CHANNEL_MASK_12
            # CCURPMFC_DIO_CHANNEL_MASK_13
            # CCURPMFC_DIO_CHANNEL_MASK_14
            # CCURPMFC_DIO_CHANNEL_MASK_15
            # CCURPMFC_DIO_CHANNEL_MASK_16
            # CCURPMFC_DIO_CHANNEL_MASK_17
            # CCURPMFC_DIO_CHANNEL_MASK_18
            # CCURPMFC_DIO_CHANNEL_MASK_19
            # CCURPMFC_DIO_CHANNEL_MASK_20
            # CCURPMFC_DIO_CHANNEL_MASK_21
            # CCURPMFC_DIO_CHANNEL_MASK_22
```

```
                # CCURPMFC_DIO_CHANNEL_MASK_23
                # CCURPMFC_DIO_CHANNEL_MASK_24
                # CCURPMFC_DIO_CHANNEL_MASK_25
                # CCURPMFC_DIO_CHANNEL_MASK_26
                # CCURPMFC_DIO_CHANNEL_MASK_27
                # CCURPMFC_DIO_CHANNEL_MASK_28
                # CCURPMFC_DIO_CHANNEL_MASK_29
                # CCURPMFC_DIO_CHANNEL_MASK_30
                # CCURPMFC_DIO_CHANNEL_MASK_31
                # CCURPMFC_DIO_ALL_CHANNELS_MASK
             CCURPMFC_DIO_MAX_REGISTERS can be one of:
                # CCURPMFC_DIO_CHAN_00_31
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.127 ccurPMFC_DIO_Set_Input_Channels_Filter()

This call allows the user to set or reset filters for a selected set of input channels. The *ChannelSelectMask* is used to select channels for filter settings. A value of *CCURPMFC_DIO_INPUT_FILTER_ENABLED* or *'1'* for filter indicates that the 100 nanosecond filter is enabled for the selected channel, while a value of *CCURPMFC_DIO_INPUT_FILTER_DISABLED* or *'0'* indicates that the filter is disabled. On powerup, filter for all channels are enabled.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Set_Input_Channels_Filter(void                *Handle,
                                  ccurpmfc_dio_channels_t  DIO_InputChannels,
                                  ccurpmfc_dio_channels_t  ChannelSelectMask)

   Description: Set Input Channel Filters

   Input:   void                    *Handle             (handle pointer)
            ccurpmfc_dio_channels_t DIO_InputChannels   (input channel registers)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
```

```
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
ccurpmfc_dio_channels_t ChannelSelectMask             (filter channel selection)
                # NULL                                (select all channels)
                # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                    # CCURPMFC_DIO_CHANNEL_MASK_0
                    # CCURPMFC_DIO_CHANNEL_MASK_1
                    # CCURPMFC_DIO_CHANNEL_MASK_2
                    # CCURPMFC_DIO_CHANNEL_MASK_3
                    # CCURPMFC_DIO_CHANNEL_MASK_4
                    # CCURPMFC_DIO_CHANNEL_MASK_5
                    # CCURPMFC_DIO_CHANNEL_MASK_6
                    # CCURPMFC_DIO_CHANNEL_MASK_7
                    # CCURPMFC_DIO_CHANNEL_MASK_8
                    # CCURPMFC_DIO_CHANNEL_MASK_9
                    # CCURPMFC_DIO_CHANNEL_MASK_10
                    # CCURPMFC_DIO_CHANNEL_MASK_11
                    # CCURPMFC_DIO_CHANNEL_MASK_12
                    # CCURPMFC_DIO_CHANNEL_MASK_13
                    # CCURPMFC_DIO_CHANNEL_MASK_14
                    # CCURPMFC_DIO_CHANNEL_MASK_15
                    # CCURPMFC_DIO_CHANNEL_MASK_16
                    # CCURPMFC_DIO_CHANNEL_MASK_17
                    # CCURPMFC_DIO_CHANNEL_MASK_18
                    # CCURPMFC_DIO_CHANNEL_MASK_19
                    # CCURPMFC_DIO_CHANNEL_MASK_20
                    # CCURPMFC_DIO_CHANNEL_MASK_21
                    # CCURPMFC_DIO_CHANNEL_MASK_22
                    # CCURPMFC_DIO_CHANNEL_MASK_23
                    # CCURPMFC_DIO_CHANNEL_MASK_24
                    # CCURPMFC_DIO_CHANNEL_MASK_25
                    # CCURPMFC_DIO_CHANNEL_MASK_26
                    # CCURPMFC_DIO_CHANNEL_MASK_27
                    # CCURPMFC_DIO_CHANNEL_MASK_28
                    # CCURPMFC_DIO_CHANNEL_MASK_29
                    # CCURPMFC_DIO_CHANNEL_MASK_30
                    # CCURPMFC_DIO_CHANNEL_MASK_31
                    # CCURPMFC_DIO_ALL_CHANNELS_MASK
                CCURPMFC_DIO_MAX_REGISTERS can be one of:
                    # CCURPMFC_DIO_CHAN_00_31
                    # CCURPMFC_DIO_CHAN_32_63
                    # CCURPMFC_DIO_CHAN_64_95
    Output:  none
    Return:  _ccurpmfc_lib_error_number_t
```

```
            # CCURPMFC_LIB_NO_ERROR          (successful)
            # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN          (device not open)
            # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
            # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.128 ccurPMFC_DIO_Set_Input_Snapshot()

This call allows the user to set the board in snapshot mode where all three input registers are acquired simultaneously (in sync) by the hardware and presented to the user. Mainly, this particular call is only useful for setting the operation to continuous mode. There is no need to set to snapshot mode as the read input registers call *ccurPMFC_DIO_Read_Input_Channel_Registers()* has an option to set it in the call.

If the user wants to collect data in the continuous mode, they should issue this call once with the *CCURPMFC_DIO_INPUT_OPERATION_CONTINUOUS* option and then call the read of the input channels with the *CCURPMFC_DO_NOT_CHANGE* option. In this way, there is no un-necessary overhead in setting the board into continuous mode once it has already been set.

Recommended procedure for continuous mode is to issue this call only once with the *CCURPMFC_DIO_INPUT_OPERATION_CONTINUOUS* option and then followup with continuous input channel reads using the *ccurPMFC_DIO_Read_Input_Channel_Register()* call with the *CCURPMFC_DIO_INPUT_OPERATION_DO_NOT_CHANGE* option for *dio_snapshot.*

Recommended procedure for snapshot mode is to issue continuous input channel reads using the *ccurPMFC_DIO_Read_Input_Channel_Register()* call with the *CCURPMFC_DIO_INPUT_OPERATION_SNAPSHOT* option for *dio_snapshot.* In this case there is really no need to issue this *ccurPMFC_DIO_Set_Input_Snapshot()* call.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Set_Input_Snapshot(void                            *Handle,
                                   _ccurpmfc_dio_input_snapshot_t  dio_snapshot)

   Description: Set DIO Input Snapshot

   Input:   void                        *Handle                  (handle pointer)
   Output:  _ccurpmfc_dio_input_snapshot_t  dio_snapshot         (dio snapshot)
               # CCURPMFC_DIO_INPUT_OPERATION_CONTINUOUS
               # CCURPMFC_DIO_INPUT_OPERATION_SNAPSHOT
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.129 ccurPMFC_DIO_Set_Mode()

This call allows the user to select a custom or a normal mode of DIO operation. In the normal mode, all 96 channels are available for DIO operation, while, in the custom mode, the custom channels supplied in the custom input registers may not operate as normal DIO channels. These custom channels behavior will depend entirely on the firmware loaded into the board.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Set_Mode(void                *Handle,
```

```
                              _ccurpmfc_dio_mode_t dio_mode)

   Description: Set DIO Mode

   Input:   void                    *Handle                   (handle pointer)
   Output:  _ccurpmfc_dio_mode_t    dio_mode                  (dio mode)
               # CCURPMFC_DIO_MODE_CUSTOM
               # CCURPMFC_DIO_MODE_NORMAL
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
*************************************************************************/
```

2.2.130 ccurPMFC_DIO_Set_Output_Sync()

This call allows the user to set the digital output channels to either *continuous* mode or *sync* mode. When the board is in *continuous* mode, any update to one of the three output registers will be immediately sent to the output lines. There will be no synchronization between the three output registers. When the sync mode is selected, no output is sent while updating the output registers. Once the output sync flag is set, the contents of all three output registers will be sent simultaneously to the output lines.

Recommended procedure for *continuous* mode is to issue this call only once with the *CCURPMFC_DIO_OUTPUT_OPERATION_CONTINUOUS* option and then followup with continuous output channel writes using the *ccurPMFC_DIO_Write_Output_Channel_Register()* call with the *CCURPMFC_DIO_INPUT_OPERATION_DO_NOT_CHANGE* option for *dio_sync.*

Recommended procedure for *sync* mode is to issue this call only once with the *CCURPMFC_DIO_OUTPUT_OPERATION_SYNC* option and then followed up with continuous output channel writes using the *ccurPMFC_DIO_Write_Output_Channel_Register()* call with the *CCURPMFC_DIO_OUTPUT_OPERATION_SYNC* option for *dio_sync.*

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Set_Output_Sync(void                          *Handle,
                                _ccurpmfc_dio_output_sync_t   dio_sync)

   Description: Set DIO Output Sync

   Input:   void                           *Handle            (handle pointer)
            _ccurpmfc_dio_output_sync_t    dio_sync           (dio sync)
               # CCURPMFC_DIO_OUTPUT_OPERATION_CONTINUOUS
               # CCURPMFC_DIO_OUTPUT_OPERATION_SYNC
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
*************************************************************************/
```

2.2.131 ccurPMFC_DIO_Set_Ports_Direction()

This call allows the user to set the direction of the digital channels. There are 24 ports with grouping of 4 channels per port. Direction control is on a port level or a group of 4 channels.

When the direction for channels are set to output, then reading the channels input registers will result in acquiring what was written to the output (readback). When the direction for the channels are set as inputs, then reading the channels input registers will result in acquiring signals coming into the board from the external digital lines.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Set_Ports_Direction(void                    *Handle,
                                    _ccurpmfc_dio_port_mask_t dio_direction)

   Description: Set DIO Port Direction Mask

   Input:   void                          *Handle                (handle pointer)
            _ccurpmfc_dio_port_mask_t     dio_direction          (port mask)
               # CCURPMFC_DIO_PORT_MASK_P0
               # CCURPMFC_DIO_PORT_MASK_P1
               # CCURPMFC_DIO_PORT_MASK_P2
               # CCURPMFC_DIO_PORT_MASK_P3
               # CCURPMFC_DIO_PORT_MASK_P4
               # CCURPMFC_DIO_PORT_MASK_P5
               # CCURPMFC_DIO_PORT_MASK_P6
               # CCURPMFC_DIO_PORT_MASK_P7
               # CCURPMFC_DIO_PORT_MASK_P8
               # CCURPMFC_DIO_PORT_MASK_P9
               # CCURPMFC_DIO_PORT_MASK_P10
               # CCURPMFC_DIO_PORT_MASK_P11
               # CCURPMFC_DIO_PORT_MASK_P12
               # CCURPMFC_DIO_PORT_MASK_P13
               # CCURPMFC_DIO_PORT_MASK_P14
               # CCURPMFC_DIO_PORT_MASK_P15
               # CCURPMFC_DIO_PORT_MASK_P16
               # CCURPMFC_DIO_PORT_MASK_P17
               # CCURPMFC_DIO_PORT_MASK_P18
               # CCURPMFC_DIO_PORT_MASK_P19
               # CCURPMFC_DIO_PORT_MASK_P20
               # CCURPMFC_DIO_PORT_MASK_P21
               # CCURPMFC_DIO_PORT_MASK_P22
               # CCURPMFC_DIO_PORT_MASK_P23
               # CCURPMFC_DIO_ALL_PORTS_MASK
   Output:     none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.132 ccurPMFC_DIO_Set_Ports_Direction_To_Input()

This call allows the user to set the direction of a selected set of digital channels to Inputs. Other channels are unchanged. There are 24 ports with grouping of 4 channels per port. Direction Input control is on a port level or a group of 4 channels.

When the direction for the channels are set as inputs, then reading the channels input registers will result in acquiring signals coming into the board from the external digital lines.

```
/************************************************************************
```

```
   _ccurpmfc_lib_error_number_t
       ccurPMFC_DIO_Set_Ports_Direction_To_Input(void      *Handle,
                                _ccurpmfc_dio_port_mask_t dio_input_direction)

   Description: Set DIO Port Direction Input Mask

   Input:  void                     *Handle                      (handle pointer)
           _ccurpmfc_dio_port_mask_t dio_input_direction         (port mask)
               # CCURPMFC_DIO_PORT_MASK_P0
               # CCURPMFC_DIO_PORT_MASK_P1
               # CCURPMFC_DIO_PORT_MASK_P2
               # CCURPMFC_DIO_PORT_MASK_P3
               # CCURPMFC_DIO_PORT_MASK_P4
               # CCURPMFC_DIO_PORT_MASK_P5
               # CCURPMFC_DIO_PORT_MASK_P6
               # CCURPMFC_DIO_PORT_MASK_P7
               # CCURPMFC_DIO_PORT_MASK_P8
               # CCURPMFC_DIO_PORT_MASK_P9
               # CCURPMFC_DIO_PORT_MASK_P10
               # CCURPMFC_DIO_PORT_MASK_P11
               # CCURPMFC_DIO_PORT_MASK_P12
               # CCURPMFC_DIO_PORT_MASK_P13
               # CCURPMFC_DIO_PORT_MASK_P14
               # CCURPMFC_DIO_PORT_MASK_P15
               # CCURPMFC_DIO_PORT_MASK_P16
               # CCURPMFC_DIO_PORT_MASK_P17
               # CCURPMFC_DIO_PORT_MASK_P18
               # CCURPMFC_DIO_PORT_MASK_P19
               # CCURPMFC_DIO_PORT_MASK_P20
               # CCURPMFC_DIO_PORT_MASK_P21
               # CCURPMFC_DIO_PORT_MASK_P22
               # CCURPMFC_DIO_PORT_MASK_P23
               # CCURPMFC_DIO_ALL_PORTS_MASK
   Output:     none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
*************************************************************************/
```

2.2.133 ccurPMFC_DIO_Set_Ports_Direction_To_Output()

This call allows the user to set the direction of a selected set of digital channels to Outputs. Other channels are unchanged. There are 24 ports with grouping of 4 channels per port. Direction Output control is on a port level or a group of 4 channels.

When the direction for channels are set to output, then reading the channels input registers will result in acquiring what was written to the output (readback).

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
       ccurPMFC_DIO_Set_Ports_Direction_To_Output(void     *Handle,
                                _ccurpmfc_dio_port_mask_t dio_output_direction)

   Description: Set DIO Port Direction Input Mask

   Input:  void                     *Handle                      (handle pointer)
           _ccurpmfc_dio_port_mask_t dio_outputt_direction       (port mask)
```

```
                # CCURPMFC_DIO_PORT_MASK_P0
                # CCURPMFC_DIO_PORT_MASK_P1
                # CCURPMFC_DIO_PORT_MASK_P2
                # CCURPMFC_DIO_PORT_MASK_P3
                # CCURPMFC_DIO_PORT_MASK_P4
                # CCURPMFC_DIO_PORT_MASK_P5
                # CCURPMFC_DIO_PORT_MASK_P6
                # CCURPMFC_DIO_PORT_MASK_P7
                # CCURPMFC_DIO_PORT_MASK_P8
                # CCURPMFC_DIO_PORT_MASK_P9
                # CCURPMFC_DIO_PORT_MASK_P10
                # CCURPMFC_DIO_PORT_MASK_P11
                # CCURPMFC_DIO_PORT_MASK_P12
                # CCURPMFC_DIO_PORT_MASK_P13
                # CCURPMFC_DIO_PORT_MASK_P14
                # CCURPMFC_DIO_PORT_MASK_P15
                # CCURPMFC_DIO_PORT_MASK_P16
                # CCURPMFC_DIO_PORT_MASK_P17
                # CCURPMFC_DIO_PORT_MASK_P18
                # CCURPMFC_DIO_PORT_MASK_P19
                # CCURPMFC_DIO_PORT_MASK_P20
                # CCURPMFC_DIO_PORT_MASK_P21
                # CCURPMFC_DIO_PORT_MASK_P22
                # CCURPMFC_DIO_PORT_MASK_P23
                # CCURPMFC_DIO_ALL_PORTS_MASK
   Output:      none
   Return:   _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR          (successful)
                # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN          (device not open)
                # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
                # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
*************************************************************************/
```

2.2.134 ccurPMFC_DIO_Write_Output_Channel_High_Registers()

This call writes a selected set of channels to high outputs. Rest of the channels are not affected. There are two modes of operation for this call. (1) Continuous (2) Sync.

When the user selects the *continuous* operation, this call immediately sends out the channel high data to the selected output lines as they are written to the output registers. There is therefore no synchronizing occurring between the three output channel registers (32 channels/register). For performance improvements with this operational mode, it is recommended that the user sets the continuous option using the *ccurPMFC_DIO_Set_Output_Sync()* call once and then supply *CCURPMFC_DIO_OUTPUT_OPERATION_DO_NOT_CHANGE* to this call for more writes. In this way, an additional register access will not occur everytime this call is issued.

When the user selects the *sync* operation, they need to issue the *ccurPMFC_DIO_Set_Output_Sync()* call once with the *CCURPMFC_DIO_OUTPUT_OPERATION*_SYNC option, followed by issuing this call with the *CCURPMFC_DIO_OUTPUT_OPERATION*_SYNC option in *dio_sync.*

Obviously, the *sync* option is only meaningful if the user selects channels *(sets channels for high)* that reside in at least two different output channel registers.

Output channel register 0: Channels 0 to 31
Output channel register 1: Channels 32 to 63
Output channel register 2: Channels 64 to 95

The *skip_dio_disable_check (*when set to *CCURPMFC_FALSE)* causes the call to test for DIO being enabled prior to proceeding. If this option is set to *CCURPMFC_TRUE*, then no validation is performed. If the DIO has not been enabled, output writes will not take place. The only reason for providing the option to disable the check is to improve the performance of the call. If the user can ensure that the DIO is enabled prior to issuing this call, they can set this option to *CCURPMFC_TRUE* so that no validation is performed and hence, improve performance.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Write_Output_Channel_High_Registers(void *Handle,
                           _ccurpmfc_dio_output_sync_t   dio_sync,
                           ccurpmfc_bool                 skip_dio_disabled_check,
                           ccurpmfc_dio_channels_t       DIO_OutputChannels)

   Description: Write DIO Output Channel High Registers

   Input:   void                          *Handle             (handle pointer)
            _ccurpmfc_dio_output_sync_t  dio_sync            (dio_sync operation)
               # CCURPMFC_DIO_OUTPUT_OPERATION_CONTINUOUS
               # CCURPMFC_DIO_OUTPUT_OPERATION_SYNC
               # CCURPMFC_DIO_OUTPUT_OPERATION_DO_NOT_CHANGE
            ccurpmfc_bool      skip_dio_disabled_check  (skip dio disabled check)
               # CCURPMFC_TRUE
               # CCURPMFC_FALSE
            ccurpmfc_dio_channels_t DIO_OutputChannels (output channel registers)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
                   # CCURPMFC_DIO_CHANNEL_MASK_18
                   # CCURPMFC_DIO_CHANNEL_MASK_19
                   # CCURPMFC_DIO_CHANNEL_MASK_20
                   # CCURPMFC_DIO_CHANNEL_MASK_21
                   # CCURPMFC_DIO_CHANNEL_MASK_22
                   # CCURPMFC_DIO_CHANNEL_MASK_23
                   # CCURPMFC_DIO_CHANNEL_MASK_24
                   # CCURPMFC_DIO_CHANNEL_MASK_25
                   # CCURPMFC_DIO_CHANNEL_MASK_26
                   # CCURPMFC_DIO_CHANNEL_MASK_27
                   # CCURPMFC_DIO_CHANNEL_MASK_28
                   # CCURPMFC_DIO_CHANNEL_MASK_29
                   # CCURPMFC_DIO_CHANNEL_MASK_30
                   # CCURPMFC_DIO_CHANNEL_MASK_31
                   # CCURPMFC_DIO_ALL_CHANNELS_MASK
               CCURPMFC_DIO_MAX_REGISTERS can be one of:
                   # CCURPMFC_DIO_CHAN_00_31
```

```
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.135 ccurPMFC_DIO_Write_Output_Channel_Low_Registers()

This call writes a selected set of channels to low outputs. Rest of the channels are not affected. There are two modes of operation for this call. (1) Continuous (2) Sync.

When the user selects the *continuous* operation, this call immediately sends out the channel low data to the output lines as they are written to the output registers. There is therefore no synchronizing occurring between the three output channel registers (32 channels/register). For performance improvements with this operational mode, it is recommended that the user sets the *continuous* option using the *ccurPMFC_DIO_Set_Output_Sync()* call once and then supply *CCURPMFC_DIO_OUTPUT_OPERATION_DO_NOT_CHANGE* to this call for more writes. In this way, an additional register access will not occur everytime this call is issued.

When the user selects the *sync* operation, they need to issue the *ccurPMFC_DIO_Set_Output_Sync()* call once with the *CCURPMFC_DIO_OUTPUT_OPERATION*_SYNC option, followed by issuing this call with the *CCURPMFC_DIO_OUTPUT_OPERATION*_SYNC option in *dio_sync*.

Obviously, the *sync* option is only meaningful if the user selects channels *(sets channels for low)* that reside in at least two different output channel registers.

Output channel register 0: Channels 0 to 31
Output channel register 1: Channels 32 to 63
Output channel register 2: Channels 64 to 95

The *skip_dio_disable_check* (when set to *CCURPMFC_FALSE*) causes the call to test for DIO being enabled prior to proceeding. If this option is set to *CCURPMFC_TRUE*, then no validation is performed. If the DIO has not been enabled, output writes will not take place. The only reason for providing the option to disable the check is to improve the performance of the call. If the user can ensure that the DIO is enabled prior to issuing this call, they can set this option to *CCURPMFC_TRUE* so that no validation is performed and hence, improve performance.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Write_Output_Channel_Low_Registers(void  *Handle,
                           _ccurpmfc_dio_output_sync_t dio_sync,
                           ccurpmfc_bool               skip_dio_disabled_check,
                           ccurpmfc_dio_channels_t     DIO_OutputChannels)

   Description: Write DIO Output Channel Low Registers

   Input:   void                         *Handle             (handle pointer)
            _ccurpmfc_dio_output_sync_t  dio_sync            (dio_sync operation)
               # CCURPMFC_DIO_OUTPUT_OPERATION_CONTINUOUS
               # CCURPMFC_DIO_OUTPUT_OPERATION_SYNC
               # CCURPMFC_DIO_OUTPUT_OPERATION_DO_NOT_CHANGE
            ccurpmfc_bool      skip_dio_disabled_check  (skip dio disabled check)
               # CCURPMFC_TRUE
```

```
            # CCURPMFC_FALSE
          ccurpmfc_dio_channels_t DIO_OutputChannels (output channel registers)
            # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                # CCURPMFC_DIO_CHANNEL_MASK_0
                # CCURPMFC_DIO_CHANNEL_MASK_1
                # CCURPMFC_DIO_CHANNEL_MASK_2
                # CCURPMFC_DIO_CHANNEL_MASK_3
                # CCURPMFC_DIO_CHANNEL_MASK_4
                # CCURPMFC_DIO_CHANNEL_MASK_5
                # CCURPMFC_DIO_CHANNEL_MASK_6
                # CCURPMFC_DIO_CHANNEL_MASK_7
                # CCURPMFC_DIO_CHANNEL_MASK_8
                # CCURPMFC_DIO_CHANNEL_MASK_9
                # CCURPMFC_DIO_CHANNEL_MASK_10
                # CCURPMFC_DIO_CHANNEL_MASK_11
                # CCURPMFC_DIO_CHANNEL_MASK_12
                # CCURPMFC_DIO_CHANNEL_MASK_13
                # CCURPMFC_DIO_CHANNEL_MASK_14
                # CCURPMFC_DIO_CHANNEL_MASK_15
                # CCURPMFC_DIO_CHANNEL_MASK_16
                # CCURPMFC_DIO_CHANNEL_MASK_17
                # CCURPMFC_DIO_CHANNEL_MASK_18
                # CCURPMFC_DIO_CHANNEL_MASK_19
                # CCURPMFC_DIO_CHANNEL_MASK_20
                # CCURPMFC_DIO_CHANNEL_MASK_21
                # CCURPMFC_DIO_CHANNEL_MASK_22
                # CCURPMFC_DIO_CHANNEL_MASK_23
                # CCURPMFC_DIO_CHANNEL_MASK_24
                # CCURPMFC_DIO_CHANNEL_MASK_25
                # CCURPMFC_DIO_CHANNEL_MASK_26
                # CCURPMFC_DIO_CHANNEL_MASK_27
                # CCURPMFC_DIO_CHANNEL_MASK_28
                # CCURPMFC_DIO_CHANNEL_MASK_29
                # CCURPMFC_DIO_CHANNEL_MASK_30
                # CCURPMFC_DIO_CHANNEL_MASK_31
                # CCURPMFC_DIO_ALL_CHANNELS_MASK
            CCURPMFC_DIO_MAX_REGISTERS can be one of:
                # CCURPMFC_DIO_CHAN_00_31
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
  Output:  none
  Return:  _ccurpmfc_lib_error_number_t
            # CCURPMFC_LIB_NO_ERROR          (successful)
            # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN          (device not open)
            # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
            # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
*************************************************************************/
```

2.2.136 ccurPMFC_DIO_Write_Output_Channel_Registers()

This call writes the user supplied channel settings to the output channel registers. There are two modes of operation for this call. (1) Continuous (2) Sync.

When the user selects the *continuous* operation, this call immediately sends out the channel data to the output lines as they are written to the output registers. There is therefore no synchronizing occurring between the three output channel registers (32 channels/register). For performance improvements with this operational mode, it is recommended that the user sets the *continuous* option using the *ccurPMFC_DIO_Set_Output_Sync()* call once and then supply

CCURPMFC_DIO_OUTPUT_OPERATION_DO_NOT_CHANGE to this call for more writes. In this way, an additional register access will not occur everytime this call is issued.

When the user selects the *sync* operation, they need to issue the *ccurPMFC_DIO_Set_Output_Sync()* call once with the *CCURPMFC_DIO_OUTPUT_OPERATION*_SYNC option, followed by issuing this call with the *CCURPMFC_DIO_OUTPUT_OPERATION*_SYNC option in *dio_sync*.

Obviously, the *sync* option is only meaningful if the user selects channels *(using the channel selection mask)* that reside in at least two different output channel registers.

Output channel register 0: Channels 0 to 31
Output channel register 1: Channels 32 to 63
Output channel register 2: Channels 64 to 95

The *skip_dio_disable_check* (when set to *CCURPMFC_FALSE*) causes the call to test for DIO being enabled prior to proceeding. If this option is set to *CCURPMFC_TRUE*, then no validation is performed. If the DIO has not been enabled, output writes will not take place. The only reason for providing the option to disable the check is to improve the performance of the call. If the user can ensure that the DIO is enabled prior to issuing this call, they can set this option to *CCURPMFC_TRUE* so that no test is performed and hence, improve performance.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DIO_Write_Output_Channel_Registers(void     *Handle,
                          _ccurpmfc_dio_output_sync_t  dio_sync,
                          ccurpmfc_bool                skip_dio_disabled_check,
                          ccurpmfc_dio_channels_t      DIO_OutputChannels,
                          ccurpmfc_dio_channels_t      ChannelSelectMask)

   Description: Write DIO Output Channel Registers

   Input:   void                          *Handle             (handle pointer)
            _ccurpmfc_dio_output_sync_t  dio_sync            (dio_sync operation)
               # CCURPMFC_DIO_OUTPUT_OPERATION_CONTINUOUS
               # CCURPMFC_DIO_OUTPUT_OPERATION_SYNC
               # CCURPMFC_DIO_OUTPUT_OPERATION_DO_NOT_CHANGE
            ccurpmfc_bool      skip_dio_disabled_check  (skip dio disabled check)
               # CCURPMFC_TRUE
               # CCURPMFC_FALSE
            ccurpmfc_dio_channels_t DIO_OutputChannels (output channel registers)
               # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                   # CCURPMFC_DIO_CHANNEL_MASK_0
                   # CCURPMFC_DIO_CHANNEL_MASK_1
                   # CCURPMFC_DIO_CHANNEL_MASK_2
                   # CCURPMFC_DIO_CHANNEL_MASK_3
                   # CCURPMFC_DIO_CHANNEL_MASK_4
                   # CCURPMFC_DIO_CHANNEL_MASK_5
                   # CCURPMFC_DIO_CHANNEL_MASK_6
                   # CCURPMFC_DIO_CHANNEL_MASK_7
                   # CCURPMFC_DIO_CHANNEL_MASK_8
                   # CCURPMFC_DIO_CHANNEL_MASK_9
                   # CCURPMFC_DIO_CHANNEL_MASK_10
                   # CCURPMFC_DIO_CHANNEL_MASK_11
                   # CCURPMFC_DIO_CHANNEL_MASK_12
                   # CCURPMFC_DIO_CHANNEL_MASK_13
                   # CCURPMFC_DIO_CHANNEL_MASK_14
                   # CCURPMFC_DIO_CHANNEL_MASK_15
                   # CCURPMFC_DIO_CHANNEL_MASK_16
                   # CCURPMFC_DIO_CHANNEL_MASK_17
```

```
                # CCURPMFC_DIO_CHANNEL_MASK_18
                # CCURPMFC_DIO_CHANNEL_MASK_19
                # CCURPMFC_DIO_CHANNEL_MASK_20
                # CCURPMFC_DIO_CHANNEL_MASK_21
                # CCURPMFC_DIO_CHANNEL_MASK_22
                # CCURPMFC_DIO_CHANNEL_MASK_23
                # CCURPMFC_DIO_CHANNEL_MASK_24
                # CCURPMFC_DIO_CHANNEL_MASK_25
                # CCURPMFC_DIO_CHANNEL_MASK_26
                # CCURPMFC_DIO_CHANNEL_MASK_27
                # CCURPMFC_DIO_CHANNEL_MASK_28
                # CCURPMFC_DIO_CHANNEL_MASK_29
                # CCURPMFC_DIO_CHANNEL_MASK_30
                # CCURPMFC_DIO_CHANNEL_MASK_31
                # CCURPMFC_DIO_ALL_CHANNELS_MASK
            CCURPMFC_DIO_MAX_REGISTERS can be one of:
                # CCURPMFC_DIO_CHAN_00_31
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
        ccurpmfc_dio_channels_t  ChannelSelectMask (output channel selection)
            # NULL                                  (select all channels)
            # u_int32_t   ccurpmfc_dio_channels_t[CCURPMFC_DIO_MAX_REGISTERS]
                # CCURPMFC_DIO_CHANNEL_MASK_0
                # CCURPMFC_DIO_CHANNEL_MASK_1
                # CCURPMFC_DIO_CHANNEL_MASK_2
                # CCURPMFC_DIO_CHANNEL_MASK_3
                # CCURPMFC_DIO_CHANNEL_MASK_4
                # CCURPMFC_DIO_CHANNEL_MASK_5
                # CCURPMFC_DIO_CHANNEL_MASK_6
                # CCURPMFC_DIO_CHANNEL_MASK_7
                # CCURPMFC_DIO_CHANNEL_MASK_8
                # CCURPMFC_DIO_CHANNEL_MASK_9
                # CCURPMFC_DIO_CHANNEL_MASK_10
                # CCURPMFC_DIO_CHANNEL_MASK_11
                # CCURPMFC_DIO_CHANNEL_MASK_12
                # CCURPMFC_DIO_CHANNEL_MASK_13
                # CCURPMFC_DIO_CHANNEL_MASK_14
                # CCURPMFC_DIO_CHANNEL_MASK_15
                # CCURPMFC_DIO_CHANNEL_MASK_16
                # CCURPMFC_DIO_CHANNEL_MASK_17
                # CCURPMFC_DIO_CHANNEL_MASK_18
                # CCURPMFC_DIO_CHANNEL_MASK_19
                # CCURPMFC_DIO_CHANNEL_MASK_20
                # CCURPMFC_DIO_CHANNEL_MASK_21
                # CCURPMFC_DIO_CHANNEL_MASK_22
                # CCURPMFC_DIO_CHANNEL_MASK_23
                # CCURPMFC_DIO_CHANNEL_MASK_24
                # CCURPMFC_DIO_CHANNEL_MASK_25
                # CCURPMFC_DIO_CHANNEL_MASK_26
                # CCURPMFC_DIO_CHANNEL_MASK_27
                # CCURPMFC_DIO_CHANNEL_MASK_28
                # CCURPMFC_DIO_CHANNEL_MASK_29
                # CCURPMFC_DIO_CHANNEL_MASK_30
                # CCURPMFC_DIO_CHANNEL_MASK_31
                # CCURPMFC_DIO_ALL_CHANNELS_MASK
            CCURPMFC_DIO_MAX_REGISTERS can be one of:
                # CCURPMFC_DIO_CHAN_00_31
                # CCURPMFC_DIO_CHAN_32_63
                # CCURPMFC_DIO_CHAN_64_95
Output:  none
Return:  _ccurpmfc_lib_error_number_t
```

```
            # CCURPMFC_LIB_NO_ERROR          (successful)
            # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN          (device not open)
            # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
            # CCURPMFC_LIB_DIO_IS_NOT_ACTIVE (DIO is not active)
 ************************************************************************/
```

2.2.137 ccurPMFC_Disable_Pci_Interrupts()

The purpose of this call is to disable PCI interrupts. This call shouldn't be used during normal reads as calls could time out. The driver handles enabling and disabling interrupts during its normal course of operation.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Disable_Pci_Interrupts(void                          *Handle,
                                   _ccurpmfc_all_interrupts_mask interrupt_mask))

   Description: Disable interrupts being generated by the board.

   Input:   void                          *Handle         (Handle pointer)
            _ccurpmfc_all_interrupts_mask interrupt_mask  (interrupt mask)
               # CCURPMFC_DMA0_INTMASK
               # CCURPMFC_DMA1_INTMASK
               # CCURPMFC_MSGDMA_INTMASK
               # CCURPMFC_ADC_FIFO_INTMASK
               # CCURPMFC_DAC_FIFO_INTMASK
               # CCURPMFC_DIO_GROUP0_INTMASK
               # CCURPMFC_DIO_GROUP1_INTMASK
               # CCURPMFC_DIO_GROUP2_INTMASK
               # CCURPMFC_ALL_DMA_INTMASK
               # CCURPMFC_ALL_ANALOG_INTMASK
               # CCURPMFC_ALL_DIO_INTMASK
               # CCURPMFC_DMA_ANALOG_INTMASK
               # CCURPMFC_ALL_INTMASK
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_IOCTL_FAILED            (driver ioctl call
                                                       failed)
 ************************************************************************/
```

2.2.138 ccurPMFC_DMA_Configure()

The purpose of this call is configure a DMA engine to be ready for commencing DMA.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DMA_Configure(void                    *Handle,
                          ccurpmfc_dma_engine_t   DMAEngineNo,
                          uint                    AvMM_FromAddr,
                          uint                    AvMM_ToAddr,
                          uint                    DMASize)

   Description: Configure DMA Engine

   Input:   void                 *Handle       (Handle pointer)
            ccurpmfc_dma_engine_t DMAEngineNo   (select DMA engine)
               # CCURPMFC_DMA0
```

```
            # CCURPMFC_DMA1
          uint                  AvMM_FromAddr (Avalon Memory Converted Source
                                               Address)
          uint                  AvMM_ToAddr   (Avalon Memory Converted
                                               Destination Address)
          uint                  DMASize       (DMA transfer size in bytes)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
            # CCURPMFC_LIB_NO_ERROR          (successful)
            # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN          (library not open)
            # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
 ****************************************************************************/
```

2.2.139 ccurPMFC_DMA_Fire()

The purpose of this call is to initiate an already configured DMA engine.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_DMA_Fire(void                          *Handle,
                     ccurpmfc_dma_engine_t         DMAEngineNo,
                     ccurpmfc_bool                 UseInterrupts,
                     ccurpmfc_bool                 WaitForCompletion,
                     int                           DmaControl)

   Description: Start DMA Engine

   Input:   void            *Handle                (Handle pointer)
            ccurpmfc_dma_engine_t DMAEngineNo      (select DMA engine)
               # CCURPMFC_DMA0
               # CCURPMFC_DMA1
            ccurpmfc_bool       UseInterrupts      (Enable Interrupt flag)
               # CCURPMFC_TRUE
               # CCURPMFC_FALSE
            ccurpmfc_bool       WaitForCompletion  (Wait for Completion Flag)
               # CCURPMFC_TRUE
               # CCURPMFC_FALSE
            int                 DmaControl         (DMA control flags)
               # CCURPMFC_DMA_CONTROL_RCON         (read constant)
               # CCURPMFC_DMA_CONTROL_WCON         (write constant)
               # CCURPMFC_DMA_CONTROL_INCREMENT    (increment)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (no error)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (library not open)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_IOCTL_FAILED        (ioctl failed)
               # CCURPMFC_LIB_DMA_FAILED          (DMA failed)
 ****************************************************************************/
```

2.2.140 ccurPMFC_Enable_Pci_Interrupts()

The purpose of this call is to enable PCI interrupts. This call shouldn't be used during normal reads as calls could time out. The driver handles enabling and disabling interrupts during its normal course of operation.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Enable_Pci_Interrupts (void                          *Handle,
                                   _ccurpmfc_all_interrupts_mask interrupt_mask)
```

```
   Description: Enable interrupts being generated by the board.

   Input:   void                          *Handle         (Handle pointer)
            _ccurpmfc_all_interrupts_mask interrupt_mask  (interrupt mask)
               # CCURPMFC_DMA0_INTMASK
               # CCURPMFC_DMA1_INTMASK
               # CCURPMFC_MSGDMA_INTMASK
               # CCURPMFC_ADC_FIFO_INTMASK
               # CCURPMFC_DAC_FIFO_INTMASK
               # CCURPMFC_DIO_GROUP0_INTMASK
               # CCURPMFC_DIO_GROUP1_INTMASK
               # CCURPMFC_DIO_GROUP2_INTMASK
               # CCURPMFC_ALL_DMA_INTMASK
               # CCURPMFC_ALL_ANALOG_INTMASK
               # CCURPMFC_ALL_DIO_INTMASK
               # CCURPMFC_DMA_ANALOG_INTMASK
               # CCURPMFC_ALL_INTMASK
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR      (successful)
               # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN      (device not open)
               # CCURPMFC_LIB_IOCTL_FAILED  (driver ioctl call failed)
 ****************************************************************************/
```

2.2.141 ccurPMFC_Fast_Memcpy()

The purpose of this call is to provide a fast mechanism to copy between hardware and memory using programmed I/O. The library performs appropriate locking while the copying is taking place. If the board provides support for double word transfers, this call will utilize it.

```
/****************************************************************************
   ccurPMFC_Fast_Memcpy(void           *Handle,
                        volatile void  *Destination,
                        volatile void  *Source,
                        int            SizeInBytes)

   Description: Perform fast copy to/from buffer using Programmed I/O (WITH LOCKING)

   Input:   void         *Handle            (Handle pointer)
            volatile void *Source           (pointer to source buffer)
            int            SizeInBytes      (transfer size in bytes)
   Ouppput: volatile void *Destination      (pointer to destination buffer)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR      (successful)
               # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN      (device not open)
 ****************************************************************************/
```

2.2.142 ccurPMFC_Fast_Memcpy_Unlocked()

The purpose of this call is to provide a fast mechanism to copy between hardware and memory using programmed I/O. The library does not perform any locking. User needs to provide external locking instead. If the board provides support for double word transfers, this call will utilize it. The *double_word_support* field in the driver information structure *ccurpmfc_driver_info_t* indicates whether the double word support is available in the hardware.

```
/****************************************************************************
   void
   ccurPMFC_Fast_Memcpy_Unlocked(volatile void *Destination,
```

```
                                volatile void *Source,
                                int           SizeInBytes
                                int           DoubleWordSupport)

   Description: Perform fast copy to/from buffer using Programmed I/O
                (WITHOUT LOCKING)

   Input:   volatile void *Source               (pointer to source buffer)
            int            SizeInBytes          (transfer size in bytes)
            int            DoubleWordSupport    (double word support flag)
               # CCURPMFC_FALSE                 (h/w double word transfers not supported)
               # CCURPMFC_TRUE                  (h/w double word transfers supported)
   Oupput:  volatile void *Destination          (pointer to destination buffer)
   Return:  none
*****************************************************************************/
```

2.2.143 ccurPMFC_Fast_Memcpy_Unlocked_FIFO()

The purpose of this call is to provide a simple mechanism to copy between hardware FIFO and memory using programmed I/O. The library does not perform any locking. User needs to provide external locking instead. If the board provides support for double word transfers, this call will utilize it. The *double_word_support* field in the driver information structure *ccurpmfc_driver_info_t* indicates whether the double word support is available in the hardware.

```
/*****************************************************************************
   void
   ccurPMFC_Fast_Memcpy_Unlocked_FIFO(volatile void *Destination,
                                      volatile void *Source,
                                      int           SizeInWords,
                                      int           PioControl,
                                      int           DoubleWordSupport)

   Description: Perform fast copy to/from FIFO buffer using Programmed I/O
                (WITHOUT LOCKING)

   Input:   volatile void *Source               (pointer to source buffer)
            int            SizeInWords          (transfer size in words)
            int            PioControl           (PIO Control)
               # CCURPMFC_PIO_CONTROL_RCON      (read constant)
               # CCURPMFC_PIO_CONTROL_WCON      (write constant)
               # CCURPMFC_PIO_CONTROL_INCREMENT (read/write increment)
            int            DoubleWordSupport    (double word support flag)
               # CCURPMFC_FALSE                 (h/w double word transfers not
                                                 supported)
               # CCURPMFC_TRUE                  (h/w double word transfers
                                                 supported)
   Oupput:  volatile void *Destination          (pointer to destination buffer)
   Return:  none
 *****************************************************************************/
```

2.2.144 ccurPMFC_Fraction_To_Hex()

This converts a fractional decimal to a hexadecimal value.

```
/*****************************************************************************
   int
   ccurPMFC_Fraction_To_Hex (double Fraction,
                             uint   *value)

   Description: Convert Fractional Decimal to Hexadecimal
```

```
   Input:       double   Fraction       (fraction to convert)
   Output:      uint     *value;        (converted hexadecimal value)
   Return:      1                       (call failed)
                0                       (good return)
 ****************************************************************************/
```

2.2.145 ccurPMFC_Get_All_Boards_Driver_Info()

This call returns driver information for all the *ccurpmfc* cards that have been found in the system.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_All_Boards_Driver_Info(void                    *Handle,
                            ccurpmfc_all_boards_driver_info *all_boards_info)

   Description: Get device information from driver for all boards.

   Input:   void                           *Handle          (Handle pointer)
   Output:  ccurpmfc_driver_info_t         *all_boards_info (info struct pointer)
        char                                 version[12]
        char                                 built[32]
        char                                 module_name[16]
        int                                  board_index
        int                                  table_index
        char                                 board_desc[32]
        int                                  bus
        int                                  slot
        int                                  func
        int                                  vendor_id
        int                                  sub_vendor_id
        int                                  sub_device_id
        union {
            u_int                            BoardInfo
            ccurpmfc_boardinfo_t             BInfo
        }
        union {
            u_int                            FirmwareDate
            ccurpmfc_firmware_date_t         FmDate
        }
        union {
            u_int                            FirmwareRevision
            ccurpmfc_firmware_revision_t     FmRev
        }
        int                                  msi_support
        int                                  irqlevel
        double                               calibration_reference_voltage

        int                                  driver_dma_size

        // DMA
        ccurpmfc_driver_dma_info_t           dma_info
       - short   num_trans_tbl_entries
       - int     avalon_page_bits
       - int     avalon_page_size
       - int     tx_interface_base
       - int     dma_max_engines
       - int     dma_max_burst_size
       - int     dma_max_transactions
       - int     dma_max_size
          - int     dma_width_in_bytes
          - int     dma_fire_command
```

```
// Interrupt
ccurpmfc_driver_int_t               interrupt
  - union {
  -   uint           status
  -   uint           InterruptsOccurredMask
  - }
  - union {
  -   uint           mask
  -   uint           WakeupInterruptMask
  - }
  - int                timeout_seconds
  - int                DmaControl
  - long long unsigned  count
  - long long unsigned  dma_count[CCURPMFC_DMA_MAX_ENGINES]
  - long long unsigned  MsgDma_count
// DIO COS Interrupt
ccurpmfc_driver_dio_cos_int_t       dio_cos_interrupt
  - union {
  -   uint           status
  -   uint           InterruptsOccurredMask
  - }
  - union {
  -   uint           mask
  -   uint           WakeupInterruptMask
  - }

  - // DIO information
  - union {
  -   ccurpmfc_dio_channels_t DIO_COS_ChannelsStatus
  -   ccurpmfc_dio_channel_t  DIO_COS_ChannelsStatusX
  - }
  - union {
  -   ccurpmfc_dio_channels_t DIO_COS_ChannelsOverflow
  -   ccurpmfc_dio_channel_t  DIO_COS_ChannelsOverflowX
  - }

ccurpmfc_interrupt_dio_cos_counters_t   counters
  - long long unsigned  dio_interrupt_count
  - long long unsigned  dma_count[CCURPMFC_DMA_MAX_ENGINES]
  - long long unsigned  DIO_COS_ChannelsCount[CCURPMFC_DIO_MAX_REGISTERS]
  - long long unsigned
               DIO_COS_ChannelsOverflowCount[CCURPMFC_DIO_MAX_REGISTERS]

int                                 Ccurpmfc_Max_Region

// Memory Region
ccurpmfc_dev_region_t               mem_region[CCURPMFC_MAX_REGION]
  - uint physical_address
  - uint size
  - uint flags
  - uint *virtual_address

// ADC
ccurpmfc_driver_adc_info_t          adc_info
  - double adc_max_voltage_range
  - int    number_of_adcs
  - int    number_of_adc_channels
  - int    number_of_adc_resolutionbits
  - int    all_adc_channels_mask
  - int    max_adc_fifo_threshold
```

```
  - int    max_adc_frequency

// DAC
ccurpmfc_driver_dac_info_t          dac_info
  - double dac_max_voltage_range
  - int    number_of_dacs
  - int    number_of_dac_channels
  - int    number_of_dac_resolutionbits
  - int    all_dac_channels_mask
  - int    max_dac_fifo_threshold
  - int    max_dac_frequency
// DIO
ccurpmfc_driver_dio_info_t          dio_info
  - int    number_of_dio_channels
  - int    number_of_dio_ports
  - int    number_of_dio_channels_per_port
  - int    number_of_dio_registers
  - int    number_of_dio_channels_per_register

// SDRAM
ccurpmfc_driver_sdram_info_t        sdram_info
  - int    sdram_max_size_in_words
  - _ccurpmfc_clock_generator_output_t sdram_output_clock
  - double sdram_output_clock_frequency

// CLOCK
ccurpmfc_driver_clock_info_t        clock_info
  - _ccurpmfc_cg_input_clock_select_register_t  default_input_clock
  - double default_input_clock_frequency
  - double default_clock_tolerance_ppt

ccurpmfc_sprom_header_t             sprom_header
  - u_int32_t   board_serial_number
  - u_short     sprom_revision

// Chip Temperature
char                                fpga_chip_temperature

char                                double_word_support

union {
    u_int                           FirmwareTime
    ccurpmfc_firmware_time_t        FmTime
}
union {
    u_int                           FirmwareFlavorCode
    ccurpmfc_firmware_option_code_t FmOptionCode
}
u_int                               NumberAdvancedIPCores

u_short                             RunLevelSectorNumber
char                                FirmwareReloadFailed
char                                MultiFirmwareSupport

_ccurpmfc_ipcore_t                  IpCore[CCURPMFC_MAX_IO_CORES]
  - u_int32_t   IpCoreCode
  - union {
  -    u_int32_t                        IpCoreRevision
  -    ccurpmfc_ipcore_revision_t       IpCRev
  - }
  - u_int32_t   IpCoreOffset
```

```
              - union {
              -    u_int32_t                       IpCoreInformation
              -    ccurpmfc_ipcore_information_t   IpCInf
              - }

             union {
                 u_int                           Dummy_time_t[2]
                 time_t                          DriverLoadCurrentTime
             }

             u_int32_t                           FirmwareBoardSerialNumber

             u_int32_t                           MaxMsgDmaDescriptors
             u_int32_t                           MaxMsgDmaSize
             u_int32_t                           FpgawbRevision
             _ccurpmfc_ipcore_t                  IpCore_Plus[CCURPMFC_MAX_IO_CORES_PLUS]
             int                                 CloningSupport
             u_short                             MaximumLinkWidth
             u_short                             NegotiatedLinkWidth
   Return:  _ccurpmfc_lib_error_number_t
             # CCURPMFC_LIB_NO_ERROR      (successful)
             # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
             # CCURPMFC_LIB_NOT_OPEN      (device not open)
             # CCURPMFC_LIB_INVALID_ARG   (invalid argument)
             # CCURPMFC_LIB_IOCTL_FAILED  (driver ioctl call failed)
      ***************************************************************************/
```

2.2.146 ccurPMFC_Get_Board_CSR()

This call returns information from the board status register.

```
      /***************************************************************************
        _ccurpmfc_lib_error_number_t
        ccurPMFC_Get_Board_CSR (void                    *Handle,
                                ccurpmfc_board_csr_t    *bcsr)

         Description: Get Board Control and Status information

         Input:   void                                 *Handle    (Handle pointer)
         Output:  ccurpmfc_board_csr_t                 *bcsr      (pointer to board csr)
                     _ccurpmfc_bcsr_identify_board_t   identify_board
                        # CCURPMFC_BCSR_IDENTIFY_BOARD_DISABLE
                        # CCURPMFC_BCSR_IDENTIFY_BOARD_ENABLE
         Return:  _ccurpmfc_lib_error_number_t
                     # CCURPMFC_LIB_NO_ERROR                (successful)
                     # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
                     # CCURPMFC_LIB_NOT_OPEN                (device not open)
                     # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
                     # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
       ***************************************************************************/
```

2.2.147 ccurPMFC_Get_Board_Info()

This call returns the board id, the board type and the firmware revision level for the selected board. This board id is *0x9277* and board type is *0x1* or *0x9278* with a board type of *0x2*.

```
      /***************************************************************************
         _ccurpmfc_lib_error_number_t
         ccurPMFC_Get_Board_Info (void                    *Handle,
                                  ccurpmfc_board_info_t   *binfo)
         Description: Get Board Information
```

```
   Input:   void                    *Handle        (Handle pointer)
   Output:  ccurpmfc_board_info_t   *binfo         (pointer to board info)
               int                                 vendor_id
               int                                 sub_vendor_id
               int                                 sub_device_id
               ccurpmfc_boardinfo_t                BInfo
                  u_char  Function
                  u_char  Type
                  u_short Id
               ccurpmfc_firmware_date_t            FmDate
                  u_short Year
                  u_char  Day
                  u_char  Month
               ccurpmfc_firmware_revision_t        FmRev
                  u_short Minor
                  u_short Major
               ccurpmfc_sprom_header_t             sprom_header
                  u_int32_t   board_serial_number
                  u_short     sprom_revision
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
 ****************************************************************************/
```

2.2.148 ccurPMFC_Get_Calibration_CSR()

This call returns the current calibration control and status register.

```
/****************************************************************************
   ccurPMFC_Get_Calibration_CSR()

   Description: Get Calibration Control and Status Register

   Input:   void                          *Handle     (Handle pointer)
   Output:  ccurpmfc_calibration_csr_t    *CalCSR     (pointer to calibration CSR)
              _ccurpmfc_calbus_control_t BusControl  (bus control)
                 # CCURPMFC_CB_GROUND
                 # CCURPMFC_CB_POSITIVE_REFERENCE
                 # CCURPMFC_CB_NEGATIVE_REFERENCE
                 # CCURPMFC_CB_BUS_OPEN
                 # CCURPMFC_CB_2_5V_REFERENCE
                 # CCURPMFC_CB_5V_REFERENCE
                 # CCURPMFC_CB_DAC_CHANNEL_0
                 # CCURPMFC_CB_DAC_CHANNEL_1
                 # CCURPMFC_CB_DAC_CHANNEL_2
                 # CCURPMFC_CB_DAC_CHANNEL_3
                 # CCURPMFC_CB_DAC_CHANNEL_4
                 # CCURPMFC_CB_DAC_CHANNEL_5
                 # CCURPMFC_CB_DAC_CHANNEL_6
                 # CCURPMFC_CB_DAC_CHANNEL_7
                 # CCURPMFC_CB_DAC_CHANNEL_8
                 # CCURPMFC_CB_DAC_CHANNEL_9
                 # CCURPMFC_CB_DAC_CHANNEL_10
                 # CCURPMFC_CB_DAC_CHANNEL_11
                 # CCURPMFC_CB_DAC_CHANNEL_12
                 # CCURPMFC_CB_DAC_CHANNEL_13
                 # CCURPMFC_CB_DAC_CHANNEL_14
```

```
                # CCURPMFC_CB_DAC_CHANNEL_15
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR        (successful)
               # CCURPMFC_LIB_BAD_HANDLE      (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN        (device not open)
               # CCURPMFC_LIB_INVALID_ARG     (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION (local region not present)
 ***************************************************************************/
```

2.2.149 ccurPMFC_Get_Driver_Error()

This call returns the last error generated by the driver.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Driver_Error (void                    *Handle,
                              ccurpmfc_user_error_t    *ret_err)

   Description: Get the last error generated by the driver.

   Input:   void                  *Handle             (Handle pointer)
   Output:  ccurpmfc_user_error_t *ret_err            (error struct pointer)
               uint error;                            (error number)
               char name[CCURPMFC_ERROR_NAME_SIZE]    (error name used in driver)
               char desc[CCURPMFC_ERROR_DESC_SIZE]    (error description)
               uint error_line_number                 (error line number)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_IOCTL_FAILED            (driver ioctl call failed)
 ***************************************************************************/

#define CCURPMFC_ERROR_NAME_SIZE    64
#define CCURPMFC_ERROR_DESC_SIZE    128

typedef struct _ccurpmfc_user_error_t
{
    uint error;                                /* error number */
    char name[CCURPMFC_ERROR_NAME_SIZE];    /* error name used in driver */
    char desc[CCURPMFC_ERROR_DESC_SIZE];    /* error description */
} ccurpmfc_user_error_t;

enum
{
    CCURPMFC_SUCCESS = 0,
    CCURPMFC_INVALID_PARAMETER,
    CCURPMFC_DMA_TIMEOUT,
    CCURPMFC_OPERATION_CANCELLED,
    CCURPMFC_RESOURCE_ALLOCATION_ERROR,
    CCURPMFC_INVALID_REQUEST,
    CCURPMFC_FAULT_ERROR,
    CCURPMFC_BUSY,
    CCURPMFC_ADDRESS_IN_USE,
    CCURPMFC_USER_INTERRUPT_TIMEOUT,
    CCURPMFC_DMA_INCOMPLETE,
    CCURPMFC_DATA_UNDERFLOW,
    CCURPMFC_DATA_OVERFLOW,
    CCURPMFC_IO_FAILURE,
    CCURPMFC_OPERATION_NOT_SUPPORTED,
```

```
    CCURPMFC_ADC_FIFO_THRESHOLD_TIMEOUT,
    CCURPMFC_DAC_FIFO_THRESHOLD_TIMEOUT,
    CCURPMFC_INTERRUPT_HANDLER_NOT_ENABLED,
    CCURPMFC_FIRMWARE_RELOAD_FAILED,
    CCURPMFC_DEVICE_AUTHORIZATION_FAILED,
};
```

2.2.150 ccurPMFC_Get_Driver_Info()

This call returns internal information that is maintained by the driver.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Driver_Info (void                      *Handle,
                             ccurpmfc_driver_info_t    *info)

   Description: Get device information from driver.

   Input:   void                              *Handle     (Handle pointer)
   Output:  ccurpmfc_driver_info_t            *info       (info struct pointer)
        char                                  version[12]
        char                                  built[32]
        char                                  module_name[16]
        int                                   board_index
        int                                   table_index
        char                                  board_desc[32]
        int                                   bus
        int                                   slot
        int                                   func
        int                                   vendor_id
        int                                   sub_vendor_id
        int                                   sub_device_id
        union {
            u_int                             BoardInfo
            ccurpmfc_boardinfo_t              BInfo
        }
        union {
            u_int                             FirmwareDate
            ccurpmfc_firmware_date_t          FmDate
        }
        union {
            u_int                             FirmwareRevision
            ccurpmfc_firmware_revision_t      FmRev
        }
        int                                   msi_support
        int                                   irqlevel
        double                                calibration_reference_voltage

        int                                   driver_dma_size

        // DMA
        ccurpmfc_driver_dma_info_t            dma_info
       - short   num_trans_tbl_entries
       - int     avalon_page_bits
       - int     avalon_page_size
       - int     tx_interface_base
       - int     dma_max_engines
       - int     dma_max_burst_size
       - int     dma_max_transactions
       - int     dma_max_size
          - int     dma_width_in_bytes
```

```
  - int     dma_fire_command

// Interrupt
ccurpmfc_driver_int_t               interrupt
  - union {
  -   uint           status
  -   uint           InterruptsOccurredMask
  - }
  - union {
  -   uint           mask
  -   uint           WakeupInterruptMask
  - }
  - int                timeout_seconds
  - int                DmaControl
  - long long unsigned  count
  - long long unsigned  dma_count[CCURPMFC_DMA_MAX_ENGINES]
  - long long unsigned  MsgDma_count
// DIO COS Interrupt
ccurpmfc_driver_dio_cos_int_t       dio_cos_interrupt
  - union {
  -   uint           status
  -   uint           InterruptsOccurredMask
  - }
  - union {
  -   uint           mask
  -   uint           WakeupInterruptMask
  - }

  - // DIO information
  - union {
  -   ccurpmfc_dio_channels_t DIO_COS_ChannelsStatus
  -   ccurpmfc_dio_channel_t  DIO_COS_ChannelsStatusX
  - }
  - union {
  -   ccurpmfc_dio_channels_t DIO_COS_ChannelsOverflow
  -   ccurpmfc_dio_channel_t  DIO_COS_ChannelsOverflowX
  - }

ccurpmfc_interrupt_dio_cos_counters_t   counters
  - long long unsigned  dio_interrupt_count
  - long long unsigned  dma_count[CCURPMFC_DMA_MAX_ENGINES]
  - long long unsigned  DIO_COS_ChannelsCount[CCURPMFC_DIO_MAX_REGISTERS]
  - long long unsigned
               DIO_COS_ChannelsOverflowCount[CCURPMFC_DIO_MAX_REGISTERS]

int                                 Ccurpmfc_Max_Region

// Memory Region
ccurpmfc_dev_region_t               mem_region[CCURPMFC_MAX_REGION]
  - uint physical_address
  - uint size
  - uint flags
  - uint *virtual_address

// ADC
ccurpmfc_driver_adc_info_t          adc_info
  - double adc_max_voltage_range
  - int    number_of_adcs
  - int    number_of_adc_channels
  - int    number_of_adc_resolutionbits
  - int    all_adc_channels_mask
```

```
  - int    max_adc_fifo_threshold
  - int    max_adc_frequency

// DAC
ccurpmfc_driver_dac_info_t          dac_info
  - double dac_max_voltage_range
  - int    number_of_dacs
  - int    number_of_dac_channels
  - int    number_of_dac_resolutionbits
  - int    all_dac_channels_mask
  - int    max_dac_fifo_threshold
  - int    max_dac_frequency
// DIO
ccurpmfc_driver_dio_info_t          dio_info
  - int    number_of_dio_channels
  - int    number_of_dio_ports
  - int    number_of_dio_channels_per_port
  - int    number_of_dio_registers
  - int    number_of_dio_channels_per_register

// SDRAM
ccurpmfc_driver_sdram_info_t        sdram_info
  - int    sdram_max_size_in_words
  - _ccurpmfc_clock_generator_output_t sdram_output_clock
  - double sdram_output_clock_frequency

// CLOCK
ccurpmfc_driver_clock_info_t        clock_info
  - _ccurpmfc_cg_input_clock_select_register_t  default_input_clock
  - double default_input_clock_frequency
  - double default_clock_tolerance_ppt

// SPROM
ccurpmfc_sprom_header_t             sprom_header
  - u_int32_t   board_serial_number
  - u_short     sprom_revision

// Chip Temperature
char                                fpga_chip_temperature

char                                double_word_support

union {
    u_int                           FirmwareTime
    ccurpmfc_firmware_time_t        FmTime
}
union {
    u_int                           FirmwareFlavorCode
    ccurpmfc_firmware_option_code_t FmOptionCode
}
u_int                               NumberAdvancedIPCores

u_short                             RunLevelSectorNumber
char                                FirmwareReloadFailed
char                                MultiFirmwareSupport

_ccurpmfc_ipcore_t                  IpCore[CCURPMFC_MAX_IO_CORES]
  - u_int32_t   IpCoreCode
  - union {
  -    u_int32_t                      IpCoreRevision
  -    ccurpmfc_ipcore_revision_t     IpCRev
```

```
        - }
        - u_int32_t   IpCoreOffset
        - union {
        -    u_int32_t                      IpCoreInformation
        -    ccurpmfc_ipcore_information_t  IpCInf
        - }

      union {
          u_int                          Dummy_time_t[2]
          time_t                         DriverLoadCurrentTime
      }

      u_int32_t                          FirmwareBoardSerialNumber

      u_int32_t                          MaxMsgDmaDescriptors
      u_int32_t                          MaxMsgDmaSize
      u_int32_t                          FpgawbRevision
      _ccurpmfc_ipcore_t                 IpCore_Plus[CCURPMFC_MAX_IO_CORES_PLUS]
      int                                CloningSupport
      u_short                            MaximumLinkWidth
      u_short                            NegotiatedLinkWidth
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR      (successful)
               # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN      (device not open)
               # CCURPMFC_LIB_INVALID_ARG   (invalid argument)
               # CCURPMFC_LIB_IOCTL_FAILED  (driver ioctl call failed)
 ***************************************************************************/
```

2.2.151 ccurPMFC_Get_Interrupt_Status()

This call returns the current status of the various interrupts.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Interrupt_Status (void                 *Handle,
                                  ccurpmfc_interrupt_t *intr)

   Description: Get Interrupt Status information

   Input:   void                    *Handle       (handle pointer)
   Output:  ccurpmfc_interrupt_t    *intr         (pointer to interrupt status)
               _ccurpmfc_intsta_dio_cos_t dio_cos_group2_int
                  # CCURPMFC_INT_DIO_COS_NONE
                  # CCURPMFC_INT_DIO_COS_OCCURRED
               _ccurpmfc_intsta_dio_cos_t dio_cos_group1_int
                  # CCURPMFC_INT_DIO_COS_NONE
                  # CCURPMFC_INT_DIO_COS_OCCURRED
               _ccurpmfc_intsta_dio_cos_t dio_cos_group0_int
                  # CCURPMFC_INT_DIO_COS_NONE
                  # CCURPMFC_INT_DIO_COS_OCCURRED
               _ccurpmfc_intsta_dac_t
                  # CCURPMFC_INT_DAC_FIFO_THRESHOLD_NONE
                  # CCURPMFC_INT_DAC_FIFO_THRESHOLD_OCCURRED
               _ccurpmfc_intsta_adc_t
                  # CCURPMFC_INT_ADC_FIFO_THRESHOLD_NONE
                  # CCURPMFC_INT_ADC_FIFO_THRESHOLD_OCCURRED
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR           (successful)
               # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN           (device not open)
```

```
            # CCURPMFC_LIB_NO_LOCAL_REGION     (local region error)
            # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
 ****************************************************************************/
```

2.2.152 ccurPMFC_Get_Interrupt_Timeout_Seconds()

This call returns the read time out maintained by the driver. It is the time that the read call will wait before it times out. The call could time out because a DMA fails to complete. The device should have been opened in the block mode (*O_NONBLOCK* not set) for reads to wait for the operation to complete.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Interrupt_Timeout_Seconds (void    *Handle,
                                           int     *int_timeout_secs)

   Description: Get Interrupt Timeout Seconds

   Input:   void            *Handle              (Handle pointer)
   Output:  int             *int_timeout_secs    (pointer to int tout secs)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR           (successful)
               # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN           (device not open)
               # CCURPMFC_LIB_INVALID_ARG        (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION    (local region not present)
               # CCURPMFC_LIB_IOCTL_FAILED       (ioctl error)
 ****************************************************************************/
```

2.2.153 ccurPMFC_Get_Lib_Error_Description()

This call returns the library error name and description for the supplied error number.

```
/****************************************************************************

   ccurPMFC_Get_Lib_Error_Description()

   Description: Get Error Description of supplied error number.

   Input:   int                                ErrorNumber     (Library error number)
   Output:  ccurpmfc_lib_error_description_t   *lib_error_desc (error description struct pointer)
               -- int  found
               -- char name[CCURPMFC_LIB_ERROR_NAME_SIZE] (last library error name)
               -- char desc[CCURPMFC_LIB_ERROR_DESC_SIZE] (last library error description)
   Return:  none

 ****************************************************************************/
```

2.2.154 ccurPMFC_Get_Lib_Error()

This call provides detailed information about the last library error that was maintained by the API. The call itself can fail with a return code if an invalid handle is provided, the device is not open or device authorization has failed. If the call succeeds *CCURPMFC_LIB_NO_ERROR*, the last library error information is supplied to the user in the *ccurpmfc_lib_error_t* structure.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Lib_Error (void                    *Handle,
                           ccurpmfc_lib_error_t    *lib_error)

   Description: Get last error generated by the library.

   Input:   void                *Handle                (Handle pointer)
   Output:  ccurpmfc_lib_error_t *lib_error            (error struct pointer)
               -- uint error                           (last library error number)
```

```
              -- char name[CCURPMFC_LIB_ERROR_NAME_SIZE] (last library error name)
              -- char desc[CCURPMFC_LIB_ERROR_DESC_SIZE] (last library error description)
              -- int  line_number                        (last library error line number in lib)
              -- char function[CCURPMFC_LIB_ERROR_FUNC_SIZE]
                                                        (library function in error)
              -- ccurpmfc_lib_error_backtrace_t BT[CCURPMFC_BACK_TRACE_DEPTH]
                                                        (backtrace of errors)
                 -- int  line_number                    (line number in library)
                 -- char function[CCURPMFC_LIB_ERROR_FUNC_SIZE]
                                                        (library function)
   Return:  _ccurpmfc_lib_error_number_t
              # CCURPMFC_LIB_NO_ERROR                   (successful)
              # CCURPMFC_LIB_BAD_HANDLE                 (no/bad handler supplied)
              # CCURPMFC_LIB_NOT_OPEN                   (device not open)
              # CCURPMFC_LIB_AUTHORIZATION_FAILURE      (device authorization failure)
 ************************************************************************/

typedef struct
{
    int  line_number;                           /* line number in library */
    char function[CCURPMFC_LIB_ERROR_FUNC_SIZE]; /* library function */
} ccurpmfc_lib_error_backtrace_t;

typedef struct
{
    uint error;                                 /* last library error number */
    char name[CCURPMFC_LIB_ERROR_NAME_SIZE];    /* last library error name */
    char desc[CCURPMFC_LIB_ERROR_DESC_SIZE];    /* last libarary error description */
    int  line_number;                           /* last library  error line number in
                                                   lib */
    char function[CCURPMFC_LIB_ERROR_FUNC_SIZE]; /* library function in error */
    ccurpmfc_lib_error_backtrace_t BT[CCURPMFC_BACK_TRACE_DEPTH];
                                                /* backtrace of errors */
} ccurpmfc_lib_error_t;
```

Possible library errors:

```
CCURPMFC_LIB_NO_ERROR                     =  0,  /* Successful */
CCURPMFC_LIB_INVALID_ARG                  = -1,  /* Invalid argument */
CCURPMFC_LIB_ALREADY_OPEN                 = -2,  /* Already open */
CCURPMFC_LIB_OPEN_FAILED                  = -3,  /* Open failed */
CCURPMFC_LIB_BAD_HANDLE                   = -4,  /* Bad handle */
CCURPMFC_LIB_NOT_OPEN                     = -5,  /* Device not opened */
CCURPMFC_LIB_MMAP_SELECT_FAILED           = -6,  /* Mmap selection failed */
CCURPMFC_LIB_MMAP_FAILED                  = -7,  /* Mmap failed */
CCURPMFC_LIB_MUNMAP_FAILED                = -8,  /* Munmap failed */
CCURPMFC_LIB_NOT_MAPPED                   = -9,  /* Not mapped */
CCURPMFC_LIB_ALREADY_MAPPED               = -10, /* Device already mapped */
CCURPMFC_LIB_IOCTL_FAILED                 = -11, /* Device IOCTL failed */
CCURPMFC_LIB_IO_ERROR                     = -12, /* I/O error */
CCURPMFC_LIB_INTERNAL_ERROR               = -13, /* Internal library error */
CCURPMFC_LIB_NOT_IMPLEMENTED              = -14, /* Call not implemented */
CCURPMFC_LIB_LOCK_FAILED                  = -15, /* Failed to get lib lock */
CCURPMFC_LIB_NO_LOCAL_REGION              = -16, /* Local region not present */
CCURPMFC_LIB_NO_CONFIG_REGION             = -17, /* Config region not present */
CCURPMFC_LIB_NO_SOLUTION_FOUND            = -18, /* No solution found */
CCURPMFC_LIB_NO_RESOURCE                  = -19, /* Resource not available */
CCURPMFC_LIB_CANNOT_OPEN_FILE             = -20, /* Cannot open file */
CCURPMFC_LIB_DMA_BUSY                     = -21, /* DMA busy */
CCURPMFC_LIB_AVALON_TRANSLATION_TABLE     = -22, /* Avalon translation table error */
CCURPMFC_LIB_ADDRESS_RANGE_ERROR          = -23, /* Physical DMA address exceeds memory
                                                    size */
CCURPMFC_LIB_NO_SPACE_IN_TABLE            = -24, /* No space available to allocate any
```

```
                                                    more physical memory */
CCURPMFC_LIB_CANNOT_ALLOCATE_PHYS_MEM      = -25, /* Cannot allocate physical memory */
CCURPMFC_LIB_DMA_FAILED                    = -26, /* DMA failed */
CCURPMFC_LIB_THREAD_CREATE_FAILED          = -27, /* Thread Creation failed */
CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE           = -28, /* Clock Generator is not active */
CCURPMFC_LIB_CANNOT_COMPUTE_OUTPUT_FREQ    = -29, /* Cannot compute output frequency */
CCURPMFC_LIB_N_DIVIDERS_EXCEEDED           = -30, /* Number of N-Dividers exceeded */
CCURPMFC_LIB_CLOCK_GENERATION_FAILED       = -31, /* Clock generation failed */
CCURPMFC_LIB_CALIBRATION_RANGE_ERROR       = -32, /* Calibration voltage out of range */
CCURPMFC_LIB_BAD_DATA_IN_CAL_FILE          = -33, /* Bad data in calibration file */
CCURPMFC_LIB_VOLTAGE_NOT_IN_RANGE          = -34, /* Voltage not in range */
CCURPMFC_LIB_ADC_IS_NOT_ACTIVE             = -35, /* ADC is not active */
CCURPMFC_LIB_DAC_IS_NOT_ACTIVE             = -36, /* DAC is not active */
CCURPMFC_LIB_ADC_INCORRECTLY_CONFIGURED    = -37, /* ADC incorrectly configured for DAC
                                                    readback */
CCURPMFC_LIB_SDRAM_IS_NOT_ACTIVE           = -38, /* SDRAM is not active */
CCURPMFC_LIB_SDRAM_INITIALIZATION_FAILED   = -39, /* SDRAM initialization failed */
CCURPMFC_LIB_DAC_FIFO_UNDERFLOW            = -40, /* DAC FIFO underflow */
CCURPMFC_LIB_DAC_FIFO_OVERFLOW             = -41, /* DAC FIFO overflow */
CCURPMFC_LIB_DAC_IS_BUSY                   = -42, /* DAC is busy */
CCURPMFC_LIB_DIO_IS_NOT_ACTIVE             = -43, /* DIO is not active */
CCURPMFC_LIB_SERIAL_PROM_FAILURE           = -44, /* Serial PROM failure - malfunction or
                                                    not present */
CCURPMFC_LIB_SERIAL_PROM_BUSY              = -45, /* Serial PROM busy */
CCURPMFC_LIB_SERIAL_PROM_WRITE_PROTECTED   = -46, /* Serial PROM is write protected */
CCURPMFC_LIB_AUTHORIZATION_FAILURE         = -47, /* Failure to authorize opening of
                                                    device */
CCURPMFC_LIB_INTHDLR_CREATE_FAILURE        = -48, /* Interrupt handler creation failure */
CCURPMFC_LIB_INTHDLR_ALREADY_RUNNING       = -49, /* Interrupt handler already running */
CCURPMFC_LIB_IPCORE_COS_IS_NOT_ACTIVE      = -50, /* IP Core COS is Not active */
CCURPMFC_LIB_NO_FREE_DESCRIPTORS_AVAILABLE = -51, /* No Free Descriptors Available */
CCURPMFC_LIB_ERROR_IN_DESCRIPTOR_LIST      = -52, /* Error in Descriptor List */
CCURPMFC_LIB_MSGDMA_NOT_SUPPORTED          = -53, /* Modular Scatter-Gather DMA Not
                                                    Supported */
CCURPMFC_LIB_MSGDMA_READS_NOT_ALLOWED_FOR_SELECTERESS
                                           = -54, /* MSG DMA Reads Not Allowed for
                                                    Selected Address */
CCURPMFC_LIB_NOT_OWNER_OF_MSGDMA           = -55, /* Not Owner of Modular Scatter-Gather
                                                    DMA */
CCURPMFC_LIB_MSGDMA_IN_USE                 = -56, /* Modular Scatter-Gather DMA In Use */
CCURPMFC_LIB_MSGDMA_NOT_SETUP              = -57, /* Modular Scatter-Gather DMA not
                                                    setup */
CCURPMFC_LIB_MSGDMA_FAILED                 = -58, /* Modular Scatter-Gather DMA failed */
CCURPMFC_LIB_MSGDMA_ACCESS_NOT_ALLOWED_FOR_SELECTDRESS
                                           = -59, /* MSGDMA access not allowed for
                                                    selected address */
CCURPMFC_LIB_REGION_ADDRESSING_NOT_SUPPORTED
                                           = -60, /* Region addressing not supported by
                                                    driver */
CCURPMFC_LIB_CLONING_NOT_SUPPORTED         = -61, /* Cloning not supported by the card */
```

2.2.155 ccurPMFC_Get_Library_Info()

This call returns useful library information to the user.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Library_Info (void                      *Handle,
                              ccurpmfc_library_info_t  *info)

   Description: Get library information
```

```
   Input:   void                              *Handle     (Handle pointer)
   Output:  ccurpmfc_library_info_t           *info       (info struct pointer)
               int                            fp;

               ccurpmfc_local_ctrl_data_t           *local_ptr;
                  -- structure in ccurpmfc_user.h
               void                                 *munmap_local_ptr;
               int                                  local_mmap_size;

               ccurpmfc_config_local_data_t         *config_ptr;
                  -- structure in ccurpmfc_user.h
               void                                 *munmap_config_ptr;
               int                                  config_mmap_size;

               ccurpmfc_user_phys_mem_t
                             hysMem[CCURPMFC_MAX_AVALON_NUM_TRANS_TBL_ENTRIES];
                  -- structure in ccurpmfc_user.h

               ccurpmfc_driver_library_common_t     *driver_lib_ptr;
                  -- structure in ccurpmfc_user.h
               void                                 *munmap_driver_lib_ptr;
               int                                  driver_lib_mmap_size;

               void                           *IpCore_Engine_ptr
               void                           *IpCore_InjIgn_Pulse_Capture_ptr
               void                           *IpCore_PWM_Input_ptr
               void                           *IpCore_PWM_Output_ptr
               void                           *IpCore_Tooth_Wheel_Generator_ptr
               ccurpmfc_ipcore_cos_t          *IpCore_COS_ptr
               void                           *IpCore_SENT_Receiver_ptr
               void                           *IpCore_SENT_Transmitter_ptr
               void                           *IpCore_Angular_Encoder_ptr
               void                           *IpCore_Angular_Decoder_ptr
               void                           *IpCore_Knock_Sensor_ptr
               void                           *IpCore_Analog_Threshold_ptr
               void                           *IpCore_Inverter_ptr
               void                           *IpCore_Motor_ptr
               void                           *IpCore_Waveform_Generator_ptr
               void                           *IpCore_High_Pressure_Fuel_Pump_ptr
               void                           *IpCore_Dio_Cos_ptr
               void                  *IpCore_Field_Oriented_Control_Algorithm_ptr
               void                           *IpCore_3_Phase_PWM_ptr
               void                           *IpCore_UVW_Encoder_ptr
               void                           *IpCore_27_ptr
               void                           *IpCore_28_ptr
               void                           *IpCore_29_ptr
               void                           *IpCore_User_Module_ptr
               void                           *FpgaWbLib
               _ccurpmfc_ipcore_t        IpCoreSpecific[CCURPMFC_MAX_IO_CORES]
               uint                           UserPid
               void                           *IpCore_0_ptr
               void                           *IpCore_1_ptr
               void                           *IpCore_2_ptr
               void                           *IpCore_3_ptr
               void                           *IpCore_4_ptr
               void                           *IpCore_5_ptr
               void                           *IpCore_6_ptr
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
```

```
            # CCURPMFC_LIB_NOT_OPEN             (device not open)
            # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
 ***************************************************************************/
```

2.2.156 ccurPMFC_Get_Mapped_Config_Ptr()

If the user wishes to bypass the API and communicate directly with the board configuration registers, then they can use this call to acquire a pointer to these registers. Please note that any type of access (read or write) by bypassing the API could compromise the API and results could be unpredictable. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the hardware programming registers before attempting to access these registers. For information on the registers, refer to the *ccurpmfc_user.h* include file that is supplied with the driver.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Mapped_Config_Ptr (void                            *Handle,
                                   ccurpmfc_config_local_data_t    **config_ptr)

   Description: Get mapped configuration pointer.

   Input:   void                            *Handle         (Handle pointer)
   Output:  ccurpmfc_config_local_data_t    **config_ptr    (config struct ptr)
               -- structure in ccurpmfc_user.h
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_CONFIG_REGION       (config region not present)
***************************************************************************/
```

2.2.157 ccurPMFC_Get_Mapped_Driver_Library_Ptr()

The driver and library share a common structure. This call returns a pointer to the shared driver/library structure.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Mapped_Driver_Library_Ptr (void                *Handle,
                           ccurpmfc_driver_library_common_t    **driver_lib_ptr)

   Description: Get mapped Driver/Library structure pointer.

   Input:   void                               *Handle          (Handle pointer)
   Output:  ccurpmfc_driver_library_common_t  **driver_lib_ptr  (driver_lib
                                                                 struct ptr)
               -- structure in ccurpmfc_user.h
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION        (local region not present)
 ***************************************************************************/
```

2.2.158 ccurPMFC_Get_Mapped_Local_Ptr()

If the user wishes to bypass the API and communicate directly with the board control and data registers, then they can use this call to acquire a pointer to these registers. Please note that any type of access (read or write) by bypassing the API could compromise the API and results could be unpredictable. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the hardware

programming registers before attempting to access these registers. For information on the registers, refer to the *ccurpmfc_user.h* include file that is supplied with the driver.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Mapped_Local_Ptr (void                          *Handle,
                                  ccurpmfc_local_ctrl_data_t   **local_ptr)

   Description: Get mapped local pointer.

   Input:   void                         *Handle     (Handle pointer)
   Output:  ccurpmfc_local_ctrl_data_t  **local_ptr (local struct ptr)
               -- structure in ccurpmfc_user.h
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
 ************************************************************************/
```

2.2.159 ccurPMFC_Get_Open_File_Descriptor()

When the library *ccurPMFC_Open()* call is successfully invoked, the board is opened using the system call *open(2)*. The file descriptor associated with this board is returned to the user with this call. This call allows advanced users to bypass the library and communicate directly with the driver with calls like *read(2), ioctl(2),* etc. Normally, this is not recommended as internal checking and locking is bypassed and the library calls can no longer maintain integrity of the functions. This is only provided for advanced users who want more control and are aware of the implications.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Open_File_Descriptor (void *Handle,
                                      int  *fd)

   Description: Get Open File Descriptor

   Input:   void                  *Handle  (Handle pointer)
   Output:  int                   *fd      (open file descriptor)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR      (successful)
               # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN      (device not open)
               # CCURPMFC_LIB_INVALID_ARG   (invalid argument)
 ************************************************************************/
```

2.2.160 ccurPMFC_Get_Physical_Memory()

This call returns to the user the physical memory pointer and size that was previously allocated by the *ccurPMFC_Mmap_Physical_Memory()* call. The physical memory is allocated by the user when they wish to perform their own DMA and bypass the API. If user specified a mmaped user memory pointer, search for it, otherwise, simply return the contents of the physical memory list specified by a valid entry_num_in_tran_table. Once again, this call is only useful for advanced users.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Physical_Memory (void                       *Handle,
                                 ccurpmfc_user_phys_mem_t  *phys_mem)

   Description: Get previously mmapped() physical memory address and size
```

```
   Input:   void                      *Handle                (Handle pointer)
            ccurpmfc_user_phys_mem_t *phys_mem              (mem struct pointer)
               void                  *mmaped_user_mem_ptr   (mmaped user virtual memory)
               uint                  entry_num_in_tran_table (entry number in translation table)
   Output:  ccurpmfc_user_phys_mem_t *phys_mem              (mem struct pointer)
               uint                     user_pid
               void                     *phys_mem_ptr
               void                     *driver_virt_mem_ptr
               void                     *mmaped_user_mem_ptr
               uint                     phys_mem_size
               uint                     phys_mem_size_freed
               uint                     entry_num_in_tran_table
               uint                     num_of_entries_used
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR               (successful)
               # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN               (device not open)
               # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
               # CCURPMFC_LIB_IOCTL_FAILED           (driver ioctl call failed)
 ***************************************************************************/
```

2.2.161 ccurPMFC_Get_RunCount_UserProcess()

This call returns to the user a count of the number of times the User Process has entered. *(This is an experimental API for debugging and testing).*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_RunCount_UserProcess(void                  *UFuncHandle,
                                     unsigned int long long *RunCount)

   Description: Get run count in user process

   Input:   void                       *UFuncHandle (UF Handle pointer))
   Output:  unsigned int long long     *RunCount    (pointer to run count)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
 ***************************************************************************/
```

2.2.162 ccurPMFC_Get_TestBus_Control()

This call is provided for internal use in testing the hardware.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_TestBus_Control (void                       *Handle,
                                 _ccurpmfc_testbus_control_t *test_control)

   Description: Return the value of the Test Bus control information

   Input:   void                    *Handle       (handle pointer)
   Output:  _ccurpmfc_testbus_control_t
                                    *test_control (pointer to control select)
               # CCURPMFC_TBUS_CONTROL_OPEN
               # CCURPMFC_TBUS_CONTROL_CAL_BUS
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR           (successful)
               # CCURPMFC_LIB_NO_LOCAL_REGION    (local region error)
               # CCURPMFC_LIB_BAD_HANDLE         (no/bad handler supplied)
```

```
            # CCURPMFC_LIB_NOT_OPEN           (device not open)
 ****************************************************************************/
```

2.2.163 ccurPMFC_Get_Value()

This call allows the user to read the board registers. The actual data returned will depend on the command register information that is requested. Refer to the hardware manual for more information on what is being returned. Most commands return a pointer to an unsigned integer.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Get_Value (void               *Handle,
                       CCURPMFC_CONTROL    cmd,
                       void               *value)

   Description: Return the value of the specified board register.

   Input:   void               *Handle         (Handle pointer)
            CCURPMFC_CONTROL    cmd            (register definition)
               -- structure in ccurpmfc_lib.h
   Output:  void               *value;         (pointer to value)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
 ****************************************************************************/
```

2.2.164 ccurPMFC_Hex_To_Fraction()

This call converts a hexadecimal value to a fractional decimal.

```
/****************************************************************************
   double
   ccurPMFC_Hex_To_Fraction (uint value)

   Description: Convert Hexadecimal to Fractional Decimal

   Input:       uint     value                 (hexadecimal to convert)
   Output:      none
   Return:      double   Fraction              (converted fractional value)
 ****************************************************************************/
```

2.2.165 ccurPMFC_Identify_Board()

This call is useful in identifying a physical board via software control. It causes the front LED to either blink or turn off. Users can also specify the number of seconds they wish to blink the LED.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Identify_Board (void                   *Handle,
                            _ccurpmfc_identify_t  Identify)

   Description: Identify the board by setting the front LED

   Input:   void                 *Handle              (Handle pointer)
            _ccurpmfc_identify_t Identify            (Identify board settings)
               # CCURPMFC_IDENTIFY_OFF
               # CCURPMFC_IDENTIFY_ON
               # Number of seconds to blink
```

```
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
 ***************************************************************************/
```

2.2.166 ccurPMFC_Initialize_Board()

This call initializes the driver structures to a default state and then resets the hardware.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Initialize_Board (void *Handle)

   Description: Initialize the board.

   Input:   void            *Handle             (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_IOCTL_FAILED      (driver ioctl call failed)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
 ***************************************************************************/
```

2.2.167 ccurPMFC_IpCore_COS_Activate()

This call is the first call to use after the system is powered up to enable the IpCore COS module. This call can also be used to de-activate the module and get the current state of the module.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_IpCore_COS_Activate (void                                  *Handle,
                                 _ccurpmfc_ipcore_cos_ip_enable_t      activate,
                                 _ccurpmfc_ipcore_cos_status_disabled_t
                                                                   *current_state)

   Description: Activate/DeActivate IP Core COS module

   Input:   void                                  *Handle  (Handle pointer)
            _ccurpmfc_ipcore_cos_ip_enable_t      activate (activate/deactivate)
               # CCURPMFC_IPCORE_COS_IP_ENABLE
               # CCURPMFC_IPCORE_COS_IP_DISABLE
               # CCURPMFC_IPCODE_COS_IP_ENABLE_DO_NOT_CHANGE
   Output:  _ccurpmfc_ipcore_cos_status_disabled_t *current_state
                                                        (active/deactive)
               # CCURPMFC_IPCORE_COS_STATUS_IP_ENABLED
               # CCURPMFC_IPCORE_COS_STATUS_IP_DISABLED
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE             (COS ip core not present)
***************************************************************************/
```

2.2.168 ccurPMFC_IpCore_COS_Configure()

The purpose of this call is to configure the Ip Core COS module. If the core is not active, it will activate it. User can decicde to immediately start capture at the end of the configuration with the StartStop option. The test mode is simply provided for debugging the hardware and software. Users need to specify at least one channel to detect a change of state, otherwise the call will fail.

A point to note is that when this call is issued, the sampling timer is restarted.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_IpCore_COS_Configure (void                              *Handle,
                                  ccurpmfc_ipcore_cos_configure_t *config)

   Description: Configure IP Core Change-of-State

   Input:   void                                *Handle (Handle pointer)
   Output:  ccurpmfc_ipcore_cos_configure_t    *config (pointer to config Struct)
               u_int32_t                    StartStop
                  # CCURPMFC_IPCORE_COS_STOP_CAPTURE
                  # CCURPMFC_IPCORE_COS_START_CAPTURE
               u_int32_t                    OperationalMode
                  # CCURPMFC_IPCORE_COS_ENABLE_NORMAL_MODE
                  # CCURPMFC_IPCORE_COS_ENABLE_TEST_MODE
               u_int32_t                    TestModePattern
                  # CCURPMFC_IPCORE_COS_TEST_PATTERN_ALTERNATING
                  # CCURPMFC_IPCORE_COS_TEST_PATTERN_INCREMENT
               u_iny32_t                    TestModeDivider
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_1
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_2
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_4
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_8
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_16
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_32
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_64
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_128
               u_int32_t                    ChannelMask_31_00
               u_int32_t                    ChannelMask_63_32
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE             (COS Ip core not present)
*****************************************************************************/
```

2.2.169 ccurPMFC_IpCore_COS_Decode_Timestamp()

This is a useful call that is available to the user to decode the raw timestamp of the chage-of-state that is supplied to the call.

```
/*****************************************************************************
   void
   ccurPMFC_IpCore_COS_Decode_Timestamp (uint                timestamp_31_00,
                                         uint                timestamp_63_32,
                     _ccurpmfc_ipcore_cos_decode_timestamp_t *decode_timestamp)

   Description: IP Core Decode Timestamp

   Input:   uint                                    timestamp_31_00
```

```
          uint                                        timestamp_63_32
   Output:  _ccurpmfc_ipcore_cos_decode_timestamp_t *decode_timestamp
                                                    (pointer to timestamp Struct)
               uint    Day
               uint    Hours
               uint    Minutes
               uint    Seconds
               uint    MilliSeconds
               uint    MicroSeconds
   Return:  none
*****************************************************************************/
```

2.2.170 ccurPMFC_IpCore_COS_Get_Info()

This is a useful call to display the current state of the COS Ip Core. Users can use this call prior to making changes to using the COS configuration call.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_IpCore_COS_Get_Info (void                       *Handle,
                                 ccurpmfc_ipcore_cos_info_t *info)

   Description: Get Information of IP Core Change-of-State

   Input:   void                                 *Handle (Handle pointer)
   Output:  ccurpmfc_ipcore_cos_info_t           *info   (pointer to info Struct)
               u_int32_t                    StartStop
                  # CCURPMFC_IPCORE_COS_STOP_CAPTURE
                  # CCURPMFC_IPCORE_COS_START_CAPTURE
               u_int32_t                    OperationalMode
                  # CCURPMFC_IPCORE_COS_ENABLE_NORMAL_MODE
                  # CCURPMFC_IPCORE_COS_ENABLE_TEST_MODE
               u_int32_t                    TestModePattern
                  # CCURPMFC_IPCORE_COS_TEST_PATTERN_ALTERNATING
                  # CCURPMFC_IPCORE_COS_TEST_PATTERN_INCREMENT
               u_iny32_t                    TestModeDivider
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_1
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_2
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_4
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_8
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_16
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_32
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_64
                  # CCURPMFC_IPCORE_COS_TEST_DIVIDER_128
               u_int32_t                    IpEnable
                  # CCURPMFC_IPCORE_COS_IP_DISABLE
                  # CCURPMFC_IPCORE_COS_IP_ENABLE
               u_int32_t                    ChannelMask_31_00
               u_int32_t                    ChannelMask_63_32
               u_int32_t                    FifoNotFull
                  # CCURPMFC_IPCORE_COS_STATUS_FIFO_FULL
                  # CCURPMFC_IPCORE_COS_STATUS_FIFO_NOT_FULL
               u_int32_t                    OverflowDetected
                  # CCURPMFC_IPCORE_COS_STATUS_FIFO_NO_OVERFLOW
                  # CCURPMFC_IPCORE_COS_STATUS_FIFO_OVERFLOW
               u_int32_t                    IpNotEnabled
                  # CCURPMFC_IPCORE_COS_STATUS_IP_ENABLED
                  # CCURPMFC_IPCORE_COS_STATUS_IP_DISABLED
               u_int32_t                    FifoCount
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
```

```
                # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN                 (device not open)
                # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
                # CCURPMFC_LIB_NO_LOCAL_REGION          (local region not present)
                # CCURPMFC_LIB_NO_RESOURCE              (COS Ip core not present)
*************************************************************************/
```

2.2.171 ccurPMFC_IpCore_COS_Read()

This call reads the COS FIFO and returns any raw change-of-state elements that have been detected by the firmware. The maximum size of the FIFO is *CCURPMFC_IPCORE_COS_MAX_READ_ELEMENTS* elements. The call requires as input the number of samples to read. This can be any number between *1* and *CCURPMFC_IPCORE_COS_MAX_READ_ELEMENTS*. Additionally, the user needs to select the mode of transfer, i.e. DMA or PIO. If DMA mode is selected, the user has the option to select the DMA engine *CCURPMFC_DMA0* or *CCURPMFC_DMA1*. An additional option to this call is to decode any raw change-of-state elements detected by setting the *timestamp* option to *CCURPMFC_TRUE*.

The call returns the number of elements that it read along with the raw and decoded information. If an overflow condition occurs (i.e. the FIFO gets full and the firmware is unable to add another change of state detected) all available elements are returned up to the user requested count and the overflow condition is cleared. Overflow can occur if the rate of change is state detection is very high and the application is unable to read the FIFO before turning around to collect more data. In that case, the application needs to speed up the process of capturing the data or reducing the rate of change of state.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_IpCore_COS_Read (void                        *Handle,
                             ccurpmfc_ipcore_cos_data_t *CosDataPtr)

   Description: Get Information of IP Core Change-of-State

   Input:   void                          *Handle     (Handle pointer)
            ccurpmfc_ipcore_cos_data_t    *CosDataPtr (pointer to data struct)
                ushort num_elements                   (number of elements to read)
                ushort decode_timestamp
                    # CCURPMFC_FALSE
                    # CCURPMFC_TRUE
                ushort transfer_mode
                    # CCURPMFC_LIBRARY_PIO_MODE
                    # CCURPMFC_LIBRARY_DMA_MODE
                ushort dma_engine_number
                    # CCURPMFC_DMA0
                    # CCURPMFC_DMA1
   Output:  ccurpmfc_ipcore_cos_data_t       *CosDataPtr (pointer to data struct)
                ushort num_elements_returned (number of elements returned)
                ushort overflow_detected     (indicate if overflow occurred)
                    # CCURPMFC_IPCORE_COS_STATUS_FIFO_NO_OVERFLOW
                    # CCURPMFC_IPCORE_COS_STATUS_FIFO_OVERFLOW
                _ccurpmfc_ipcore_cos_data_t
                       element[CCURPMFC_IPCORE_COS_MAX_READ_ELEMENTS]
                    uint    timestamp_31_00
                    uint    timestamp_63_32
                    uint    channel_mask_31_00
                    uint    channel_mask_63_32
                _ccurpmfc_ipcore_cos_decode_timestamp_t
                       timestamp[CCURPMFC_IPCORE_COS_MAX_READ_ELEMENTS]
                    uint    Days
                    uint    Hours
                    uint    Minutes
                    uint    Second
```

```
                  uint    MicroSeconds
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE             (COS Ip core not present)
               # CCURPMFC_LIB_IPCORE_COS_NOT_ACTIVE   (COS Ip core not active)
*****************************************************************************/
```

2.2.172 ccurPMFC_IpCore_COS_Start_Stop()

This call can be used to control the starting and stopping of the change-of-state capture. When the capture is started, the user can select to restart the timer by setting the *ResetTimer* option to *CCURPMFC_TRUE*.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_IpCore_COS_Start_Stop (void                                  *Handle,
                            _ccurpmfc_ipcore_cos_start_capture_t    StartStop,
                            uint                                    ResetTimer)

   Description: Start/Stop IP Core COS capture

   Input:   void                                  *Handle   (Handle pointer)
            _ccurpmfc_ipcore_cos_start_capture_t StartStop (start/stop COS
                                                            capture)
               # CCURPMFC_IPCORE_COS_STOP_CAPTURE
               # CCURPMFC_IPCORE_COS_START_CAPTURE
            uint                                        ResetTimer (clear timer)
               # CCURPMFC_TRUE
               # CCURPMFC_FALSE
   Output:  None
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE             (COS Ip core not present)
               # CCURPMFC_LIB_IPCORE_COS_NOT_ACTIVE   (COS Ip core not active)
*****************************************************************************/
```

2.2.173 ccurPMFC_IpCore_Get_Info()

This call returns information of all the IP Core modules available. The COS core will return a *CCURPMFC_IPCODE_CHANGE_OF_STATE* code in *IpCoreCode*. Additional information about the core is also returned.

This call also returns to the user a memory mapped pointer address that the user can use to directly access the IP Core and bypass the driver and API. This type of access to the hardware should only be performed by Advanced users who are extremely familiar with both the hardware and internals of the core, otherwise, the system operation could be compromised.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_IpCore_Get_Info (void                      *Handle,
                             ccurpmfc_ipcore_info_t   *ip_info)

   Description: Get IP Core Information
```

```
Input:   void                            *Handle  (Handle pointer)
Output:  ccurpmfc_ipcore_info_t          *ip_info (pointer to Ip Core Struct)
           u_int32_t             NumberAdvancedIPCores;
           ccurpmfc_ipcore_info_t IpInfo[CCURPMFC_MAX_IO_CORES]
               _ccurpmfc_ipcore_t IpCore;
                   u_int32_t   IpCoreCode
                         # CCURPMFC_IPCODE_0 ... CCURPMFC_IPCODE_2
                         # CCURPMFC_IPCODE_LVDT_TX
                         # CCURPMFC_IPCODE_LVDT_RX
                         # CCURPMFC_IPCODE_5 ... CCURPMFC_IPCODE_6
                         # CCURPMFC_IPCODE_ENGINE_IP
                         # CCURPMFC_IPCODE_INJ_IGN_PULSE_CAPTURE
                         # CCURPMFC_IPCODE_PWM_INPUT
                         # CCURPMFC_IPCODE_PWM_OUTPUT
                         # CCURPMFC_IPCODE_TOOTH_WHEEL_GENERATOR
                         # CCURPMFC_IPCODE_CHANGE_OF_STATE
                         # CCURPMFC_IPCODE_SENT_RECEIVER
                         # CCURPMFC_IPCODE_SENT_TRANSMITTER
                         # CCURPMFC_IPCODE_ANGULAR_ENCODER
                         # CCURPMFC_IPCODE_ANGULAR_DECODER
                         # CCURPMFC_IPCODE_KNOCK_SENSOR
                         # CCURPMFC_IPCODE_ANALOG_THRESHOLD
                         # CCURPMFC_IPCODE_INVERTER
                         # CCURPMFC_IPCODE_MOTOR
                         # CCURPMFC_IPCODE_WAVEFORM_GENERATOR
                         # CCURPMFC_IPCODE_HIGH_PRESSURE_FUEL_PUMP
                         # CCURPMFC_IPCODE_DIO_COS
                         # CCURPMFC_IPCODE_FIELD_ORIENTED_CONTROL_ALGORITHM
                         # CCURPMFC_IPCODE_3_PHASE_PWM
                         # CCURPMFC_IPCODE_UVW_ENCODER
                         # CCURPMFC_IPCODE_UVW_DECODER
                         # CCURPMFC_IPCODE_RESOLVER_TX
                         # CCURPMFC_IPCODE_RESOLVER_RX
                         # CCURPMFC_IPCODE_USER_MODULE
                         # CCURPMFC_IPCODE_31 ... CCURPMFC_IPCODE_31
                         # CCURPMFC_IPCODE_SYNCHRO_TX
                         # CCURPMFC_IPCODE_SYNCHRO_RX
                         # CCURPMFC_IPCODE_IRIGB_TX
                         # CCURPMFC_IPCODE_IRIGB_RX
                         # CCURPMFC_IPCODE_AM_TX
                         # CCURPMFC_IPCODE_AM_RX
                         # CCURPMFC_IPCODE_38 ... CCURPMFC_IPCODE_39
                         # CCURPMFC_IPCODE_SPI_MASTER
                         # CCURPMFC_IPCODE_SPI_SLAVE
                         # CCURPMFC_IPCODE_42 ... CCURPMFC_IPCODE_79
                         # CCURPMFC_IPCODE_NGC_SIO
                         # CCURPMFC_IPCODE_81 ... CCURPMFC_IPCODE_147
                   union {
                        u_int32_t                 IpCoreRevision;
                        ccurpmfc_ipcore_revision_t IpCRev
                    }
                    u_int32_t   IpCoreOffset
                    u_int32_t   IpCoreInformation
               void       *IpCoreMappedPtr
               char        IpCoreName[CCURPMFC_IPCORE_NAME_SIZE]
               char        IpCoreDescription[CCURPMFC_IPCORE_DESC_SIZE]
Return:  _ccurpmfc_lib_error_number_t
           # CCURPMFC_LIB_NO_ERROR             (successful)
           # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
           # CCURPMFC_LIB_NOT_OPEN             (device not open)
           # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
```

```
              # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
*****************************************************************************/
```

2.2.174 ccurPMFC_IpCore_Get_Mapped_Ptr()

This call returns to the user a memory mapped pointer address that the user can use to directly access the IP Core and bypass the driver and API. This type of access to the hardware should only be performed by Advanced users who are extremely familiar with both the hardware and internals of the core, otherwise, the system operation could be compromised.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_IpCore_Get_Mapped_Ptr (void        *Handle,
                                   u_int32_t   IpCoreCode,
                                   void        **ipcore_ptr)

   Description: Get mapped requested IP Core pointer.

   Input:   void                        *Handle      (Handle pointer)
            u_int32_t                   IpCoreCode
               # CCURPMFC_IPCODE_0 ... CCURPMFC_IPCODE_2
               # CCURPMFC_IPCODE_LVDT_TX
               # CCURPMFC_IPCODE_LVDT_RX
               # CCURPMFC_IPCODE_5 ... CCURPMFC_IPCODE_6
               # CCURPMFC_IPCODE_ENGINE_IP
               # CCURPMFC_IPCODE_INJ_IGN_PULSE_CAPTURE
               # CCURPMFC_IPCODE_PWM_INPUT
               # CCURPMFC_IPCODE_PWM_OUTPUT
               # CCURPMFC_IPCODE_TOOTH_WHEEL_GENERATOR
               # CCURPMFC_IPCODE_CHANGE_OF_STATE
               # CCURPMFC_IPCODE_SENT_RECEIVER
               # CCURPMFC_IPCODE_SENT_TRANSMITTER
               # CCURPMFC_IPCODE_ANGULAR_ENCODER
               # CCURPMFC_IPCODE_ANGULAR_DECODER
               # CCURPMFC_IPCODE_KNOCK_SENSOR
               # CCURPMFC_IPCODE_ANALOG_THRESHOLD
               # CCURPMFC_IPCODE_INVERTER
               # CCURPMFC_IPCODE_MOTOR
               # CCURPMFC_IPCODE_WAVEFORM_GENERATOR
               # CCURPMFC_IPCODE_HIGH_PRESSURE_FUEL_PUMP
               # CCURPMFC_IPCODE_DIO_COS
               # CCURPMFC_IPCODE_FIELD_ORIENTED_CONTROL_ALGORITHM
               # CCURPMFC_IPCODE_3_PHASE_PWM
               # CCURPMFC_IPCODE_UVW_ENCODER
               # CCURPMFC_IPCODE_UVW_DECODER
               # CCURPMFC_IPCODE_RESOLVER_TX
               # CCURPMFC_IPCODE_RESOLVER_RX
               # CCURPMFC_IPCODE_USER_MODULE
               # CCURPMFC_IPCODE_31 ... CCURPMFC_IPCODE_31
               # CCURPMFC_IPCODE_SYNCHRO_TX
               # CCURPMFC_IPCODE_SYNCHRO_RX
               # CCURPMFC_IPCODE_IRIGB_TX
               # CCURPMFC_IPCODE_IRIGB_RX
               # CCURPMFC_IPCODE_AM_TX
               # CCURPMFC_IPCODE_AM_RX
               # CCURPMFC_IPCODE_38 ... CCURPMFC_IPCODE_39
               # CCURPMFC_IPCODE_SPI_MASTER
               # CCURPMFC_IPCODE_SPI_SLAVE
               # CCURPMFC_IPCODE_42 ... CCURPMFC_IPCODE_79
               # CCURPMFC_IPCODE_NGC_SIO
               # CCURPMFC_IPCODE_81 ... CCURPMFC_IPCODE_147
```

```
   Output:  void                        **ipcore_ptr (ipcore ptr)
               -- structure in ccurpmfc_user.h
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCURPMFC_LIB_NO_RESOURCE          (Ip core not present)
 ************************************************************************/
```

2.2.175 ccurPMFC_MMap_Physical_Memory()

This call is provided for advanced users to create a physical memory of specified size that can be used for DMA or MsgDMA. The allocated DMA memory is rounded to a page size. If a physical memory is not available, this call will fail, at which point the user will need to issue the *ccurPMFC_Munmap_Physical_Memory()* API call to remove any previously allocated physical memory.

When user wishes to allocate a physical memory, they must make sure that the ***phys_mem_ptr*** in the ***ccurpmfc_user_phys_mem_t*** structure is set to 0, otherwise the call will fail.

Instead of creating a physical memory, this same call can be used to map a user specified region if ***region addressing*** support is enabled as part of the Cloning feature. In this case, the user will need to supply a valid physical address of a Cloning Region to the ***phys_mem_ptr*** argument in this call.

Additionally, it is meaningless to perform Cloning on a FIFO region for two reasons. Firstly, each data in a FIFO is synchronous, however, the Cloned region is accessed asynchronously. Secondly, when the FIFO runs empty *(underflow)* or cannot accept more data *(overflow)* the results are unpredictable.

<u>Caution</u>: Since physical addresses are supplied for the MsgDma operation, care must be taken to ensure that the supplied addresses are valid and that while DMA is in progress, the memory regions must not be freed or made inactive, otherwise, the results could be unpredictable and could lead to the possible corruption of the system.

If the user supplies a non-zero ***phys_mem_ptr*** argument, the driver will attempt to request access to the memory region supplied by the user. If access to the region is denied, the call will fail. Reasons for access being denied is because the region has been reserved by some other process and is possibly in use. In this case, if the user still wishes to get access to the region, they can do so *at their own risk* by supplying the *CCURPMFC_DISABLE_REGION_PROTECTION* flag to the *flags* argument. If the call still fails, there is no way for the user to access the memory region as the kernel controls this access. One such reason is that the user is trying to access an invalid region.

Whether a physical memory is acquired by the driver or supplied by the user, the driver by default *caches* the memory region and returns a mapped virtual address to the user. If the user does not wish the region to be *cached*, they can supply the *CCURPMFC_DISABLE_ADDRESS_CACHE* flag to the *flags* argument. This may be useful if the user is running into problems with the region being *cached*, however, a noticeable performance degredation will be observed when accessing the region.

The *CCURPMFC_DEVICE_ADDRESS_ENTRY* is used internally by the driver and is only available as information to the user.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_MMap_Physical_Memory (void                        *Handle,
                                  int                         size,
                                  ccurpmfc_user_phys_mem_t    *phys_mem)
```

```
   Description: Allocate a physical DMA memory for size bytes.

   Input:   void                        *Handle          (Handle pointer)
            int                         size             (size in bytes)
   Output:  ccurpmfc_user_phys_mem_t    *phys_mem        (mem struct pointer)
               uint                     user_pid
               void                     *phys_mem_ptr
               void                     *driver_virt_mem_ptr
               void                     *mmaped_user_mem_ptr
               uint                     phys_mem_size
               uint                     phys_mem_size_freed
               uint                     entry_num_in_tran_table
               ushort                   flags
                  # CCURPMFC_DEVICE_ADDRESS_ENTRY
                  # CCURPMFC_DISABLE_ADDRESS_CACHE
                  # CCURPMFC_DISABLE_REGION_PROTECTION
               ushort                   num_of_entries_used
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (device not open)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_MMAP_SELECT_FAILED  (mmap selection failed)
               # CCURPMFC_LIB_MMAP_FAILED         (mmap failed)
               # CCURPMFC_LIB_NO_SPACE_IN_TABLE   (no space in phys memory table)
               # CCURPMFC_LIB_REGION_ADDRESSING_NOT_SUPPORTED
                                                  (region addressing not
                                                   supported by the card)
               # CCURPMFC_LIB_MSGDMA_ACCESS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                                  (access not allowed for
                                                   selected address)
*****************************************************************************/
```

2.2.176 ccurPMFC_MsgDma_Clone()

US Patent US 11.281.584 B1, Inventor Darius Dubash

This call allows the user to Clone a transfer so that the process is continuously performing MsgDma once it has started until the Cloning operation is stopped by the user. This approach is different from standard MsgDma where a user has to re-initiate a MsgDma transfer every time it completes.

The following are the operation modes for this call:

- CCURPMFC_MSGDMA_CLONE_INITIALIZE
- CCURPMFC_MSGDMA_CLONE_ONE_CYCLE_WAIT
- CCURPMFC_MSGDMA_CLONE_START
- CCURPMFC_MSGDMA_CLONE_STOP

In order to perform a Cloning operation, the user first performs the same functions of MsgDma to seize, configure descriptors and MsgDma setup using the *ccurPMFC_MsgDma_Seize()*, *ccurPMFC_MsgDma_Configure_Descriptor()* and *ccurPMFC_MsgDma_Setup()* calls. Once that is done, the user needs to stop any previous MsgDma operation and initialize the cloning operation using (*CCURPMFC_MSGDMA_CLONE_STOP* | *CCURPMFC_MSGDMA_CLONE_INITIALIZE*) modes.

Now, whenever the user is ready, they can commence cloning operation with the *CCURPMFC_MSGDMA_CLONE_START* mode. At this point, MsgDma transfers start occurring continuously at the hardware level. If a chained MsgDma is configured, the entire chain is completed

before it is repeated. Once Cloning has commenced, it can me stopped with the help of the *CCURPMFC_MSGDMA_CLONE_STOP* mode.

Once the operation has started with the *CCURPMFC_MSGDMA_CLONE_START* mode, it will run continuously under hardware control until stopped. There is no way to determine precisely how long a single descriptor cycle takes to complete. If the *CCURPMFC_MSGDMA_CLONE_ONE_CYCLE_WAIT* mode is set along with the *CCURPMFC_MSGDMA_CLONE_START* mode, the call will be blocked for the first transfer until the full descriptor cycle has completed. This approximate duration is also saved internally in the driver and is available to the user in the *CloneArgs->MsgDmaExtDesOnlyCycleDelay* argument. Anytime the user wishes to block their application for a duration of approximately one cycle delay, they can invoke this call with the *CCURPMFC_MSGDMA_CLONE_ONE_CYCLE_WAIT* as the only mode. If the user wishes to block more or less than the one cycle delay whenever the call is issued, they can specify the number of additional nano-seconds to block in the *CloneArgs->AdditionalOneCycleDelay.* A negative value will reduce the delay while a positive value will increase it. This call will have no effect on the Cloning operation in progress.

This Cloning feature can prove very helpful to users who don't want to perform single MsgDma calls to transfer a region from a card to a physical memory that is continuously changing. They can basically Clone the two regions and simply read the physical memory while the hardware is continuously updating it with the latest data from the card region at MsgDma rate. There is no CPU overhead during Cloning, however, it will be utilizing the PCI bus during its operation.

Only one Cloning or MsgDma operation can be active at a given time. Additionally, it is meaningless to perform Cloning on a FIFO region for two reasons. Firstly, each data in a FIFO is synchronous, however, the Cloned region is accessed asynchronously. Secondly, when the FIFO runs empty *(underflow)* or cannot accept more data *(overflow)* the results are unpredictable.

***Caution:** Since physical addresses are supplied for the MsgDma operation, care must be taken to ensure that the supplied addresses are valid and that while Cloning is in progress, the memory regions must not be freed or made inactive, otherwise, the results could be unpredictable and could lead to the possible corruption of the system.*

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   _ccurPMFC_MsgDma_Clone(void                                        *Handle,
                          _ccurpmfc_msgdma_clone_mode_mask_t ModeMask)

   Description: Clone Modular Scatter-Gather DMA

   Input:   void                                *Handle  (Handle pointer)
            _ccurpmfc_msgdma_clone_mode_mask_t  ModeMask (Mode Mask)
               # CCURPMFC_MSGDMA_CLONE_STOP
               # CCURPMFC_MSGDMA_CLONE_INITIALIZE
               # CCURPMFC_MSGDMA_CLONE_START
               # CCURPMFC_MSGDMA_CLONE_START_BLOCK
            ccurpmfc_msgdma_clone_args_t          *CloneArgs
               - unsigned long long               AdditionalOneCycleDelay
                                                  (Additional blocking for
                                                  Once Cycle Delay (nanoseconds))
   Output:  ccurpmfc_msgdma_clone_args_t          *CloneArgs
               - unsigned long long               MsgDmaExtDesOneCycleDelay
                                                  (MsgDma Extended Descriptor One
                                                  Cycle Delay (nanoseconds))
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_DMA_FAILED        (MsgDma failed)
```

```
                  # CCURPMFC_LIB_MSGDMA_IN_USE     (MsgDma in use)
                  # CCURPMFC_LIB_MSGDMA_NOT_SETUP  (MsgDma not setup)
                  # CCURPMFC_LIB_MSGDMA_ACCESS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                                   (MsgDma not allowed for
                                                    selected address)
                  # CCURPMFC_LIB_CLONING_NOT_SUPPORTED
                                                   (Cloning not supported by the
                                                    card)
    ************************************************************************/
```

2.2.177 ccurPMFC_MsgDma_Configure_Descriptor()

This call assists the user in setting up modular scatter-gather DMA descriptors. It allows the user to specify a read and write address offset along with length of transfer. Additionally, the call also provides the option to attach to other previously created descriptor blocks for scatter-gather operation. To perform scatter-gather DMA operation, the user creates a chain of descriptors, each having its own read/write/length information along with a start and end of the chain. The DMA operation is started from the first descriptor block in the chain and sequentially processes the descriptor blocks until the last descriptor block in the chain is processed.

To distinguish between descriptors, they are labeled with descriptor ID's. They range from ID 1 to 31. Users can supply a valid specific ID to this call or let the call itself find a free descriptor ID available. It is entirely left up to the user to determine how to manage the various descriptors and their relative linkages.

If the user wishes to have a previously created descriptor to point to a newly created descriptor, they can supply the previously created descriptor ID to the *AttachToDescriptorID* argument in the newly created descriptor. The newly created descriptor will not point to any descriptor and will always be the last descriptor in the chain.

DMA transfers can occur from either of the following:

1. Physical PCIe memory to Physical PCIe memory
2. Physical PCIe memory to Avalon Memory
3. Avalon Memory to Physical PCIe memory
4. Avalon Memory to Avalon Memory

There are certain restrictions and limitations to this scatter-gather operation:

1. Scatter-gather DMA is only supported in certain FPGA cards
2. Reads from Avalon memory below DiagRam location are not allowed for MIOC FPGA cards.
3. Invalid memory address supplied could result in the scatter-gather IP to lock up and the only way to recover will be to reload the driver.
4. Read and write addresses must be at a minimum full-word aligned and for maximum performance, it is recommended to be quad-word aligned.
5. Lengths are in bytes and must be at a minimum a multiple of a full-word and for maximum performance, it is recommended to be quad-word multiple.
6. You cannot cause a chain of descriptors to loop on itself.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_MsgDma_Configure_Descriptor (void              *Handle,
                         _ccurpmfc_msgdma_descriptors_id_t *DescriptorID,
                         ccurpmfc_msgdma_descriptor_t      *Descriptor,
                         _ccurpmfc_msgdma_descriptors_id_t AttachToDescriptorID)

   Description: Configure Modular Scatter-Gather DMA descriptor

   Input:   void                          *Handle (Handle pointer)
            _ccurpmfc_msgdma_descriptors_id_t *DescriptorID
                                           (Set to NULL or valid ID)
               # 0                         (let function find a free ID)
               # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1 ...
```

```
                CCURPMFC_MSGDMA_DESCRIPTOR_ID_31
             ccurpmfc_msgdma_descriptor_t *Descriptor (pointer to descriptor)
                __u64  ReadAddress
                __u64  WriteAddress
                __u32  Length
             _ccurpmfc_msgdma_descriptors_id_t AttachToDescriptorID
                                                  (Attach to descriptor ID)
                # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1 ...
                  CCURPMFC_MSGDMA_DESCRIPTOR_ID_31
   Output:   _ccurpmfc_msgdma_descriptors_id_t *DescriptorID (returned ID)
                # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1 ...
                  CCURPMFC_MSGDMA_DESCRIPTOR_ID_31
   Return:   _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR              (successful)
                # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN              (device not open)
                # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
                # CCURPMFC_LIB_NO_FREE_DESCRIPTORS_AVAILABLE
                                                  (no free descriptors available)
                # CCURPMFC_LIB_MSG_DMA_NOT_SUPPORTED (modular scatter-gather
                                                    DMA not supported)
                # CCURPMFC_LIB_MSG_DMA_READS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                                   (MSG DMA Reads not allowed
                                                    for selected address)
                # CCURPMFC_LIB_NOT_OWNER_OF_MSGDMA   (not owner of modular
                                                    scatter-gather)
 ***************************************************************************/
```

2.2.178 ccurPMFC_MsgDma_Configure_Single()

This call performs a similar function to the *ccurPMFC_MsgDma_Configure()* call with the exception that no DMA chaining is performed and only the single descriptor ID-1 is used to perform the DMA operation. The user has the option to supply a valid descriptor block when using the *ccurPMFC_MsgDma_Configure_Single()* API or a *NULL* pointer to the descriptor as an argument when using the *ccurPMFC_Transfer_Data()* API to perform the transfer.

Normally this call needs to be issued once with a *NULL* pointer for the *Descriptor (i.e. during initialization)* prior to using the *ccurPMFC_Transfer_Data()* call with the *LibMode* set to *CCURPMFC_LIBRARY_MSGDMA_MOD*. In this way, the descriptor ID-1 will be set up correctly prior to the transfer.

If instead, the user wishes to perform the DMA operation using the *ccurPMFC_MsgDma_Fire_Single()* call, they need to issue the *ccurPMFC_MsgDma_Configure_Single()* call with a valid descriptor block, otherwise, results will be unpredictable.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_MsgDma_Configure_Single (void                              *Handle,
                                ccurpmfc_msgdma_descriptor_t           *Descriptor)

   Description: Configure Single Modular Scatter-Gather DMA descriptor

   Input:   void                          *Handle (Handle pointer)
            ccurpmfc_msgdma_descriptor_t *Descriptor (pointer to descriptor)
               __u64  ReadAddress
               __u64  WriteAddress
               __u32  Length
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
```

```
                # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN             (device not open)
                # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
                # CCURPMFC_LIB_DMA_BUSY             (MsgDma Busy, cannot be
                                                     reset)
                # CCURPMFC_LIB_MSG_DMA_NOT_SUPPORTED (modular scatter-gather DMA
                                                     not supported)
                # CCURPMFC_LIB_MSG_DMA_READS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                                    (MSG DMA Reads not allowed
                                                     for selected address)
 ***************************************************************************/
```

2.2.179 ccurPMFC_MsgDma_Fire()

This call initiates a scatter-gather DMA operation that has been previously configured and setup by the *ccurPMFC_MsgDma_Configure()* and *ccurPMFC_MsgDma_Setup()* call.

The *StartDescriptorID* can be set to either '0' or a valid Descriptor ID. Normally, the user will set the *StartDescriptiorID* in the *ccurPMFC_MsgDma_Setup()* API during initialization and set it to '0' in this *ccurPMFC_MsgDma_Fire()* API. In this way, this call will not suffer the overhead of loading the *StartDescriptorID* in the internal prefetcher register when repeatedly calling the *ccurPMFC_MsgDma_Fire()* API. If the user specifies a valid *StartDescriptorID* that is already setup as a scatter-gather chain using the *ccurPMFC_MsgDma_Configure()* call, then this *ccurPMFC_MsgDma_Fire()* API will initiate the DMA starting with the user supplied start descriptor ID.
The *DescriptorIDMask* is a mask of all the valid descriptor ID's specified in the scatter-gather chain that was created earlier with the *ccurPMFC_MsgDma_Configure()* API. If this is incorrectly specified, the DMA operation will be unpredictable. This *ccurPMFC_MsgDma_Fire()* API call uses this mask to set the *ControlWord* for each of the IDs. Specifying this mask reduces the overhead in the call by not searching the scatter-gather chain to set the individual control words.

ControlWord for each descriptor is set based on the *DescriptorIDMask* mask. Normally, the following two flags are set:

- CCURPMFC_MSGD_DESC_CONTROL_GO
- CCURPMFC_MSGD_DESC_CONTROL_OWNED_BY_HW

LastIdForInterrupts is set to 0 if the DMA operation will use polling instead of using interrupts to detect completion of the operation. If interrupts are to be used, the ID of the last descriptor in the DMA chain is to be specified. This is the ID that will be interrupted when the entire chain is completed. Incorrect ID entered will result in unpredictable results. Normally, interrupt handling adds additional overhead and reduces performance, however, it reduces the overhead experienced by the CPU and PCIe bus during polling.

Once the scatter-gather DMA operation commences, it performs DMA operations starting with the *StartDescriptorID* and traversing through the chain sequentially until it reaches the last descriptor ID in the chain, at which point the DMA operation concludes.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   _ccurPMFC_MsgDma_Fire (void                                *Handle,
                     _ccurpmfc_msgdma_descriptors_id_t        StartDescriptorID,
                     _ccurpmfc_msgdma_descriptors_id_mask_t   DescriptorIDMask,
                     int                                      ControlWord,
                     _ccurpmfc_msgdma_descriptors_id_t
                                                              LastIdForInterrupts)

   Description: Fire Modular Scatter-Gather DMA descriptor

   Input:   void                                  *Handle (Handle pointer)
            _ccurpmfc_msgdma_descriptors_id_t     StartDescriptorID (Set to
                                                  valid ID)
```

```
               # 0                                  (don't set start descriptor ID
                                                     in prefetcher)
               # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1 ...
                 CCURPMFC_MSGDMA_DESCRIPTOR_ID_31
            _ccurpmfc_msgdma_descriptors_id_mask_t DescriptorIDMask
                                                  (descriptor ID mask)
               # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1_MASK ...
                 CCURPMFC_MSGDMA_DESCRIPTOR_ID_31_MASK
               # CCURPMFC_MSGDMA_DESCRIPTOR_ID_ALL_MASK
            int                                   ControlWord
               # CCURPMFC_MSGD_DESC_CONTROL_GO
               # CCURPMFC_MSGD_DESC_CONTROL_OWNED_BY_HW
            _ccurpmfc_msgdma_descriptors_id_t     LastIdForInterrupts (Set 0 or
                                                  Last ID for interrupts)
               # 0
               # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1 ...
                 CCURPMFC_MSGDMA_DESCRIPTOR_ID_31
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
               # CCURPMFC_LIB_DMA_FAILED        (MsgDma failed)
               # CCURPMFC_LIB_DMA_BUSY          (MsgDma busy)
*************************************************************************/
```

2.2.180 ccurPMFC_MsgDma_Fire_Single()

This call is similar in functionality to the *ccurPMFC_MsgDma_Fire()* call with the exception that it operates on the single descriptor ID-1. It can be used when a single DMA rather than scatter-gather DMA operation needs to be performed. This call can be called once the *ccurPMFC_MsgDma_Configure_Single()* call has been issued to set up the read/write address offset and length of transfer. Unless the read/write address offset or length of transfer is changed, the *ccurPMFC_MsgDma_Fire_Single()* call can be made repeatedly to perform the same DMA transfer.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   _ccurPMFC_MsgDma_Fire_Single (void *Handle,
                                 int  UseInterrupts)

   Description: Fire Single Modular Scatter-Gather DMA descriptor

   Input:   void  *Handle                   (Handle pointer)
            int   UseInterrupts             (Use interrupts flag)
               # CCURPMFC_TRUE
               # CCURPMFC_FALSE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_DMA_FAILED           (MsgDma failed)
               # CCURPMFC_LIB_DMA_BUSY             (MsgDma busy)
               # CCURPMFC_LIB_NOT_OWNER_OF_MSGDMA  (not owner of modular
                                                    scatter-gather)
 *************************************************************************/
```

UseInterrupts is a flag that can be set to specify if interrupt handling should be enabled.

2.2.181 ccurPMFC_MsgDma_Free_Descriptor()

This call can be used to free up already used descriptors.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_MsgDma_Free_Descriptor (void                          *Handle,
                        _ccurpmfc_msgdma_descriptors_id_mask_t DescriptorIDMask)

   Description: Free Modular Scatter-Gather DMA descriptor

   Input:   void                                  *Handle (Handle pointer)
            _ccurpmfc_msgdma_descriptors_id_mask_t DescriptorIDMask
                                                  (descriptor ID mask)
               # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1_MASK ...
                 CCURPMFC_MSGDMA_DESCRIPTOR_ID_31_MASK
               # CCURPMFC_MSGDMA_DESCRIPTOR_ID_ALL_MASK
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (device not open)
               # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
               # CCURPMFC_LIB_MSG_DMA_NOT_SUPPORTED (modular scatter-gather DMA
                                                     not supported)
               # CCURPMFC_LIB_DMA_BUSY              (MsgDma busy)
               # CCURPMFC_LIB_NOT_OWNER_OF_MSGDMA   (not owner of modular
                                                     scatter-gather)
 *****************************************************************************/
```

2.2.182 ccurPMFC_MsgDma_Get_Descriptor()

This call returns information on the selected descriptor.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_MsgDma_Get_Descriptor (void                        *Handle,
                           _ccurpmfc_msgdma_descriptors_id_t DescriptorID,
                           ccurpmfc_msgdma_descriptor_t      *Descriptor,
                           __u64                             *DescriptorAddress)

   Description: Get Modular Scatter-Gather DMA Descriptor

   Input:   void                                *Handle (Handle pointer)
            _ccurpmfc_msgdma_descriptors_id_t DescriptorID (descriptor ID)
               # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1 ...
                 CCURPMFC_MSGDMA_DESCRIPTOR_ID_31
   Output:  ccurpmfc_msgdma_descriptor_t *Descriptor (pointer to descriptor)
               __u64  ReadAddress
               __u64  WriteAddress
               __u64  NextDescriptorPointer
               __u32  Length
               __u32  Control
               __u32  ReadBurstCount
               __u32  WriteBurstCount
               __u32  ReadStride
               __u32  WriteStride
               __u32  ActualBytesTransferred
               __u32  Status
            __u64                          *DescriptorAddress (descriptor address)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (device not open)
               # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
```

```
            # CCURPMFC_LIB_MSG_DMA_NOT_SUPPORTED (modular scatter-gather DMA
                                                  not supported)
*************************************************************************/
```

Pointer to *DescriptorAddress* can be specified to return its address offset within the configuration space. This argument can be set to *NULL* if address is not required.

2.2.183 ccurPMFC_MsgDma_Get_Dispatcher_CSR()

This call returns useful control and status register information on the dispatcher.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_MsgDma_Get_Dispatcher_CSR (void                          *Handle,
                                       ccurpmfc_msgdma_dispatcher_t  *Dispatcher)

   Description: Get Modular Scatter-Gather DMA Dispatcher CSR

   Input:   void                              *Handle (Handle pointer)
   Output:  ccurpmfc_msgdma_dispatcher_t      *Dispatcher (pointer to dispatcher)
               __u32  Status
                  # CCURPMFC_MSGD_DISP_STATUS_IRQ            :IRQ
                  # CCURPMFC_MSGD_DISP_STATUS_STOPPED_ETERM  :Stopped on Early
                                                              Termination
                  # CCURPMFC_MSGD_DISP_STATUS_STOPPED_ERROR  :Stopped on Error
                  # CCURPMFC_MSGD_DISP_STATUS_RESETTING      :Resetting
                  # CCURPMFC_MSGD_DISP_STATUS_STOPPED        :Stopped
                  # CCURPMFC_MSGD_DISP_STATUS_RESP_BUF_FULL  :Response Buffer
                                                              Full
                  # CCURPMFC_MSGD_DISP_STATUS_RESP_BUF_EMPTY :Response Buffer
                                                              Empty
                  # CCURPMFC_MSGD_DISP_STATUS_DESC_BUF_FULL  :Descriptor Buffer
                                                              Full
                  # CCURPMFC_MSGD_DISP_STATUS_DESC_BUF_EMPTY :Descriptor Buffer
                                                              Empty
                  # CCURPMFC_MSGD_DISP_STATUS_BUSY           :Busy
               __u32  Control
                  # CCURPMFC_MSGD_DISP_CONTROL_STOP_DESC     :Stop Descriptors
                  # CCURPMFC_MSGD_DISP_CONTROL_INT_ENA_MASK  :Global Interrupt
                                                              Enable Mask
                  # CCURPMFC_MSGD_DISP_CONTROL_STOP_ETERM    :Stop on Early
                                                              Termination
                  # CCURPMFC_MSGD_DISP_CONTROL_STOP_ON_ERROR :Stop on Error
                  # CCURPMFC_MSGD_DISP_CONTROL_RESET_DISP    :Reset Dispatcher
                  # CCURPMFC_MSGD_DISP_CONTROL_STOP_DISP     :Stop Dispatcher
               __u32  ReadFillLevel
               __u32  WriteFillLevel
               __u32  ResponseFillLevel
               __u32  ReadSequenceNumber
               __u32  WriteSequenceNumber
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN              (device not open)
               # CCURPMFC_LIB_INVALID_ARG           (invalid argument)
               # CCURPMFC_LIB_MSG_DMA_NOT_SUPPORTED (modular scatter-gather DMA
                                                     not supported)
 *************************************************************************/
```

2.2.184 ccurPMFC_MsgDma_Get_Prefetcher_CSR()

This call returns useful control and status register information on the prefetcher.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_MsgDma_Get_Prefetcher_CSR (void                          *Handle,
                                       ccurpmfc_msgdma_prefetcher_t  *Prefetcher)

   Description: Get Modular Scatter-Gather DMA Prefetcher CSR

   Input:   void                              *Handle (Handle pointer)
   Output:  ccurpmfc_msgdma_prefetcher_t      *Prefetcher (pointer to prefetcher)
               __u32 Status
                  # CCURPMFC_MSGD_PREF_STATUS_IRQ          :IRQ Occurred
               __u32 Control
                  # CCURPMFC_MSGD_PREF_CONTROL_PARK_MODE    :Park Mode
                  # CCURPMFC_MSGD_PREF_CONTROL_INT_ENA_MASK :Global Interrupt
                                                             Enable Mask
                  # CCURPMFC_MSGD_PREF_CONTROL_RESET        :Reset Prefetcher
                                                             Core
                  # CCURPMFC_MSGD_PREF_CONTROL_DESC_POLL_EN :Descriptor Polling
                                                             Enable
                  # CCURPMFC_MSGD_PREF_CONTROL_RUN          :Start Descriptor
                                                             fetching operation
               __u64 NextDescriptorPointer
               __u32 DescriptorPollingFrequency
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_MSG_DMA_NOT_SUPPORTED (modular scatter-gather DMA
                                                     not supported)
*****************************************************************************/
```

2.2.185 ccurPMFC_MsgDma_Release()

This *ccurPMFC_MsgDma_Release()* API call is used to free up the Modular Scatter-Gather DMA resource that has been previously reserved by the *ccurPMFC_MsgDma_Seize()* API. At this point, another user can take control of the MsgDMA operation by issuing the *ccurPMFC_MsgDma_Seize()* call.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t   ccurPMFC_MsgDma_Release (void *Handle)

   Description: Release MsgDMA operation for others to use

   Input:   void                                  *Handle (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA
                                                    not supported)
               # CCURPMFC_LIB_DMA_BUSY             (MsgDma Busy, cannot be
                                                    reset)
               # CCURPMFC_LIB_NOT_OWNER_OF_MSGDMA  (not owner of modular
                                                    scatter-gather)
*****************************************************************************/
```

2.2.186 ccurPMFC_MsgDma_Seize()

Modular Scatter-Gather DMA is a two part operation. The first part is to configure the Scatter-Gather DMA and the second part is to execute the DMA. Various MsgDma API calls have been provided for this. Since this two part operation is not atomic, it is necessary for the user of these calls to prevent other applications from configuring and using the MsgDMA resources while it is being actively used by another application. For this reason, the *ccurPMFC_MsgDma_Seize()* and *ccurPMFC_MsgDma_Release()* API calls have been introduced to assist the user in preventing other applications from accessing the Scatter-Gather DMA resource while it is reserved. Basically, before any MsgDma API call is issued that could modify the setting and execution of the MsgDma operation, the user needs to issue the *ccurPMFC_MsgDma_Seize()* API call once. In this way, no one else will have access to the MsgDma resource until the application has issued the *ccurPMFC_MsgDma_Release()* API call or has terminated the application.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t ccurPMFC_MsgDma_Seize (void *Handle)

   Description: Seize MsgDMA operation for private to use and become owner

   Input:   void                                        *Handle (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA
                                                    not supported)
               # CCURPMFC_LIB_MSGDMA_IN_USE        (modular scatter-gather DMA in
                                                    use)
*************************************************************************/
```

2.2.187 ccurPMFC_MsgDma_Setup()

This call is used in conjunction with the *ccurPMFC_MsgDma_Configure() and ccurPMFC_MsgDma_Fire()* calls. This call is made after all the descriptors are first configured with the help of the *ccurPMFC_MsgDma_Configure()* call. The purpose of this call is to specify the first descriptor in the chain. Additionally, the user can set the *ForceReset* flag to reset the dispatcher and prefetcher. Optionally, the user can request useful active descriptor information if *ActiveDescriptorsInfo* argument is specified *(i.e not NULL)*. In addition to returning useful active descriptor information, the descriptor chain and prefetcher settings are also validated for proper configuration.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_MsgDma_Setup (void                                *Handle,
                     _ccurpmfc_msgdma_descriptors_id_t  StartDescriptorID,
                     int                                ForceReset,
                     ccurpmfc_msgdma_active_descriptors_info_t
                                                        *ActiveDescriptorsInfo)

   Description: Setup MsgDMA Dispatcher and Prefetcher

   Input:   void                                        *Handle (Handle pointer)
            _ccurpmfc_msgdma_descriptors_id_t           *StartDescriptorID (Set
                                                        to valid ID)
               # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1 ...
                 CCURPMFC_MSGDMA_DESCRIPTOR_ID_31
            int                                         ForceReset
   Output:  ccurpmfc_msgdma_active_descriptors_info_t   *ActiveDescriptorsInfo;
               _ccurpmfc_msgdma_descriptors_id_t       ID
                   # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1 ...
                     CCURPMFC_MSGDMA_DESCRIPTOR_ID_31
```

```
                _ccurpmfc_msgdma_descriptors_id_mask_t  Mask
                    # CCURPMFC_MSGDMA_DESCRIPTOR_ID_1_MASK ...
                      CCURPMFC_MSGDMA_DESCRIPTOR_ID_31_MASK
                    # CCURPMFC_MSGDMA_DESCRIPTOR_ID_ALL_MASK
                __u32                                   NumberOfDescriptors
                __u32                                   TotalBytes
   Return:  _ccurpmfc_lib_error_number_t
                # CCURPMFC_LIB_NO_ERROR                 (successful)
                # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
                # CCURPMFC_LIB_NOT_OPEN                 (device not open)
                # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
                # CCURPMFC_LIB_DMA_BUSY                 (MsgDma Busy, cannot be
                                                         reset)
                # CCURPMFC_LIB_ERROR_IN_DESCRIPTOR_LIST (invalid descreptor list)
                # CCURPMFC_LIB_MSG_DMA_NOT_SUPPORTED    (modular scatter-gather
                                                         DMA not supported)
                # CCURPMFC_LIB_NOT_OWNER_OF_MSGDMA      (not owner of modular
                                                         scatter-gather)
*****************************************************************************/
```

2.2.188 ccurPMFC_Munmap_Physical_Memory()

This call simply removes a physical memory that was previously allocated by the *ccurPMFC_MMap_Physical_Memory()* API call.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Munmap_Physical_Memory (void   *Handle,
                                    void   *mmaped_user_mem_ptr)

   Description: Unmap a previously mapped physical DMA memory.

   Input:   void            *Handle                (Handle pointer)
   Output:  void            *mmaped_user_mem_ptr   (virtual memory pointer)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_MUNMAP_FAILED        (failed to un-map memory)
               # CCURPMFC_LIB_NOT_MAPPED           (memory not mapped)
               # CCURPMFC_LIB_MSGDMA_IN_USE        (modular scatter-gather DMA
                                                    in use)
 *****************************************************************************/
```

2.2.189 ccurPMFC_NanoDelay()

This call goes into a tight loop spinning for the requested nano seconds specified by the user.

```
/*****************************************************************************
   void
   ccurPMFC_NanoDelay (unsigned long long NanoDelay)

   Description: Delay (loop) for user specified nano-seconds

   Input:   unsigned long long NanoDelay       (number of nano-secs to delay)
   Output:  none
   Return:  none
 *****************************************************************************/
```

2.2.190 ccurPMFC_Open()

This is the first call that needs to be issued by a user to open a device and access the board through the rest of the API calls. What is returned is a handle to a *void pointer* that is supplied as an argument to the other API calls. The *Board_Number* is a valid board number [0..9] that is associated with a physical card. There must exist a character special file */dev/ccurpmfc<Board_Number>* for the call to be successful. One character special file is created for each board found when the driver is successfully loaded.

The *oflag* is the flag supplied to the *open(2)* system call by this API. It is normally '0' *(zero)*, however the user may use the *O_NONBLOCK* option for *read(2)* calls which will change the default reading in block mode.

This driver allows multiple applications to open the same board by specifying an additional *oflag O_APPEND*. It is then the responsibility of the user to ensure that the various applications communicating with the same cards are properly synchronized. Various tests supplied in this package has the *O_APPEND* flags enabled, however, it is strongly recommended that only one application be run with a single card at a time, unless the user is well aware of how the applications are going to interact with each other and accept any unpredictable results.

In case of error, *errno* is also set for some non-system related errors encountered.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Open (void     **My_Handle,
                  int      Board_Number,
                  int      oflag)

   Description: Open a device.

   Input:   void            **Handle               (Handle pointer to pointer)
            int             Board_Number           (0-9 board number)
            int             oflag                  (open flags)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_ALREADY_OPEN         (device already opened)
               # CCURPMFC_LIB_OPEN_FAILED          (device open failed)
               # CCURPMFC_LIB_ALREADY_MAPPED       (memory already mmapped)
               # CCURPMFC_LIB_MMAP_SELECT_FAILED   (mmap selection failed)
               # CCURPMFC_LIB_MMAP_FAILED          (mmap failed)
 *****************************************************************************/
```

2.2.191 ccurPMFC_Pause_UserProcess()

This call causes a running User Process to sleep for user specified micro-seconds. *(This is an experimental API for debugging and testing).*

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Pause_UserProcess(void *UFuncHandle,
                              int  usleep)

   Description: Pause running user process

   Input:   void                      *UFuncHandle (UF Handle pointer)
            int                       usleep       (micro-seconds sleep)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
```

```
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
 ***************************************************************************/
```

2.2.192 ccurPMFC_Program_All_Output_Clocks()

This is the main call to program all the output clocks with a single call. All existing clock activity is stopped and replaced with the new clocks selection. Though the user can select the Input Clock Frequency with this call, it is expected that they will use the default CCURPMFC_DEFAULT_INPUT_CLOCK_FREQUENCY value.

The input clock can be one of:

```
CCURPMFC_CG_INPUT_CLOCK_SELECT_IN0    → 10MHz TCXO (Temperature Compensated
                                        Oscillator Clock).
CCURPMFC_CG_INPUT_CLOCK_SELECT_IN1    → External Input
CCURPMFC_CG_INPUT_CLOCK_SELECT_IN2    → FPGA Supplied
CCURPMFC_CG_INPUT_CLOCK_SELECT_INXAXB → Not used
```

When using this card, the default clock should be set to *CCURPMFC_CG_INPUT_CLOCK_SELECT_N0* i.e. the 10MHz internal clock.

If the desired output clock frequencies are unable to be computed due to hardware limitation, they may wish to increase the desired tolerance *DesiredTolerancePPT* for the particular clock. Note that this tolerance is only applicable to computing a clock value as close to the desired frequency *DesiredFrequency* and not a representation of the accuracy of the output clocks.

Additionally, the programming could fail if the number of N-Divider resource gets exhausted due to the user selecting several output clocks with widely different output clocks.

```
/***************************************************************************
   ccurPMFC_Program_All_Output_Clocks()
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Program_All_Output_Clocks(void                    *Handle,
                                      double                  InputClockFrequency,
                _ccurpmfc_cg_input_clock_select_register_t    InputClockSel,
                ccurpmfc_compute_all_output_clocks_t          *AllClocks,
                int                                           ProgramClocks,
                int                                           ActivateClocks)

   Description: Program All Output Clocks

   Input:   void                                          *Handle
                                                          (Handle pointer)
            double                                        InputClockFrequency
                                                          (input clock frequency)
            _ccurpmfc_cg_input_clock_select_register_t    InputClockSel
                                                          (select input clock)
               # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN0
               # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN1
               # CCURPMFC_CG_INPUT_CLOCK_SELECT_IN2
               # CCURPMFC_CG_INPUT_CLOCK_SELECT_INXAXB
            ccurpmfc_compute_all_output_clocks_t       *AllClocks    (pointer to
                                                                     all Clocks)
               ccurpmfc_compute_single_output_clock_t  *Clock        (Pointer to
                                                      returned output clock info)
                  long double                          DesiredFrequency
                  double                               DesiredTolerancePPT
             int                                       ProgramClocks  (program
                                                                      clocks)
             int                                       ActivateClocks (1=activate
```

```
                                                        clocks after program)
Output:   ccurpmfc_compute_all_output_clocks_t      *AllClocks     (Pointer to
                                                   returned output clocks info)
             ccurpmfc_compute_single_output_clock_t *Clock         (Pointer to
                                                   returned output clock info)
                _ccurpmfc_clock_generator_output_t  OutputClock
                   # CCURPMFC_CLOCK_GENERATOR_OUTPUT_0
                   # CCURPMFC_CLOCK_GENERATOR_OUTPUT_1
                   # CCURPMFC_CLOCK_GENERATOR_OUTPUT_2
                   # CCURPMFC_CLOCK_GENERATOR_OUTPUT_3
                   # CCURPMFC_CLOCK_GENERATOR_OUTPUT_4
                   # CCURPMFC_CLOCK_GENERATOR_OUTPUT_5
                   # CCURPMFC_CLOCK_GENERATOR_OUTPUT_6
                   # CCURPMFC_CLOCK_GENERATOR_OUTPUT_7
                   # CCURPMFC_CLOCK_GENERATOR_OUTPUT_8
                   # CCURPMFC_CLOCK_GENERATOR_OUTPUT_9
                 double                                InputClockFrequency
                 long double                           FrequencyDeviation
                 int                                   FrequencyFound
                 long double                           ActualFrequency
                 double                                ActualTolerancePPT
                 __u64                                 Mdiv_Numerator
                 __u32                                 Mdiv_Denominator
                 __u64                                 Ndiv_Numerator
                 __u32                                 Ndiv_Denominator
                 _ccurpmfc_cg_outmux_ndiv_select_t     Ndiv_ToUse
                    # CCURPMFC_CG_OUTPUT_MUX_NDIV_0
                    # CCURPMFC_CG_OUTPUT_MUX_NDIV_1
                    # CCURPMFC_CG_OUTPUT_MUX_NDIV_2
                    # CCURPMFC_CG_OUTPUT_MUX_NDIV_3
                    # CCURPMFC_CG_OUTPUT_MUX_NDIV_4
                 __u32                                     Rdiv_value
                 __u32                                     Rdivider
                 __u32                                     Pdivider
Return:   _ccurpmfc_lib_error_number_t
             # CCURPMFC_LIB_NO_ERROR                (successful)
             # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
             # CCURPMFC_LIB_NOT_OPEN                (device not open)
             # CCURPMFC_LIB_NO_LOCAL_REGION         (local region error)
             # CCURPMFC_LIB_IO_ERROR                (device not ready)
             # CCURPMFC_LIB_N_DIVIDERS_EXCEEDED     (number of N-Dividers
                                                     exceeded)
             # CCURPMFC_LIB_CANNOT_COMPUTE_OUTPUT_FREQ (cannot compute
                                                        output freq)
             # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
             # CCURPMFC_LIB_CLOCK_GENERATION_FAILED  (clock generation
                                                     failed)
*****************************************************************************/
```

2.2.193 ccurPMFC_Read()

This call performs a programmed I/O driver read of either the ADC *channel registers* or the *FIFO*. Prior to issuing this call, the user needs to set up the desired read mode of operation using the *ccurPMFC_ADC_Set_Driver_Read_Mode()* with *CCURPMFC_ADC_PIO_CHANNEL* or *CCURPMFC_ADC_PIO_FIFO* argument. For *channel register* reads, the size is limited to *CCURPMFC_MAX_ADC_CHANNELS* words and for *FIFO* reads, it is limited to *CCURPMFC_ADC_FIFO_DATA_MAX* words.

It basically calls the *read(2)* system call with the exception that it performs necessary *locking* and returns the *errno* returned from the system call in the pointer to the *error* variable. An *errno* of *ENOBUFS* can occur for *FIFO* reads when it encounters an overflow condition.

For specific information about the data being returned for the various read modes, refer to the *read(2)* system call description the *Driver Direct Access* section.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Read (void     *Handle,
                  void     *buf,
                  int      size,
                  int      *bytes_read,
                  int      *error)

   Description: Perform a read operation.

   Input:   void     *Handle               (Handle pointer)
            int      size                  (size of buffer in bytes)
   Output:  void     *buf                  (pointer to buffer)
            int      *bytes_read           (bytes read)
            int      *error                (returned errno)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR      (successful)
               # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN      (device not open)
               # CCURPMFC_LIB_IO_ERROR      (read failed)
 ************************************************************************/
```

2.2.194 ccurPMFC_Reload_Firmware()

The purpose of this call is to power cycle the board which in turn will reload the latest firmware on the board.

```
/************************************************************************
   ccurPMFC_Reload_Firmware()

   Description: This call power-cycles the board which in turn forces it to
                reload its firmware. Typically, this is called after a new
                firmware has been installed in the board. This saves the need
                to perform a system reboot after a firmware installation.

   Input:   void *Handle                   (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR      (successful)
               # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN      (device not open)
               # CCURPMFC_LIB_IOCTL_FAILED  (driver ioctl call failed)
 ************************************************************************/
```

2.2.195 ccurPMFC_Remove_Irq()

The purpose of this call is to remove the interrupt handler that was previously set up. The interrupt handler is managed internally by the driver and the library. The user should not issue this call, otherwise reads will time out.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Remove_Irq (void *Handle)

   Description: By default, the driver sets up a shared IRQ interrupt handler
                when the device is opened. Now if for any reason, another
                device is sharing the same IRQ as this driver, the interrupt
                handler will also be entered every time the other shared
```

```
               device generates an interrupt. There are times that a user,
               for performance reasons may wish to run the board without
               interrupts enabled. In that case, they can issue this ioctl
               to remove the interrupt handling capability from the driver.

   Input:   void *Handle                    (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR      (successful)
               # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN      (device not open)
               # CCURPMFC_LIB_IOCTL_FAILED  (driver ioctl call failed)
 ****************************************************************************/
```

2.2.196 ccurPMFC_Reset_Board()

This call resets the board to a known hardware state. It may be a good idea to start an application by first resetting the board so that it is set to a known state.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Reset_Board (void *Handle)

   Description: Reset the board.

   Input:   void *Handle                      (Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR        (successful)
               # CCURPMFC_LIB_BAD_HANDLE      (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN        (device not open)
               # CCURPMFC_LIB_IOCTL_FAILED    (driver ioctl call failed)
               # CCURPMFC_LIB_NO_LOCAL_REGION (local region not present)
 ****************************************************************************/
```

2.2.197 ccurPMFC_Reset_Clock()

This call performs a hardware reset of the clock. All active output clocks are stopped and set to default state. The user can activate clocks if they wish after a reset via the *activate* argument.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Reset_Clock (void  *Handle,
                         int   activate)

   Description: Perform Hardware Clock Reset

   Input:   void                    *Handle   (Handle pointer)
            int                     activate  (1=activate after reset)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR        (successful)
               # CCURPMFC_LIB_BAD_HANDLE      (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN        (device not open)
               # CCURPMFC_LIB_IOCTL_FAILED    (driver ioctl call failed)
               # CCURPMFC_LIB_NO_LOCAL_REGION (local region not present)
 ****************************************************************************/
```

2.2.198 ccurPMFC_Resume_UserProcess()

Use this call to resume an already paused User Process. *(This is an experimental API for debugging and testing).*

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Resume_UserProcess(void *UFuncHandle)

   Description: Resume paused running user process

   Input:   void         *UFuncHandle           (UF Handle pointer)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
 *****************************************************************************/
```

2.2.199 ccurPMFC_Return_Board_Info_Description()

Return board information description

```
/*****************************************************************************
  char *
  ccurPMFC_Return_Board_Info_Description (_ccurpmfc_board_function_t
                                                          BoardFunction)

   Description: Return Board Information Description

   Input:   _ccurpmfc_board_function_t   BoardFunction          (board function)
               # CCURPMFC_BOARD_FUNCTION_MULTIFUNCTION_IO
               # CCURPMFC_BOARD_FUNCTION_ENGINE_CONTROL
               # CCURPMFC_BOARD_FUNCTION_IPCORE_COS
               # CCURPMFC_BOARD_FUNCTION_BASE_LEVEL
               # CCURPMFC_BOARD_FUNCTION_CUSTOM_IPCORE
               # CCURPMFC_BOARD_FUNCTION_UNDEFINED
   Output:  none
   Return:  char                         *BoardFuncDesc         (board function
                                                                   description)
 *****************************************************************************/
```

2.2.200 ccurPMFC_SDRAM_Activate()

This call must be the first call to activate the SDRAM. Without activation, all other calls will fail. The user can also use this call to return the current state of the SDRAM without any change.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_SDRAM_Activate (void                          *Handle,
                            _ccurpmfc_sdram_all_enable_t  activate,
                            _ccurpmfc_sdram_all_enable_t  *current_state)

   Description: Activate/DeActivate SDRAM module

   Input:   void                          *Handle         (Handle pointer)
            _ccurpmfc_sdram_all_enable_t   activate       (activate/deactivate)
               # CCURPMFC_SDRAM_ALL_DISABLE
               # CCURPMFC_SDRAM_ALL_ENABLE
               # CCURPMFC_SDRAM_ALL_ENABLE_DO_NOT_CHANGE
   Output:  _ccurpmfc_sdram_all_enable_t   *current_state  (active/deactive)
               # CCURPMFC_SDRAM_ALL_DISABLE
```

```
            # CCURPMFC_SDRAM_ALL_ENABLE
   Return:  _ccurpmfc_lib_error_number_t
            # CCURPMFC_LIB_NO_ERROR                (successful)
            # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN                (device not open)
            # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
            # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
            # CCURPMFC_LIB_SDRAM_INITIALIZATION_FAILED (SDRAM init failed)
 ***************************************************************************/
```

2.2.201 ccurPMFC_SDRAM_Get_CSR()

This call returns the SDRAM control and status register information.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_SDRAM_Get_CSR (void                    *Handle,
                           ccurpmfc_sdram_csr_t    *sdram_csr)

   Description: Get SDRAM Control and Status information

   Input:   void                                  *Handle    (Handle pointer)
   Output:  ccurpmfc_sdram_csr_t                  *sdram_csr (pointer to SDRAM csr)
               _ccurpmfc_sdram_read_auto_increment_t   read_auto_increment;
                  # CCURPMFC_SDRAM_READ_AUTO_INCREMENT_DISABLE
                  # CCURPMFC_SDRAM_READ_AUTO_INCREMENT_ENABLE
               _ccurpmfc_sdram_write_auto_increment_t  write_auto_increment;
                  # CCURPMFC_SDRAM_WRITE_AUTO_INCREMENT_DISABLE
                  # CCURPMFC_SDRAM_WRITE_AUTO_INCREMENT_ENABLE
               _ccurpmfc_sdram_read_timeout_t          read_timeout;
                  # CCURPMFC_SDRAM_READ_TIMEOUT_DID_NOT_OCCUR
                  # CCURPMFC_SDRAM_READ_TIMEOUT_OCCURRED
               _ccurpmfc_sdram_calibration_fail_t      calibration_failed;
                  # CCURPMFC_SDRAM_CALIBRATION_FAIL_RESET
                  # CCURPMFC_SDRAM_CALIBRATION_FAIL_SET
               _ccurpmfc_sdram_calibration_pass_t      calibration_passed;
                  # CCURPMFC_SDRAM_CALIBRATION_PASS_RESET
                  # CCURPMFC_SDRAM_CALIBRATION_PASS_SET
               _ccurpmfc_sdram_initilization_done_t    initialization_done;
                  # CCURPMFC_SDRAM_INITIALIZATION_NOT_COMPLETE
                  # CCURPMFC_SDRAM_INITIALIZATION_COMPLETE
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_SDRAM_IS_NOT_ACTIVE     (SDRAM is not active)
 ***************************************************************************/
```

2.2.202 ccurPMFC_SDRAM_Read()

This call provided the user the ability to read any portion of the SDRAM. Its range is from 1 to 0x10000000 (256Mwords). Offset to this routine is only set if it is 0 or greater. Maximum offset is 0x0FFFFFFF. If offset is negative, then the read commences from the last read location.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_SDRAM_Read (void      *Handle,
                        u_int32_t *Buf,
                        int       Offset,
```

```
                        u_int32_t Size,
                        u_int32_t *Words_read)

   Description: Perform a SDRAM read operation.

   Input:   void                        *Handle     (Handle pointer)
            int                         Offset      (word offset into SDRAM)
            u_int32_t                   Size        (size of buffer in words)
   Output:  u_int32_t                   *Buf        (pointer to buffer)
            u_int32_t                   *Words_read (words read)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR              (successful)
               # CCURPMFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURPMFC_LIB_SDRAM_IS_NOT_ACTIVE   (SDRAM is not active)

 ***************************************************************************/
```

2.2.203 ccurPMFC_SDRAM_Set_CSR()

This call sets the SDRAM control and status register.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_SDRAM_Set_CSR (void                    *Handle,
                           ccurpmfc_sdram_csr_t    *sdram_csr)

   Description: Set SDRAM Control and Status information

   Input:   void                                  *Handle    (Handle pointer)
               _ccurpmfc_sdram_read_auto_increment_t   read_auto_increment;
                  # CCURPMFC_SDRAM_READ_AUTO_INCREMENT_DISABLE
                  # CCURPMFC_SDRAM_READ_AUTO_INCREMENT_ENABLE
                  # CCURPMFC_SDRAM_READ_AUTO_INCREMENT_DO_NOT_CHANGE
               _ccurpmfc_sdram_write_auto_increment_t  write_auto_increment;
                  # CCURPMFC_SDRAM_WRITE_AUTO_INCREMENT_DISABLE
                  # CCURPMFC_SDRAM_WRITE_AUTO_INCREMENT_ENABLE
                  # CCURPMFC_SDRAM_READ_AUTO_INCREMENT_DO_NOT_CHANGE
               _ccurpmfc_sdram_read_timeout_t          read_timeout;
                  # CCURPMFC_SDRAM_READ_TIMEOUT_DID_NOT_OCCUR
                  # CCURPMFC_SDRAM_READ_TIMEOUT_OCCURRED
                  # CCURPMFC_SDRAM_READ_TIMEOUT_DO_NOT_CHANGE
               _ccurpmfc_sdram_calibration_fail_t      calibration_failed;
                  # CCURPMFC_SDRAM_CALIBRATION_FAIL_RESET
                  # CCURPMFC_SDRAM_CALIBRATION_FAIL_SET
                  # CCURPMFC_SDRAM_CALIBRATION_FAIL_DO_NOT_CHANGE
               _ccurpmfc_sdram_calibration_pass_t      calibration_passed;
                  # CCURPMFC_SDRAM_CALIBRATION_PASS_RESET
                  # CCURPMFC_SDRAM_CALIBRATION_PASS_SET
                  # CCURPMFC_SDRAM_CALIBRATION_PASS_DO_NOT_CHANGE
               _ccurpmfc_sdram_initilization_done_t    initialization_done;
                  # CCURPMFC_SDRAM_INITIALIZATION_NOT_COMPLETE
                  # CCURPMFC_SDRAM_INITIALIZATION_COMPLETE
                  # CCURPMFC_SDRAM_INITIALIZATION_DO_NOT_CHANGE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                 (successful)
               # CCURPMFC_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                 (device not open)
               # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION          (local region not present)
               # CCURPMFC_LIB_SDRAM_IS_NOT_ACTIVE      (SDRAM is not active)
```

```
 ***************************************************************************/
```

2.2.204 ccurPMFC_SDRAM_Write()

This call provided the user the ability to write to any portion of the SDRAM. Its range is from 1 to 0x10000000 (256Mwords). Offset to this routine is only set if it is 0 or greater. Maximum offset is 0x0FFFFFFF. If offset is negative, then the write commences from the last written location.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_SDRAM_Write (void         *Handle,
                         u_int32_t    *Buf,
                         int          Offset,
                         u_int32_t    Size,
                         u_int32_t    *Words_written)

   Description: Perform a SDRAM write operation.

   Input:   void                        *Handle        (Handle pointer)
            u_int32_t                   *Buf           (pointer to buffer)
            int                         Offset         (word offset into SDRAM)
            u_int32_t                   Size           (size of buffer in words)
   Output:  u_int32_t                   *Words_written (words written)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURPMFC_LIB_SDRAM_IS_NOT_ACTIVE     (SDRAM is not active)
 ***************************************************************************/
```

2.2.205 ccurPMFC_Set_Board_CSR()

This call is used to set the board control register.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Set_Board_CSR (void                    *Handle,
                           ccurpmfc_board_csr_t    *bcsr)

   Description: Set Board Control and Status information

   Input:   void                        *Handle         (Handle pointer)
            ccurpmfc_board_csr_t                *bcsr   (pointer to board csr)
               _ccurpmfc_bcsr_identify_board_t  identify_board
                  # CCURPMFC_BCSR_IDENTIFY_BOARD_DISABLE
                  # CCURPMFC_BCSR_IDENTIFY_BOARD_ENABLE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN                (device not open)
               # CCURPMFC_LIB_INVALID_ARG             (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION         (local region not present)
 ***************************************************************************/
```

2.2.206 ccurPMFC_Set_Calibration_CSR()

This call sets the current calibration control and status register.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Set_Calibration_CSR (void                      *Handle,
```

```
                                ccurpmfc_calibration_csr_t *CalCSR)

   Description: Set Calibration Control and Status Register

   Input:   void                          *Handle     (Handle pointer)
            ccurpmfc_calibration_csr_t    *CalCSR     (pointer to calibration CSR)
              _ccurpmfc_calbus_control_t BusControl  (bus control)
                 # CCURPMFC_CB_GROUND
                 # CCURPMFC_CB_POSITIVE_REFERENCE
                 # CCURPMFC_CB_NEGATIVE_REFERENCE
                 # CCURPMFC_CB_BUS_OPEN
                 # CCURPMFC_CB_2_5V_REFERENCE
                 # CCURPMFC_CB_5V_REFERENCE
                 # CCURPMFC_CB_DAC_CHANNEL_0
                 # CCURPMFC_CB_DAC_CHANNEL_1
                 # CCURPMFC_CB_DAC_CHANNEL_2
                 # CCURPMFC_CB_DAC_CHANNEL_3
                 # CCURPMFC_CB_DAC_CHANNEL_4
                 # CCURPMFC_CB_DAC_CHANNEL_5
                 # CCURPMFC_CB_DAC_CHANNEL_6
                 # CCURPMFC_CB_DAC_CHANNEL_7
                 # CCURPMFC_CB_DAC_CHANNEL_8
                 # CCURPMFC_CB_DAC_CHANNEL_9
                 # CCURPMFC_CB_DAC_CHANNEL_10
                 # CCURPMFC_CB_DAC_CHANNEL_11
                 # CCURPMFC_CB_DAC_CHANNEL_12
                 # CCURPMFC_CB_DAC_CHANNEL_13
                 # CCURPMFC_CB_DAC_CHANNEL_14
                 # CCURPMFC_CB_DAC_CHANNEL_15
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
 ***************************************************************************/
```

2.2.207 ccurPMFC_Set_Interrupt_Status()

This call sets/clears the various interrupts. In the case of DIO COS interrupts, the change-of-status for all the channels in a particular channel group (Group0=Ch00..31, Group1=Ch32..63, Group2=Ch64..95) are cleared in the DIO COS status registers in order to clear the corresponding DIO COS interrupt status.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Set_Interrupt_Status (void                 *Handle,
                                  ccurpmfc_interrupt_t *intr)

   Description: Set Interrupt Status

   Input:   void                     *Handle       (handle pointer)
            ccurpmfc_interrupt_t     *intr         (pointer to interrupt status)
               _ccurpmfc_intsta_dio_cos_t dio_cos_group2_int
                  # CCURPMFC_INT_DIO_COS_NONE
                  # CCURPMFC_INT_DIO_COS_RESET
                  # CCURPMFC_INT_DIO_COS_DO_NOT_CHANGE
               _ccurpmfc_intsta_dio_cos_t dio_cos_group1_int
                  # CCURPMFC_INT_DIO_COS_NONE
                  # CCURPMFC_INT_DIO_COS_RESET
                  # CCURPMFC_INT_DIO_COS_DO_NOT_CHANGE
               _ccurpmfc_intsta_dio_cos_t dio_cos_group0_int
```

```
                  # CCURPMFC_INT_DIO_COS_NONE
                  # CCURPMFC_INT_DIO_COS_RESET
                  # CCURPMFC_INT_DIO_COS_DO_NOT_CHANGE
               _ccurpmfc_intsta_dac_t
                  # CCURPMFC_INT_DAC_FIFO_THRESHOLD_NONE
                  # CCURPMFC_INT_DAC_FIFO_THRESHOLD_RESET
                  # CCURPMFC_INT_DAC_FIFO_THRESHOLD_DO_NOT_CHANGE
               _ccurpmfc_intsta_adc_t
                  # CCURPMFC_INT_ADC_FIFO_THRESHOLD_NONE
                  # CCURPMFC_INT_ADC_FIFO_THRESHOLD_RESET
                  # CCURPMFC_INT_ADC_FIFO_THRESHOLD_DO_NOT_CHANGE
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (device not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (local region error)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)

 ****************************************************************************/
```

2.2.208 ccurPMFC_Set_Interrupt_Timeout_Seconds()

This call sets the read *timeout* maintained by the driver. It allows the user to change the default time out from 30 seconds to a user specified value. It is the time that the read call will wait before it times out. The call could time out if the DMA fails to complete. The device should have been opened in the blocking mode *(O_NONBLOCK not set)* for reads to wait for the operation to complete.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Set_Interrupt_Timeout_Seconds (void    *Handle,
                                           int     timeout_secs)

   Description: Set Interrupt Timeout Seconds

   Input:   void         *Handle           (Handle pointer)
            int          timeout_secs      (interrupt tout secs)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR      (successful)
               # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN      (device not open)
               # CCURPMFC_LIB_INVALID_ARG   (invalid argument)
 ****************************************************************************/
```

2.2.209 ccurPMFC_Set_TestBus_Control()

This call is provided for internal use in testing the hardware.

```
/****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Set_TestBus_Control (void                        *Handle,
                                 _ccurpmfc_testbus_control_t test_control)

   Description: Set the value of the Test Bus control information

   Input:   void                    *Handle       (handle pointer)
   Output:  _ccurpmfc_testbus_control_t
                                    test_control  (control select)
               # CCURPMFC_TBUS_CONTROL_OPEN
               # CCURPMFC_TBUS_CONTROL_CAL_BUS
```

```
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (local region error)
               # CCURPMFC_LIB_BAD_HANDLE          (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN            (device not open)
 ************************************************************************/
```

2.2.210 ccurPMFC_Set_Value()

This call allows the advanced user to set the writable board registers. The actual data written will depend on the command register information that is requested. Refer to the hardware manual for more information on what can be written to.

Normally, users should not be changing these registers as it will bypass the API integrity and could result in an unpredictable outcome.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Set_Value (void                *Handle,
                       CCURPMFC_CONTROL    cmd,
                       void                *value)

   Description: Set the value of the specified board register.

   Input:   void             *Handle        (Handle pointer)
            CCURPMFC_CONTROL cmd            (register definition)
               -- structure in ccurpmfc_lib.h
            void             *value         (pointer to value to be set)
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR      (successful)
               # CCURPMFC_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN      (device not open)
               # CCURPMFC_LIB_INVALID_ARG   (invalid argument)
 ************************************************************************/
```

2.2.211 ccurPMFC_SPROM_Read()

This is a basic call to read short word entries from the serial prom. The user specifies a word offset within the serial prom and a word count, and the call returns the data read in a pointer to short words.

```
/************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_SPROM_Read(void                *Handle,
                       ccurpmfc_sprom_rw_t *spr)

   Description: Read Serial Prom for specified number of words

   Input:       void                   *Handle    (handle pointer)
                ccurpmfc_sprom_rw_t    *spr       (pointer to struct)
                   u_short word_offset
                   u_short num_words
   Output:      ccurpmfc_sprom_rw_t    *spr       (pointer to struct)
                   u_short *data_ptr
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (error)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
               # CCURPMFC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
************************************************************************/
```

2.2.212 ccurPMFC_SPROM_Read_Item()

This call is used to read well defined sections in the serial prom. The user supplies the serial prom section that needs to be read and the data is returned in a section specific structure.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_SPROM_Read_Item(void                      *Handle,
                            _ccurpmfc_sprom_access_t item,
                            void                      *item_ptr)

   Description: Read Serial Prom for specified item

   Input:       void                   *Handle    (handle pointer)
                _ccurpmfc_sprom_access_t item     (select item)
                   # CCURPMFC_SPROM_HEADER
   Output:      void                   *item_ptr  (pointer to item struct)
                   *ccurpmfc_sprom_header_t sprom_header
                      u_int32_t        board_serial_number
                      u_short          sprom_revision
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (error)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
               # CCURPMFC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
*************************************************************************/
```

2.2.213 ccurPMFC_SPROM_Write()

This is a basic call to write short word entries to the serial prom. The user specifies a word offset within the serial prom and a word count, and the call writes the data pointed to by the *spw* pointer, in short words.

Prior to using this call, the user will need to issue the *ccurPMFC_SPROM_Write_Override()* to allow writing to the serial prom.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_SPROM_Write(void                *Handle,
                        ccurpmfc_sprom_rw_t *spw)

   Description: Write data to Serial Prom for specified number of words

   Input:       void                   *Handle    (handle pointer)
                ccurpmfc_sprom_rw_t    *spw       (pointer to struct)
                   u_short word_offset
                   u_short num_words
                   u_short *data_ptr
   Output:      none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (error)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
               # CCURPMFC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
*************************************************************************/
```

2.2.214 ccurPMFC_SPROM_Write_Item()

This call is used to write well defined sections in the serial prom. The user supplies the serial prom section that needs to be written and the data points to the section specific structure. This call should normally not be used by the user.

Prior to using this call, the user will need to issue the *ccurPMFC_SPROM_Write_Override()* to allowing writing to the serial prom.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_SPROM_Write_Item(void                      *Handle,
                             _ccurpmfc_sprom_access_t item,
                             void                      *item_ptr)

   Description: Write Serial Prom with specified item

   Input:       void                   *Handle    (handle pointer)
                _ccurpmfc_sprom_access_t item     (select item)
                   # CCURPMFC_SPROM_HEADER
                void                   *item_ptr  (pointer to item struct)
                   *ccurpmfc_sprom_header_t sprom_header
                      u_int32_t        board_serial_number
                      u_short          sprom_revision
   Output:      none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR            (successful)
               # CCURPMFC_LIB_NO_LOCAL_REGION     (error)
               # CCURPMFC_LIB_INVALID_ARG         (invalid argument)
               # CCURPMFC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
               # CCURPMFC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
 *************************************************************************/
```

2.2.215 ccurPMFC_SPROM_Write_Override()

The serial prom is non-volatile and its information is preserved during a power cycle. It contains useful information and settings that the customer could lose if they were to inadvertently overwrite. For this reason, all calls that write to the serial proms will fail with a write protect error, unless this write protect override API is invoked prior to writing to the serial proms. Once the Write Override is enabled, it will stay in effect until the user closes the device or re-issues this call to disable writes to the serial prom.
The calls that will fail unless the write protect is disabled are:

- ccurPMFC_Write_Serial_Prom()
- ccurPMFC_Write_Serial_Prom_Item()

When *action* is set to *CCURPMFC_TRUE*, the serial prom write protecting is disabled, otherwise, it is enabled.

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_SPROM_Write_Override (void *Handle,
                                  int action)

   Description: Set Serial Prom Write Override

   Input:       void            *Handle             (handle pointer)
                int             action              (override action)
                   # CCURPMFC_TRUE
                   # CCURPMFC_FALSE
   Output:      none
```

```
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR             (successful)
               # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN             (device not open)
               # CCURPMFC_LIB_INVALID_ARG          (invalid argument)
               # CCURPMFC_LIB_NO_LOCAL_REGION      (local region not present)
 ***************************************************************************/
```

2.2.216 ccurPMFC_Transfer_Data()

This is the main call that the user can use to transfer data from physical memory that the user has previously allocated to a region in the local register, and vice-versa. The operation can be performed via DMA or programmed I/O mode. In the case of DMA mode, the user can select whether interrupts are to be used to wait for DMA to complete instead of polling. User can also specify which DMA engine to use during this operation.

If the board supports modular scatter-gather DMA, then the user can specify that instead of the basic DMA engine. In this case, the user needs to first call the *ccurPMFC_MsgDma_Configure_Single()* with the *NULL* argument to setup descriptor ID-1 for scatter-gather DMA operation.

When scatter-gather DMA is selected, the *DmaEngineNo* argument is ignored and the *IoControl* argument must be set to *CCURPMFC_DMA_CONTROL_INCREMENT.*

There are certain limitations to modular scatter-gather feature:

1. Scatter-gather DMA is only supported in certain cards
2. Reads from Avalon memory below DiagRam location are not allowed for MIOC FPGA cards.
3. Invalid memory address supplied could result in the scatter-gather IP to lock up and the only way to recover will be to reload the driver or reboot the system.
4. Read and write addresses must be at a minimum full-word aligned and for maximum performance, it is recommended to be quad-word aligned.
5. Lengths are in bytes and must be at a minimum a multiple of a full-word and for maximum performance, it is recommended to be quad-word multiple.
6. Scatter-gather chaining cannot be performed with this call.

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Transfer_Data(void                              *Handle,
                          volatile void                     *PciDmaMemory,
                          volatile void                     *AvalonMem,
                          uint                              TransferSize,
                          _ccurpmfc_direction_t             XferDirection,
                          _ccurpmfc_library_rw_mode_t       LibMode,
                          ccurpmfc_dma_engine_t             DMAEngineNo,
                          ccurpmfc_bool                     UseInterrupts,
                          int                               IoControl)

   Description: Routine to transfer data from PCI memory to Avalon memory
                or vice-versa

   Input:   void                      *Handle         (Handle pointer)
            volatile void             *PciDmaMemory   (pointer to virtual memory)
            volatile void             *AvalonMem      (pointer to virtual Avalon
                                                       memory)
            uint                      TransferSize    (size of transfer in bytes)
            _ccurpmfc_direction_t     XferDirection   (direction of transfer)
               # CCURPMFC_AVALON_2_PCIMEM
               # CCURPMFC_PCIMEM_2_AVALON
            _ccurpmfc_library_rw_mode_t  LibMode      (Lib transfer mode)
               # CCURPMFC_LIBRARY_PIO_MODE
```

```
            # CCURPMFC_LIBRARY_DMA_MODE
            # CCURPMFC_LIBRARY_MSGDMA_MODE
         ccurpmfc_dma_engine_t     DMAEngineNo    (select DMA engine)
            # CCURPMFC_DMA0
            # CCURPMFC_DMA1
            # CCURPMFC_NONE
         ccurpmfc_bool             UseInterrupts  (enable interrupts)
            # CCURPMFC_TRUE
            # CCURPMFC_FALSE
         int                       IoControl      (DMA or PIO control flags)
            # CCURPMFC_DMA_CONTROL_RCON           (DMA: read constant)
            # CCURPMFC_DMA_CONTROL_WCON           (DMA: write constant)
            # CCURPMFC_DMA_CONTROL_INCREMENT      (DMA: increment)
            # CCURPMFC_PIO_CONTROL_RCON           (PIO: read constant)
            # CCURPMFC_PIO_CONTROL_WCON           (PIO: write constant)
            # CCURPMFC_PIO_CONTROL_INCREMENT      (PIO: increment)

   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
            # CCURPMFC_LIB_NO_ERROR               (no error)
            # CCURPMFC_LIB_BAD_HANDLE             (no/bad handler supplied)
            # CCURPMFC_LIB_NOT_OPEN               (library not open)
            # CCURPMFC_LIB_INVALID_ARG            (invalid argument)
            # CCURPMFC_LIB_IOCTL_FAILED           (driver ioctl call failed)
            # CCURPMFC_LIB_MSG_DMA_READS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                                 (MSG DMA Reads not allowed
                                                  for selected address)
 ***************************************************************************/
```

2.2.217 ccurPMFC_Update_Clock_Generator_Divider()

Update the selected clock generator divider so that its changes take affect. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```
/***************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Update_Clock_Generator_Divider (void                     *Handle,
                           _ccurpmfc_clock_generator_divider_t    WhichDivider)

   Description: Update Clock Generator Divider

   Input:   void                                 *Handle         (Handle pointer)
            _ccurpmfc_clock_generator_divider_t  WhichDivider    (select divider)
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_M
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N0
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N1
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N2
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N3
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_N_ALL
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P0
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P1
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P2
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_PFB
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_P_ALL
               # CCURPMFC_CLOCK_GENERATOR_DIVIDER_PXAXB
   Output:  none
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR                (successful)
               # CCURPMFC_LIB_BAD_HANDLE              (no/bad handler supplied)
```

```
            # CCURPMFC_LIB_NOT_OPEN                 (library not open)
            # CCURPMFC_LIB_NO_LOCAL_REGION          (local region error)
            # CCURPMFC_LIB_INVALID_ARG              (invalid argument)
            # CCURPMFC_LIB_CLOCK_IS_NOT_ACTIVE      (Clock is not active)
*************************************************************************/
```

2.2.218 ccurPMFC_UserProcess_Command()

The user can control the execution of the created User Process with the help of this call. *(This is an experimental API for debugging and testing).*

```
/*************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_UserProcess_Command(void                  *Handle,
                                void                  *UFuncHandle,
                                ccurpmfc_uf_action_t  Action)

   Description: Command User process

   Input:   void                  *UFuncHandle  (User Process Handle pointer)
            ccurpmfc_uf_action_t  Action        (command action)
               # CCURPMFC_UF_ACTION_STOP
               # CCURPMFC_UF_ACTION_RUN
               # CCURPMFC_UF_ACTION_TERMINATE
   Output:  none
   Return:  none
 *************************************************************************/
```

2.2.219 ccurPMFC_VoltsToData()

This call returns to the user the raw converted value for the requested voltage in the specified format. Voltage supplied must be within the input range of the selected board type. If the voltage is out of range, the call sets the voltage to the appropriate limit value.

```
/*************************************************************************
   uint
   ccurPMFC_VoltsToData (double                  volts,
                         ccurpmfc_volt_convert_t *conv)

   Description: Convert Volts to data

   Input:   double                       volts        (volts to convert)
            ccurpmfc_volt_convert_t      *conv     (pointer to conversion struct)
              double                     VoltageRange (maximum voltage range)
              _ccurpmfc_csr_dataformat_t Format       (format)
                 # CCURPMFC_OFFSET_BINARY
                 # CCURPMFC_TWOS_COMPLEMENT
              ccurpmfc_bool              BiPolar      (bi-polar)
                 # CCURPMFC_TRUE
                 # CCURPMFC_FALSE
              int                   ResolutionBits (Number of resolution bits)
   Output:  none
   Return:  uint                    data           (returned data)
 *************************************************************************/
```

2.2.220 ccurPMFC_Wait_For_Interrupt()

This call is made available to advanced users to bypass the API and perform their own data collection. The user can wait for a DMA complete interrupt. If a time out value greater than zero is specified, the call will time out after the specified seconds, otherwise it will not time out.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Wait_For_Interrupt (void                    *Handle,
                                ccurpmfc_driver_int_t  *drv_int)

   Description: Wait For Interrupt

   Input:   void                    *Handle         (Handle pointer)
            ccurpmfc_driver_int_t   *drv_int        (pointer to drv_int struct)
               uint                  WakeupInterruptMask
                  # CCURPMFC_DMA0_INTMASK
                  # CCURPMFC_DMA1_INTMASK
                  # CCURPMFC_MSGDMA_INTMASK
                  # CCURPMFC_ADC_FIFO_INTMASK
                  # CCURPMFC_DAC_FIFO_INTMASK
               int                   timeout_seconds
   Output:  ccurpmfc_driver_int_t   *drv_int       (pointer to drv_int struct)
               long long unsigned    count
               long long unsigned    dma_count[CCURPMFC_DMA_MAX_ENGINES]
               long long unsigned    MsgDma_count
               uint                  InterruptsOccurredMask
               uint                  WakeupInterruptMask
               int                   DmaControl     (DMA control flags)
                  # CCURPMFC_DMA_CONTROL_RCON      (read constant)
                  # CCURPMFC_DMA_CONTROL_WCON      (write constant)
                  # CCURPMFC_DMA_CONTROL_INCREMENT (increment)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURPMFC_LIB_INVALID_ARG       (invalid argument)
 *****************************************************************************/
```

2.2.221 ccurPMFC_Write()

This call is currently not supported.

```
/*****************************************************************************
   _ccurpmfc_lib_error_number_t
   ccurPMFC_Write (void    *Handle,
                   void    *buf,
                   int     size,
                   int     *bytes_written,
                   int     *error)

   Description: Perform a write operation.

   Input:   void     *Handle                   (Handle pointer)
            int      size                      (number of bytes to write)
   Output:  void     *buf                      (pointer to buffer)
            int      *bytes_written            (bytes written)
            int      *error                    (returned errno)
   Return:  _ccurpmfc_lib_error_number_t
               # CCURPMFC_LIB_NO_ERROR          (successful)
               # CCURPMFC_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURPMFC_LIB_NOT_OPEN          (device not open)
               # CCURPMFC_LIB_IO_ERROR          (write failed)
               # CCURPMFC_LIB_NOT_IMPLEMENTED   (call not implemented)
 *****************************************************************************/
```

3. Test Programs

This driver and API are accompanied with an extensive set of test examples. Examples under the *Direct Driver Access* do not use the API, while those under *Application Program Interface Access* use the API.

3.1 Direct Driver Access Example Tests

These set of tests are located in the *.../test* directory and do not use the API. They communicate directly with the driver. Users should be extremely familiar with both the driver and the hardware registers if they wish to communicate directly with the hardware.

3.1.1 ccurpmfc_disp

Useful program to display the local board registers. This program uses the *curses* library.

```
Usage: ./ccurpmfc_disp [-b BoardNo] [-d Delay] [-l LoopCnt] [-o Offset] [-s Size]
 -b BoardNo  (Board number -- default is 0)
 -d Delay    (Delay between screen refresh -- default is 0)
 -l LoopCnt  (Loop count -- default is 0)
 -o Offset   (Hex offset to read from -- default is 0x0)
 -s Size     (Number of bytes to read -- default is 0x400)
```

Example display:

./ccurpmfc_disp

```
 Board Number    [-b]: 0
 Delay           [-d]: 0 milli-seconds
 Loop Count      [-l]: ***Forever***
 Offset          [-o]: 0x00000000
 Size            [-s]: 1024 (bytes)

 ScanCount =    59758

         00       04       08       0C       10       14       18       1C
         ======== ======== ======== ======== ======== ======== ======== ========
000000   92900101 06192019 00040000 00000000 00000000 00000000 00000000 00000000
000020   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000040   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000060   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000080   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000a0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000c0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000e0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000100   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000120   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000140   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000160   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000180   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001a0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001c0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001e0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000200   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000220   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000240   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000260   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000280   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0002a0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0002c0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0002e0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000300   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000320   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000340   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000360   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000380   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
0003a0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0003c0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0003e0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

3.1.2 ccurpmfc_dump

This test is for debugging purpose. It dumps all the hardware registers.

Usage: ccurpmfc_dump [-b board]
 -b board: board number -- default board is 0

Example display:

./ccurpmfc_dump

```
Device Name: /dev/ccurpmfc0

  LOCAL REGION: Physical Addr=0xbd300000 Size=131072 (0x00020000)
 CONFIG REGION: Physical Addr=0xbd320000 Size=32768 (0x00008000)

     LOCAL: Register 0x7ffff7fd7000 Offset=0x0 Size=0x00020000
    CONFIG: Register 0x7ffff7fcf000 Offset=0x0 Size=0x00008000
    LIBPTR: Register 0x7ffff7fcd000 Offset=0x0 Size=0x00001078

======= LOCAL BOARD REGISTERS =========
LBR: @0x000000 --> 0x92900101
LBR: @0x000004 --> 0x06192019
LBR: @0x000008 --> 0x00040000
LBR: @0x00000c --> 0x00000000
LBR: @0x000010 --> 0x00000000
LBR: @0x000014 --> 0x00000000
LBR: @0x000018 --> 0x00000000
LBR: @0x00001c --> 0x00000000
LBR: @0x000020 --> 0x00000000
LBR: @0x000024 --> 0x00000000
LBR: @0x000028 --> 0x00000000
LBR: @0x00002c --> 0x00000000
LBR: @0x000030 --> 0x00000000
.
.
.
LBR: @0x01ffcc --> 0x00000000
LBR: @0x01ffd0 --> 0x00000000
LBR: @0x01ffd4 --> 0x00000000
LBR: @0x01ffd8 --> 0x00000000
LBR: @0x01ffdc --> 0x00000000
LBR: @0x01ffe0 --> 0x00000000
LBR: @0x01ffe4 --> 0x00000000
LBR: @0x01ffe8 --> 0x00000000
LBR: @0x01ffec --> 0x00000000
LBR: @0x01fff0 --> 0x00000000
LBR: @0x01fff4 --> 0x00000000
LBR: @0x01fff8 --> 0x00000000
LBR: @0x01fffc --> 0x00000000

======= LOCAL CONFIG REGISTERS =========
#### CONFIG REGS (PCIeLinkPartnerRegs) ####
```

```
LCR: @0x0000 --> 0x00000000
LCR: @0x0004 --> 0x00000000
LCR: @0x0008 --> 0x00000000
LCR: @0x000c --> 0x00000000
LCR: @0x0010 --> 0x00000000
LCR: @0x0014 --> 0x00000000
LCR: @0x0018 --> 0x00000000
LCR: @0x001c --> 0x00000000
LCR: @0x0020 --> 0x00000000
LCR: @0x0024 --> 0x00000000
LCR: @0x0028 --> 0x00000000
LCR: @0x002c --> 0x00000000
LCR: @0x0030 --> 0x00000000
.
.
.
LCR: @0x0fc0 --> 0x00000000
LCR: @0x0fc4 --> 0x00000000
LCR: @0x0fc8 --> 0x00000000
LCR: @0x0fcc --> 0x00000000
LCR: @0x0fd0 --> 0x00000000
LCR: @0x0fd4 --> 0x00000000
LCR: @0x0fd8 --> 0x00000000
LCR: @0x0fdc --> 0x00000000
LCR: @0x0fe0 --> 0x00000000
LCR: @0x0fe4 --> 0x00000000
LCR: @0x0fe8 --> 0x00000000
LCR: @0x0fec --> 0x00000000
LCR: @0x0ff0 --> 0x00000000
LCR: @0x0ff4 --> 0x00000000
LCR: @0x0ff8 --> 0x00000000
LCR: @0x0ffc --> 0x00000000

#### CONFIG REGS (AvalonMM_2_PCIeAddrTrans) ####
LCR: @0x1000 --> 0x00000000
LCR: @0x1004 --> 0x00000000
LCR: @0x1008 --> 0x00000000
LCR: @0x100c --> 0x00000000
LCR: @0x1010 --> 0x00000000
LCR: @0x1014 --> 0x00000000
LCR: @0x1018 --> 0x00000000
LCR: @0x101c --> 0x00000000
LCR: @0x1020 --> 0x00000000
LCR: @0x1024 --> 0x00000000
LCR: @0x1028 --> 0x00000000
LCR: @0x102c --> 0x00000000
LCR: @0x1030 --> 0x00000000
.
.
.
LCR: @0x1fb0 --> 0x00000000
LCR: @0x1fb4 --> 0x00000000
LCR: @0x1fb8 --> 0x00000000
LCR: @0x1fbc --> 0x00000000
LCR: @0x1fc0 --> 0x00000000
```

```
LCR: @0x1fc4 --> 0x00000000
LCR: @0x1fc8 --> 0x00000000
LCR: @0x1fcc --> 0x00000000
LCR: @0x1fd0 --> 0x00000000
LCR: @0x1fd4 --> 0x00000000
LCR: @0x1fd8 --> 0x00000000
LCR: @0x1fdc --> 0x00000000
LCR: @0x1fe0 --> 0x00000000
LCR: @0x1fe4 --> 0x00000000
LCR: @0x1fe8 --> 0x00000000
LCR: @0x1fec --> 0x00000000
LCR: @0x1ff0 --> 0x00000000
LCR: @0x1ff4 --> 0x00000000
LCR: @0x1ff8 --> 0x00000000
LCR: @0x1ffc --> 0x00000000

#### CONFIG REGS (DMA Control Table) ####
LCR: @0x4000 --> 0x00000011
LCR: @0x4004 --> 0x0000c000
LCR: @0x4008 --> 0x00903400
LCR: @0x400c --> 0x00000000
LCR: @0x4010 --> 0x00000000
LCR: @0x4014 --> 0x00000000
LCR: @0x4018 --> 0x00000000
LCR: @0x401c --> 0x00000000
LCR: @0x4020 --> 0x00000000
LCR: @0x4024 --> 0x00000000
LCR: @0x4028 --> 0x00000000
LCR: @0x402c --> 0x00000000
LCR: @0x4030 --> 0x00000000
LCR: @0x4034 --> 0x00000000
LCR: @0x4038 --> 0x00000000
LCR: @0x403c --> 0x00000000

======= PCI CONFIG REG ADDR MAPPING =========
PCR: @0x0000 --> 0x92901542
PCR: @0x0004 --> 0x00100406
PCR: @0x0008 --> 0x08800001
PCR: @0x000c --> 0x00000010
PCR: @0x0010 --> 0xbd320000
PCR: @0x0014 --> 0x00000000
PCR: @0x0018 --> 0xbd300000
PCR: @0x001c --> 0x00000000
PCR: @0x0020 --> 0x00000000
PCR: @0x0024 --> 0x00000000
PCR: @0x0028 --> 0x00000000
PCR: @0x002c --> 0x01001542
PCR: @0x0030 --> 0x00000000
PCR: @0x0034 --> 0x00000050
PCR: @0x0038 --> 0x00000000
PCR: @0x003c --> 0x0000010b
PCR: @0x0040 --> 0x00000000
PCR: @0x0044 --> 0x02006160
PCR: @0x0048 --> 0x00000000
PCR: @0x004c --> 0x00000000
```

```
PCR: @0x0050 --> 0x00857805
PCR: @0x0054 --> 0xfeeff00c
PCR: @0x0058 --> 0x00000000
PCR: @0x005c --> 0x00004185
PCR: @0x0060 --> 0x00000000
PCR: @0x0064 --> 0x00000000
PCR: @0x0068 --> 0x00007811
PCR: @0x006c --> 0x00000000
PCR: @0x0070 --> 0x00000000
PCR: @0x0074 --> 0x00000000
PCR: @0x0078 --> 0x00038001
PCR: @0x007c --> 0x00000000
PCR: @0x0080 --> 0x00020010
PCR: @0x0084 --> 0x00648001
PCR: @0x0088 --> 0x00002830
PCR: @0x008c --> 0x01406441
PCR: @0x0090 --> 0x10410040
PCR: @0x0094 --> 0x00000000
PCR: @0x0098 --> 0x00000000
PCR: @0x009c --> 0x00000000
PCR: @0x00a0 --> 0x00000000
PCR: @0x00a4 --> 0x0000001f
PCR: @0x00a8 --> 0x0000000d
PCR: @0x00ac --> 0x00000000
PCR: @0x00b0 --> 0x00010001
PCR: @0x00b4 --> 0x00000000
PCR: @0x00b8 --> 0x00000000
PCR: @0x00bc --> 0x00000000
PCR: @0x00c0 --> 0x00000000
PCR: @0x00c4 --> 0x00000000
PCR: @0x00c8 --> 0x00000000
PCR: @0x00cc --> 0x00000000
PCR: @0x00d0 --> 0x00000000
PCR: @0x00d4 --> 0x00000000
PCR: @0x00d8 --> 0x00000000
PCR: @0x00dc --> 0x00000000
PCR: @0x00e0 --> 0x00000000
PCR: @0x00e4 --> 0x00000000
PCR: @0x00e8 --> 0x00000000
PCR: @0x00ec --> 0x00000000
PCR: @0x00f0 --> 0x00000000
PCR: @0x00f4 --> 0x00000000
PCR: @0x00f8 --> 0x00000000
PCR: @0x00fc --> 0x00000000
```

3.1.3 ccurpmfc_rdreg

This is a simple program that returns the local register value for a given offset.

```
Usage: ./ccurpmfc_rdreg [-b Board] [-C] [-f] [-o Offset] [-s Size]
 -b Board   : Board number -- default board is 0
 -C         : Select Config Registers instead of Local Registers
 -f         : Fast Memory Reads
 -o Offset  : Hex offset to read from -- default offset is 0x0
 -s Size    : Number of bytes to read in decimal -- default size is 0x4
```

Example display:
./ccurpmfc_rdreg –s64

```
Device Name: /dev/ccurpmfc0

  LOCAL REGION: Physical Addr=0xc4900000 Size=65536 (0x00010000)
 CONFIG REGION: Physical Addr=0xc4910000 Size=32768 (0x00008000)

         LOCAL: Register 0x7ffff7fe7000 Offset=0x0 Size=0x00010000
        CONFIG: Register 0x7ffff7fdf000 Offset=0x0 Size=0x00008000
        LIBPTR: Register 0x7ffff7fdd000 Offset=0x0 Size=0x00001070

#### LOCAL REGS #### (length=64)
+LCL+        0   92900101  06192019  00040000  00000000 *.....% ..U......*
+LCL+     0x10   00000000  00000000  00000000  00000000 *................*
+LCL+     0x20   00000000  00000000  00000000  00000000 *................*
+LCL+     0x30   00000000  00000000  00000000  00000000 *................*
  18.784us (   3.41 MB/s)
```

./ccurpmfc_rdreg -C -o4020 -s20

```
Device Name: /dev/ccurpmfc0

  LOCAL REGION: Physical Addr=0xc4900000 Size=65536 (0x00010000)
 CONFIG REGION: Physical Addr=0xc4910000 Size=32768 (0x00008000)

         LOCAL: Register 0x7ffff7fe7000 Offset=0x0 Size=0x00010000
        CONFIG: Register 0x7ffff7fdf000 Offset=0x0 Size=0x00008000
        LIBPTR: Register 0x7ffff7fdd000 Offset=0x0 Size=0x00001070

#### LOCAL REGS #### (length=20)
+LCL+   0x4020   00000000  00000000  00000000  00000000 *................*
+LCL+   0x4030   00000000
   5.262us (   3.80 MB/s)
```

3.1.4 ccurpmfc_reg

This call displays all the boards local and configuration registers.

```
Usage: ./ccurpmfc_reg [-b board]
 -b board: Board number -- default board is 0
```

Example display:

./ccurpmfc_reg

```
Device Name: /dev/ccurpmfc0

  LOCAL REGION: Physical Addr=0xc4900000 Size=65536 (0x00010000)
 CONFIG REGION: Physical Addr=0xc4910000 Size=32768 (0x00008000)

         LOCAL: Register 0x7ffff7fe7000 Offset=0x0 Size=0x00010000
        CONFIG: Register 0x7ffff7fdf000 Offset=0x0 Size=0x00008000
        LIBPTR: Register 0x7ffff7fdd000 Offset=0x0 Size=0x00001070
LOCAL Register 0x7ffff7fe7000 size=0x00010000

#### LOCAL REGS #### (length=65536)
+LCL+        0   92900101  06192019  00040000  00000000 *.....% ..U......*
+LCL+     0x10   00000000  00000000  00000000  00000000 *................*
+LCL+     0x20   00000000  00000000  00000000  00000000 *................*
+LCL+     0x30   00000000  00000000  00000000  00000000 *................*
```

```
+LCL+     0x40   00000000  00000000  00000000  00000000 *................*
+LCL+     0x50   00000000  00000000  00000000  00000000 *................*
+LCL+     0x60   00000000  00000000  00000000  00000000 *................*
+LCL+     0x70   00000000  00000000  00000000  00000000 *................*
+LCL+     0x80   00000000  00000000  00000000  00000000 *................*
+LCL+     0x90   00000000  00000000  00000000  00000000 *................*
+LCL+     0xa0   00000000  00000000  00000000  00000000 *................*
+LCL+     0xb0   00000000  00000000  00000000  00000000 *................*
+LCL+     0xc0   00000000  00000000  00000000  00000000 *................*
+LCL+     0xd0   00000000  00000000  00000000  00000000 *................*
+LCL+     0xe0   00000000  00000000  00000000  00000000 *................*
+LCL+     0xf0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x100   00000000  00000000  00000000  00000000 *................*
+LCL+    0x110   00000000  00000000  00000000  00000000 *................*
+LCL+    0x120   00000000  00000000  00000000  00000000 *................*
+LCL+    0x130   00000000  00000000  00000000  00000000 *................*
+LCL+    0x140   00000000  00000000  00000000  00000000 *................*
+LCL+    0x150   00000000  00000000  00000000  00000000 *................*
+LCL+    0x160   00000000  00000000  00000000  00000000 *................*
+LCL+    0x170   00000000  00000000  00000000  00000000 *................*
+LCL+    0x180   00000000  00000000  00000000  00000000 *................*
+LCL+    0x190   00000000  00000000  00000000  00000000 *................*
+LCL+    0x1a0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x1b0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x1c0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x1d0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x1e0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x1f0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x200   00000000  00000000  00000000  00000000 *................*
.
.
.
+LCL+   0xfed0   00000000  00000000  00000000  00000000 *................*
+LCL+   0xfee0   00000000  00000000  00000000  00000000 *................*
+LCL+   0xfef0   00000000  00000000  00000000  00000000 *................*
+LCL+   0xff00   00000000  00000000  00000000  00000000 *................*
+LCL+   0xff10   00000000  00000000  00000000  00000000 *................*
+LCL+   0xff20   00000000  00000000  00000000  00000000 *................*
+LCL+   0xff30   00000000  00000000  00000000  00000000 *................*
+LCL+   0xff40   00000000  00000000  00000000  00000000 *................*
+LCL+   0xff50   00000000  00000000  00000000  00000000 *................*
+LCL+   0xff60   00000000  00000000  00000000  00000000 *................*
+LCL+   0xff70   00000000  00000000  00000000  00000000 *................*
+LCL+   0xff80   00000000  00000000  00000000  00000000 *................*
+LCL+   0xff90   00000000  00000000  00000000  00000000 *................*
+LCL+   0xffa0   00000000  00000000  00000000  00000000 *................*
+LCL+   0xffb0   00000000  00000000  00000000  00000000 *................*
+LCL+   0xffc0   00000000  00000000  00000000  00000000 *................*
+LCL+   0xffd0   00000000  00000000  00000000  00000000 *................*
+LCL+   0xffe0   00000000  00000000  00000000  00000000 *................*
+LCL+   0xfff0   00000000  00000000  00000000  00000000 *................*

CONFIG Register 0x7ffff7fdf000 size=0x00008000

#### CONFIG REGS (PCIeLinkPartnerRegs) #### (length=4096)
+CFG+        0   00000000  00000000  00000000  00000000 *................*
+CFG+     0x10   00000000  00000000  00000000  00000000 *................*
+CFG+     0x20   00000000  00000000  00000000  00000000 *................*
+CFG+     0x30   00000000  00000000  00000000  00000000 *................*
+CFG+     0x40   00000000  00000000  00000000  00000000 *................*
+CFG+     0x50   00000000  00000000  00000000  00000000 *................*
```

```
+CFG+     0x60   00000004  00000004  00000008  00000008 *................*
+CFG+     0x70   00000000  00000000  00000000  00000000 *................*
+CFG+     0x80   00000000  00000000  00000000  00000000 *................*
+CFG+     0x90   00000000  00000000  00000000  00000000 *................*
+CFG+     0xa0   00000000  00000000  00000000  00000000 *................*
+CFG+     0xb0   00000000  00000000  00000000  00000000 *................*
+CFG+     0xc0   00000000  00000000  00000000  00000000 *................*
+CFG+     0xd0   00000000  00000000  00000000  00000000 *................*
+CFG+     0xe0   00000004  00000004  00000008  00000008 *................*
+CFG+     0xf0   00000000  00000000  00000000  00000000 *................*
+CFG+    0x100   00000000  00000000  00000000  00000000 *................*
+CFG+    0x110   00000000  00000000  00000000  00000000 *................*
+CFG+    0x120   00000000  00000000  00000000  00000000 *................*
+CFG+    0x130   00000000  00000000  00000000  00000000 *................*
+CFG+    0x140   00000000  00000000  00000000  00000000 *................*
+CFG+    0x150   00000000  00000000  00000000  00000000 *................*
+CFG+    0x160   00000000  00000000  00000000  00000000 *................*
+CFG+    0x170   00000000  00000000  00000000  00000000 *................*
+CFG+    0x180   00000000  00000000  00000000  00000000 *................*
+CFG+    0x190   00000000  00000000  00000000  00000000 *................*
+CFG+    0x1a0   00000000  00000000  00000000  00000000 *................*
+CFG+    0x1b0   00000000  00000000  00000000  00000000 *................*
+CFG+    0x1c0   00000000  00000000  00000000  00000000 *................*
.
.
.
+CFG+    0xf00   00000000  00000000  00000000  00000000 *................*
+CFG+    0xf10   00000000  00000000  00000000  00000000 *................*
+CFG+    0xf20   00000000  00000000  00000000  00000000 *................*
+CFG+    0xf30   00000000  00000000  00000000  00000000 *................*
+CFG+    0xf40   00000000  00000000  00000000  00000000 *................*
+CFG+    0xf50   00000000  00000000  00000000  00000000 *................*
+CFG+    0xf60   00000000  00000000  00000000  00000000 *................*
+CFG+    0xf70   00000000  00000000  00000000  00000000 *................*
+CFG+    0xf80   00000000  00000000  00000000  00000000 *................*
+CFG+    0xf90   00000000  00000000  00000000  00000000 *................*
+CFG+    0xfa0   00000000  00000000  00000000  00000000 *................*
+CFG+    0xfb0   00000000  00000000  00000000  00000000 *................*
+CFG+    0xfc0   00000000  00000000  00000000  00000000 *................*
+CFG+    0xfd0   00000000  00000000  00000000  00000000 *................*
+CFG+    0xfe0   00000000  00000000  00000000  00000000 *................*
+CFG+    0xff0   00000000  00000000  00000000  00000000 *................*

#### CONFIG REGS (AvalonMM_2_PCIeAddrTrans) #### (length=4096)
+CFG+   0x1000   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1010   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1020   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1030   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1040   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1050   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1060   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1070   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1080   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1090   00000000  00000000  00000000  00000000 *................*
+CFG+   0x10a0   00000000  00000000  00000000  00000000 *................*
+CFG+   0x10b0   00000000  00000000  00000000  00000000 *................*
+CFG+   0x10c0   00000000  00000000  00000000  00000000 *................*
+CFG+   0x10d0   00000000  00000000  00000000  00000000 *................*
.
.
.
```

```
+CFG+   0x1f50   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1f60   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1f70   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1f80   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1f90   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1fa0   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1fb0   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1fc0   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1fd0   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1fe0   00000000  00000000  00000000  00000000 *................*
+CFG+   0x1ff0   00000000  00000000  00000000  00000000 *................*


#### CONFIG REGS (DMA Control Table) #### (length=64)
+CFG+   0x4000   00000011  0000c000  00903400  00000000 *.........4.....*
+CFG+   0x4010   00000000  00000000  00000000  00000000 *................*
+CFG+   0x4020   00000000  00000000  00000000  00000000 *................*
+CFG+   0x4030   00000000  00000000  00000000  00000000 *................*


======= LOCAL REGISTERS =========
    BoardInfo                                                =0x92900101  @0x00000000
    FirmwareDate                                             =0x06192019  @0x00000004
    FirmwareRevision                                         =0x00040000  @0x00000008
    FirmwareTime                                             =0x00093204  @0x0000000c
    FirmwareFlavorCode                                       =0x47453031  @0x00000010
    NumberAdvancedIPCores                                    =0x00000001  @0x00000014
    BoardCSR                                                 =0x00000000  @0x00002000
    InterruptStatus                                          =0x00000000  @0x00002010
    SPI_CommandStatus                                        =0x03004000  @0x000020f0
    SPI_FirmwareAddress                                      =0x01ffff00  @0x000020f4
    SPI_Ram[0]                                               =0x7fb75d7d  @0x00002100
    SPROM_StatAddrWriteData                                  =0x001f0000  @0x00002300
    SPROM_ReadData                                           =0x001f0000  @0x00002304
    FPGA_ChipIdentification[0]                               =0x00e9400c  @0x00002400
    FPGA_ChipIdentification[1]                               =0x18730811  @0x00002404
    FPGA_ChipTemperature                                     =0x00000023  @0x00002410
    ClockGen_CSR                                             =0x00000003  @0x00002500
    ClockGen_access                                          =0x000d00f6  @0x00002504
    CalibrationCSR                                           =0x00000000  @0x00002600
    TestBusControl                                           =0x00000000  @0x00002604
    ADC_Enable                                               =0x00000001  @0x00003000
    ADC_ControlStatus[CCURPMFC_ADC_0]                        =0x00000001  @0x00003010
    ADC_ControlStatus[CCURPMFC_ADC_1]                        =0x00000001  @0x00003014
    ADC_FifoCSR                                              =0x81000000  @0x00003030
    ADC_FifoThreshold                                        =0x00020000  @0x00003034
    ADC_FifoChannelSelect                                    =0x0000ffff  @0x00003038
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_0]          =0x7fd5d725  @0x00003100
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_1]          =0x7fd0b404  @0x00003104
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_2]          =0x7fcc0e9f  @0x00003108
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_3]          =0x7fd41be3  @0x0000310c
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_4]          =0x7fc2410a  @0x00003110
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_5]          =0x7fd2ccac  @0x00003114
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_6]          =0x7ff9f82b  @0x00003118
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_7]          =0x7fd002ff  @0x0000311c
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_8]          =0x7fc9d098  @0x00003120
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_9]          =0x7fb24778  @0x00003124
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_10]         =0x7fc4b32d  @0x00003128
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_11]         =0x7fb61607  @0x0000312c
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_12]         =0x7fb6c93a  @0x00003130
    ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_13]         =0x7fe4526b  @0x00003134
```

```
ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_14]   =0x7fc54c68  @0x00003138
ADC_PositiveCalibration[CCURPMFC_ADC_CHANNEL_15]   =0x7fd45401  @0x0000313c
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_0]    =0x7fd42e90  @0x00003140
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_1]    =0x7fd1488d  @0x00003144
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_2]    =0x7fcae7da  @0x00003148
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_3]    =0x7fd5b5ef  @0x0000314c
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_4]    =0x7fc1635b  @0x00003150
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_5]    =0x7fd1c38b  @0x00003154
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_6]    =0x7ffa3a42  @0x00003158
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_7]    =0x7fce5fe4  @0x0000315c
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_8]    =0x7fc7cf4b  @0x00003160
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_9]    =0x7fb18546  @0x00003164
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_10]   =0x7fc57a75  @0x00003168
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_11]   =0x7fb5d398  @0x0000316c
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_12]   =0x7fb6057f  @0x00003170
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_13]   =0x7fe2e1a0  @0x00003174
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_14]   =0x7fc33037  @0x00003178
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_15]   =0x7fd3a2d9  @0x0000317c
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_0]      =0x00000002  @0x00003180
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_1]      =0x00000001  @0x00003184
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_2]      =0x00000000  @0x00003188
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_3]      =0x0000ffff  @0x0000318c
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_4]      =0x00000001  @0x00003190
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_5]      =0x00000001  @0x00003194
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_6]      =0x0000fffe  @0x00003198
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_7]      =0x00000000  @0x0000319c
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_8]      =0x00000001  @0x000031a0
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_9]      =0x00000000  @0x000031a4
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_10]     =0x00000000  @0x000031a8
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_11]     =0x0000ffff  @0x000031ac
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_12]     =0x00000000  @0x000031b0
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_13]     =0x00000000  @0x000031b4
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_14]     =0x0000fffe  @0x000031b8
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_15]     =0x0000ffff  @0x000031bc
ADC_Data[CCURPMFC_ADC_CHANNEL_0]                   =0x00007fff  @0x00003200
ADC_Data[CCURPMFC_ADC_CHANNEL_1]                   =0x00007fff  @0x00003204
ADC_Data[CCURPMFC_ADC_CHANNEL_2]                   =0x00007fff  @0x00003208
ADC_Data[CCURPMFC_ADC_CHANNEL_3]                   =0x00007fff  @0x0000320c
ADC_Data[CCURPMFC_ADC_CHANNEL_4]                   =0x00008000  @0x00003210
ADC_Data[CCURPMFC_ADC_CHANNEL_5]                   =0x00008000  @0x00003214
ADC_Data[CCURPMFC_ADC_CHANNEL_6]                   =0x00007fff  @0x00003218
ADC_Data[CCURPMFC_ADC_CHANNEL_7]                   =0x00007fff  @0x0000321c
ADC_Data[CCURPMFC_ADC_CHANNEL_8]                   =0x00007fff  @0x00003220
ADC_Data[CCURPMFC_ADC_CHANNEL_9]                   =0x00008000  @0x00003224
ADC_Data[CCURPMFC_ADC_CHANNEL_10]                  =0x00007fff  @0x00003228
ADC_Data[CCURPMFC_ADC_CHANNEL_11]                  =0x00008000  @0x0000322c
ADC_Data[CCURPMFC_ADC_CHANNEL_12]                  =0x00008000  @0x00003230
ADC_Data[CCURPMFC_ADC_CHANNEL_13]                  =0x00008000  @0x00003234
ADC_Data[CCURPMFC_ADC_CHANNEL_14]                  =0x00008000  @0x00003238
ADC_Data[CCURPMFC_ADC_CHANNEL_15]                  =0x00007fff  @0x0000323c
ADC_FifoData                                       =0xbaadbeef  @0x00003300
DAC_Enable                                         =0x00000000  @0x00004000
DAC_UpdateSourceSelect                             =0x00000000  @0x00004004
DAC_FifoChannelSelect                              =0x00000000  @0x00004008
DAC_ControlStatus[CCURPMFC_DAC_0]                  =0x00000000  @0x00004010
DAC_ControlStatus[CCURPMFC_DAC_1]                  =0x00000000  @0x00004014
DAC_ControlStatus[CCURPMFC_DAC_2]                  =0x00000000  @0x00004018
DAC_ControlStatus[CCURPMFC_DAC_3]                  =0x00000000  @0x0000401c
DAC_FifoCSR                                        =0x00000000  @0x00004030
DAC_FifoThreshold                                  =0x00000000  @0x00004034
DAC_FifoWriteCount                                 =0x00000000  @0x00004038
```

```
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_0]        =0x00000000  @0x00004100
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_1]        =0x00000000  @0x00004104
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_2]        =0x00000000  @0x00004108
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_3]        =0x00000000  @0x0000410c
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_4]        =0x00000000  @0x00004110
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_5]        =0x00000000  @0x00004114
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_6]        =0x00000000  @0x00004118
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_7]        =0x00000000  @0x0000411c
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_8]        =0x00000000  @0x00004120
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_9]        =0x00000000  @0x00004124
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_10]       =0x00000000  @0x00004128
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_11]       =0x00000000  @0x0000412c
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_12]       =0x00000000  @0x00004130
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_13]       =0x00000000  @0x00004134
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_14]       =0x00000000  @0x00004138
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_15]       =0x00000000  @0x0000413c
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_0]      =0x00000000  @0x00004140
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_1]      =0x00000000  @0x00004144
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_2]      =0x00000000  @0x00004148
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_3]      =0x00000000  @0x0000414c
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_4]      =0x00000000  @0x00004150
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_5]      =0x00000000  @0x00004154
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_6]      =0x00000000  @0x00004158
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_7]      =0x00000000  @0x0000415c
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_8]      =0x00000000  @0x00004160
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_9]      =0x00000000  @0x00004164
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_10]     =0x00000000  @0x00004168
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_11]     =0x00000000  @0x0000416c
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_12]     =0x00000000  @0x00004170
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_13]     =0x00000000  @0x00004174
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_14]     =0x00000000  @0x00004178
DAC_OffsetCalibration[CCURPMFC_DAC_CHANNEL_15]     =0x00000000  @0x0000417c
DAC_Data[CCURPMFC_DAC_CHANNEL_0]                   =0x00000000  @0x00004200
DAC_Data[CCURPMFC_DAC_CHANNEL_1]                   =0x00000000  @0x00004204
DAC_Data[CCURPMFC_DAC_CHANNEL_2]                   =0x00000000  @0x00004208
DAC_Data[CCURPMFC_DAC_CHANNEL_3]                   =0x00000000  @0x0000420c
DAC_Data[CCURPMFC_DAC_CHANNEL_4]                   =0x00000000  @0x00004210
DAC_Data[CCURPMFC_DAC_CHANNEL_5]                   =0x00000000  @0x00004214
DAC_Data[CCURPMFC_DAC_CHANNEL_6]                   =0x00000000  @0x00004218
DAC_Data[CCURPMFC_DAC_CHANNEL_7]                   =0x00000000  @0x0000421c
DAC_Data[CCURPMFC_DAC_CHANNEL_8]                   =0x00000000  @0x00004220
DAC_Data[CCURPMFC_DAC_CHANNEL_9]                   =0x00000000  @0x00004224
DAC_Data[CCURPMFC_DAC_CHANNEL_10]                  =0x00000000  @0x00004228
DAC_Data[CCURPMFC_DAC_CHANNEL_11]                  =0x00000000  @0x0000422c
DAC_Data[CCURPMFC_DAC_CHANNEL_12]                  =0x00000000  @0x00004230
DAC_Data[CCURPMFC_DAC_CHANNEL_13]                  =0x00000000  @0x00004234
DAC_Data[CCURPMFC_DAC_CHANNEL_14]                  =0x00000000  @0x00004238
DAC_Data[CCURPMFC_DAC_CHANNEL_15]                  =0x00000000  @0x0000423c
DAC_FifoData                                       =0x00000000  @0x00004300
DIO_Enable                                         =0x00000000  @0x00005000
DIO_Mode                                           =0x00000000  @0x00005004
DIO_InputSnapshot                                  =0x00000000  @0x00005008
DIO_OutputSync                                     =0x00000000  @0x0000500c
DIO_Direction                                      =0x00000000  @0x00005020
DIO_Set_OutputDirection                            =0x00000000  @0x00005024
DIO_Set_InputDirection                             =0x00000000  @0x00005028
DIO_OutputChannels[CCURPMFC_DIO_CHAN_00_31]        =0x00000000  @0x00005030
DIO_OutputChannels[CCURPMFC_DIO_CHAN_32_63]        =0x00000000  @0x00005034
DIO_OutputChannels[CCURPMFC_DIO_CHAN_64_95]        =0x00000000  @0x00005038
DIO_OutputChannelsX.chan_00_31                     =0x00000000  @0x00005030
DIO_OutputChannelsX.chan_32_63                     =0x00000000  @0x00005034
```

```
DIO_OutputChannelsX.chan_64_95                      =0x00000000  @0x00005038
DIO_Set_OutputChannelsHigh[CCURPMFC_DIO_CHAN_00_31]=0x00000000  @0x00005040
DIO_Set_OutputChannelsHigh[CCURPMFC_DIO_CHAN_32_63]=0x00000000  @0x00005044
DIO_Set_OutputChannelsHigh[CCURPMFC_DIO_CHAN_64_95]=0x00000000  @0x00005048
DIO_Set_OutputChannelsHighX.chan_00_31              =0x00000000  @0x00005040
DIO_Set_OutputChannelsHighX.chan_32_63              =0x00000000  @0x00005044
DIO_Set_OutputChannelsHighX.chan_64_95              =0x00000000  @0x00005048
DIO_Set_OutputChannelsLow[CCURPMFC_DIO_CHAN_00_31] =0x00000000  @0x00005050
DIO_Set_OutputChannelsLow[CCURPMFC_DIO_CHAN_32_63] =0x00000000  @0x00005054
DIO_Set_OutputChannelsLow[CCURPMFC_DIO_CHAN_64_95] =0x00000000  @0x00005058
DIO_Set_OutputChannelsLowX.chan_00_31               =0x00000000  @0x00005050
DIO_Set_OutputChannelsLowX.chan_32_63               =0x00000000  @0x00005054
DIO_Set_OutputChannelsLowX.chan_64_95               =0x00000000  @0x00005058
DIO_CustomChannels[CCURPMFC_DIO_CHAN_00_31]         =0x00000000  @0x00005060
DIO_CustomChannels[CCURPMFC_DIO_CHAN_32_63]         =0x00000000  @0x00005064
DIO_CustomChannels[CCURPMFC_DIO_CHAN_64_95]         =0x00000000  @0x00005068
DIO_CustomChannelsX.chan_00_31                      =0x00000000  @0x00005060
DIO_CustomChannelsX.chan_32_63                      =0x00000000  @0x00005064
DIO_CustomChannelsX.chan_64_95                      =0x00000000  @0x00005068
DIO_InputChannels[CCURPMFC_DIO_CHAN_00_31]          =0x00000000  @0x00005070
DIO_InputChannels[CCURPMFC_DIO_CHAN_32_63]          =0x00000000  @0x00005074
DIO_InputChannels[CCURPMFC_DIO_CHAN_64_95]          =0x00000000  @0x00005078
DIO_InputChannelsX.chan_00_31                       =0x00000000  @0x00005070
DIO_InputChannelsX.chan_32_63                       =0x00000000  @0x00005074
DIO_InputChannelsX.chan_64_95                       =0x00000000  @0x00005078
DIO_InputChannelsFilter[CCURPMFC_DIO_CHAN_00_31]    =0x00000000  @0x00005090
DIO_InputChannelsFilter[CCURPMFC_DIO_CHAN_32_63]    =0x00000000  @0x00005094
DIO_InputChannelsFilter[CCURPMFC_DIO_CHAN_64_95]    =0x00000000  @0x00005098
DIO_InputChannelsFilterX.chan_00_31                 =0x00000000  @0x00005090
DIO_InputChannelsFilterX.chan_32_63                 =0x00000000  @0x00005094
DIO_InputChannelsFilterX.chan_64_95                 =0x00000000  @0x00005098
DIO_ChannelsPolarity[CCURPMFC_DIO_CHAN_00_31]       =0x00000000  @0x000050a0
DIO_ChannelsPolarity[CCURPMFC_DIO_CHAN_32_63]       =0x00000000  @0x000050a4
DIO_ChannelsPolarity[CCURPMFC_DIO_CHAN_64_95]       =0x00000000  @0x000050a8
DIO_ChannelsPolarityX.chan_00_31                    =0x00000000  @0x000050a0
DIO_ChannelsPolarityX.chan_32_63                    =0x00000000  @0x000050a4
DIO_ChannelsPolarityX.chan_64_95                    =0x00000000  @0x000050a8
DIO_COS_ChannelsEnable[CCURPMFC_DIO_CHAN_00_31]     =0x00000000  @0x000050b0
DIO_COS_ChannelsEnable[CCURPMFC_DIO_CHAN_32_63]     =0x00000000  @0x000050b4
DIO_COS_ChannelsEnable[CCURPMFC_DIO_CHAN_64_95]     =0x00000000  @0x000050b8
DIO_COS_ChannelsEnableX.chan_00_31                  =0x00000000  @0x000050b0
DIO_COS_ChannelsEnableX.chan_32_63                  =0x00000000  @0x000050b4
DIO_COS_ChannelsEnableX.chan_64_95                  =0x00000000  @0x000050b8
DIO_COS_ChannelsMode[CCURPMFC_DIO_CHAN_00_31]       =0x00000000  @0x000050c0
DIO_COS_ChannelsMode[CCURPMFC_DIO_CHAN_32_63]       =0x00000000  @0x000050c4
DIO_COS_ChannelsMode[CCURPMFC_DIO_CHAN_64_95]       =0x00000000  @0x000050c8
DIO_COS_ChannelsModeX.chan_00_31                    =0x00000000  @0x000050c0
DIO_COS_ChannelsModeX.chan_32_63                    =0x00000000  @0x000050c4
DIO_COS_ChannelsModeX.chan_64_95                    =0x00000000  @0x000050c8
DIO_COS_ChannelsEdgeSense[CCURPMFC_DIO_CHAN_00_31] =0x00000000  @0x000050d0
DIO_COS_ChannelsEdgeSense[CCURPMFC_DIO_CHAN_32_63] =0x00000000  @0x000050d4
DIO_COS_ChannelsEdgeSense[CCURPMFC_DIO_CHAN_64_95] =0x00000000  @0x000050d8
DIO_COS_ChannelsEdgeSenseX.chan_00_31               =0x00000000  @0x000050d0
DIO_COS_ChannelsEdgeSenseX.chan_32_63               =0x00000000  @0x000050d4
DIO_COS_ChannelsEdgeSenseX.chan_64_95               =0x00000000  @0x000050d8
DIO_COS_ChannelsOverflow[CCURPMFC_DIO_CHAN_00_31]   =0x00000000  @0x000050e0
DIO_COS_ChannelsOverflow[CCURPMFC_DIO_CHAN_32_63]   =0x00000000  @0x000050e4
DIO_COS_ChannelsOverflow[CCURPMFC_DIO_CHAN_64_95]   =0x00000000  @0x000050e8
DIO_COS_ChannelsOverflowX.chan_00_31                =0x00000000  @0x000050e0
DIO_COS_ChannelsOverflowX.chan_32_63                =0x00000000  @0x000050e4
DIO_COS_ChannelsOverflowX.chan_64_95                =0x00000000  @0x000050e8
```

```
    DIO_COS_ChannelsStatus[CCURPMFC_DIO_CHAN_00_31]    =0x00000000  @0x000050f0
    DIO_COS_ChannelsStatus[CCURPMFC_DIO_CHAN_32_63]    =0x00000000  @0x000050f4
    DIO_COS_ChannelsStatus[CCURPMFC_DIO_CHAN_64_95]    =0x00000000  @0x000050f8
    DIO_COS_ChannelsStatusX.chan_00_31                 =0x00000000  @0x000050f0
    DIO_COS_ChannelsStatusX.chan_32_63                 =0x00000000  @0x000050f4
    DIO_COS_ChannelsStatusX.chan_64_95                 =0x00000000  @0x000050f8
    SDRAM_Enable                                       =0x00000000  @0x00007000
    SDRAM_CSR                                          =0x00000000  @0x00007004
    SDRAM_Address                                      =0x00000000  @0x00007008
    SDRAM_Data                                         =0x00000000  @0x0000700c
    DiagRam[0]                                         =0x00000000  @0x00008000
    FpgawbRevision                                     =0x00000000  @0x0001f000

======= CONFIG REGISTERS =========
    PcieLinkPartners.a2p_interrupt_status              =0x00000000  @0x00000040
    PcieLinkPartners.a2p_interrupt_enable              =0x00000000  @0x00000050

#### PCIe Link Partners (p2a_mailbox) #### (length=32)
+P2A+    0x800   00000000  00000000  00000000  00000000 *................*
+P2A+    0x810   00000000  00000000  00000000  00000000 *................*

#### PCIe Link Partners (a2p_mailbox) #### (length=32)
+A2P+    0x900   00000000  00000000  00000000  00000000 *................*
+A2P+    0x910   00000000  00000000  00000000  00000000 *................*
    DMAengine[CCURPMFC_DMA0].dma_status                =0x00000011  @0x00004000
    DMAengine[CCURPMFC_DMA0].dma_readaddress           =0x0000c000  @0x00004004
    DMAengine[CCURPMFC_DMA0].dma_writeaddress          =0x00903400  @0x00004008
    DMAengine[CCURPMFC_DMA0].dma_length                =0x00000000  @0x0000400c
    DMAengine[CCURPMFC_DMA0].dma_control               =0x00000000  @0x00004018

    DMAengine[CCURPMFC_DMA1].dma_status                =0x00000000  @0x00004020
    DMAengine[CCURPMFC_DMA1].dma_readaddress           =0x00000000  @0x00004024
    DMAengine[CCURPMFC_DMA1].dma_writeaddress          =0x00000000  @0x00004028
    DMAengine[CCURPMFC_DMA1].dma_length                =0x00000000  @0x0000402c
    DMAengine[CCURPMFC_DMA1].dma_control               =0x00000000  @0x00004038

    MsgDmaDispatcherCsr.Status                         =0x00000000  @0x00004200
    MsgDmaDispatcherCsr.Control                        =0x00000000  @0x00004204
    MsgDmaDispatcherCsr.ReadFillLevel                  =0x00000000  @0x00004208
    MsgDmaDispatcherCsr.WriteFillLevel                 =0x00000000  @0x0000420a
    MsgDmaDispatcherCsr.ResponseFillLevel              =0x00000000  @0x0000420c
    MsgDmaDispatcherCsr.ReadSequenceNumber             =0x00000000  @0x00004210
    MsgDmaDispatcherCsr.WriteSequenceNumber            =0x00000000  @0x00004212

    MsgDmaPrefetcherCsr.Control                        =0x00000000  @0x00004220
    MsgDmaPrefetcherCsr.NextDescriptorPointerLow       =0x00000000  @0x00004224
    MsgDmaPrefetcherCsr.NextDescriptorPointerHigh      =0x00000000  @0x00004228
    MsgDmaPrefetcherCsr.DescriptorPollingFrequency     =0x00000000  @0x0000422c
    MsgDmaPrefetcherCsr.Status                         =0x00000000  @0x00004230

=== Descriptor at offset 0 ===
    MsgDmaExtendedDescriptor[Id].ReadAddressLow        =0x00000000  @0x00004800
    MsgDmaExtendedDescriptor[Id].WriteAddressLow       =0x00000000  @0x00004804
    MsgDmaExtendedDescriptor[Id].Length                =0x00000000  @0x00004808
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerLow
                                                       =0x00000000  @0x0000480c
    MsgDmaExtendedDescriptor[Id].ActualBytesTransferred=0x00000000  @0x00004810
    MsgDmaExtendedDescriptor[Id].Status                =0x00000000  @0x00004814
    MsgDmaExtendedDescriptor[Id].SequenceNumber        =0x00000000  @0x0000481c
    MsgDmaExtendedDescriptor[Id].ReadBurstCount        =0x00000000  @0x0000481e
    MsgDmaExtendedDescriptor[Id].WriteBurstCount       =0x00000000  @0x0000481f
```

```
    MsgDmaExtendedDescriptor[Id].ReadStride            =0x00000000  @0x00004820
    MsgDmaExtendedDescriptor[Id].WriteStride           =0x00000000  @0x00004822
    MsgDmaExtendedDescriptor[Id].ReadAddressHigh       =0x00000000  @0x00004824
    MsgDmaExtendedDescriptor[Id].WriteAddressHigh      =0x00000000  @0x00004828
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh
                                                       =0x00000000  @0x0000482c
    MsgDmaExtendedDescriptor[Id].Control               =0x00000000  @0x0000483c

 === Descriptor at offset 1 ===
    MsgDmaExtendedDescriptor[Id].ReadAddressLow        =0x00000000  @0x00004840
    MsgDmaExtendedDescriptor[Id].WriteAddressLow       =0x00000000  @0x00004844
    MsgDmaExtendedDescriptor[Id].Length                =0x00000000  @0x00004848
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerLow
                                                       =0x00000000  @0x0000484c
    MsgDmaExtendedDescriptor[Id].ActualBytesTransferred=0x00000000  @0x00004850
    MsgDmaExtendedDescriptor[Id].Status                =0x00000000  @0x00004854
    MsgDmaExtendedDescriptor[Id].SequenceNumber        =0x00000000  @0x0000485c
    MsgDmaExtendedDescriptor[Id].ReadBurstCount        =0x00000000  @0x0000485e
    MsgDmaExtendedDescriptor[Id].WriteBurstCount       =0x00000000  @0x0000485f
    MsgDmaExtendedDescriptor[Id].ReadStride            =0x00000000  @0x00004860
    MsgDmaExtendedDescriptor[Id].WriteStride           =0x00000000  @0x00004862
    MsgDmaExtendedDescriptor[Id].ReadAddressHigh       =0x00000000  @0x00004864
    MsgDmaExtendedDescriptor[Id].WriteAddressHigh      =0x00000000  @0x00004868
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh
                                                       =0x00000000  @0x0000486c
    MsgDmaExtendedDescriptor[Id].Control               =0x00000000  @0x0000487c

=== Descriptor at offset 2 ===
    MsgDmaExtendedDescriptor[Id].ReadAddressLow        =0x00000000  @0x00004880
    MsgDmaExtendedDescriptor[Id].WriteAddressLow       =0x00000000  @0x00004884
    MsgDmaExtendedDescriptor[Id].Length                =0x00000000  @0x00004888
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerLow
                                                       =0x00000000  @0x0000488c
    MsgDmaExtendedDescriptor[Id].ActualBytesTransferred=0x00000000  @0x00004890
    MsgDmaExtendedDescriptor[Id].Status                =0x00000000  @0x00004894
    MsgDmaExtendedDescriptor[Id].SequenceNumber        =0x00000000  @0x0000489c
    MsgDmaExtendedDescriptor[Id].ReadBurstCount        =0x00000000  @0x0000489e
    MsgDmaExtendedDescriptor[Id].WriteBurstCount       =0x00000000  @0x0000489f
    MsgDmaExtendedDescriptor[Id].ReadStride            =0x00000000  @0x000048a0
    MsgDmaExtendedDescriptor[Id].WriteStride           =0x00000000  @0x000048a2
    MsgDmaExtendedDescriptor[Id].ReadAddressHigh       =0x00000000  @0x000048a4
    MsgDmaExtendedDescriptor[Id].WriteAddressHigh      =0x00000000  @0x000048a8
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh
                                                       =0x00000000  @0x000048ac
    MsgDmaExtendedDescriptor[Id].Control               =0x00000000  @0x000048bc

 === Descriptor at offset 3 ===
    MsgDmaExtendedDescriptor[Id].ReadAddressLow        =0x00000000  @0x000048c0
    MsgDmaExtendedDescriptor[Id].WriteAddressLow       =0x00000000  @0x000048c4
    MsgDmaExtendedDescriptor[Id].Length                =0x00000000  @0x000048c8
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerLow
                                                       =0x00000000  @0x000048cc
    MsgDmaExtendedDescriptor[Id].ActualBytesTransferred=0x00000000  @0x000048d0
    MsgDmaExtendedDescriptor[Id].Status                =0x00000000  @0x000048d4
    MsgDmaExtendedDescriptor[Id].SequenceNumber        =0x00000000  @0x000048dc
    MsgDmaExtendedDescriptor[Id].ReadBurstCount        =0x00000000  @0x000048de
    MsgDmaExtendedDescriptor[Id].WriteBurstCount       =0x00000000  @0x000048df
    MsgDmaExtendedDescriptor[Id].ReadStride            =0x00000000  @0x000048e0
    MsgDmaExtendedDescriptor[Id].WriteStride           =0x00000000  @0x000048e2
    MsgDmaExtendedDescriptor[Id].ReadAddressHigh       =0x00000000  @0x000048e4
    MsgDmaExtendedDescriptor[Id].WriteAddressHigh      =0x00000000  @0x000048e8
```

```
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh
                                                    =0x00000000  @0x000048ec
    MsgDmaExtendedDescriptor[Id].Control            =0x00000000  @0x000048fc
        .
        .
        .
=== Descriptor at offset 29 ===
    MsgDmaExtendedDescriptor[Id].ReadAddressLow     =0x00000000  @0x00004f40
    MsgDmaExtendedDescriptor[Id].WriteAddressLow    =0x00000000  @0x00004f44
    MsgDmaExtendedDescriptor[Id].Length             =0x00000000  @0x00004f48
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerLow
                                                    =0x00000000  @0x00004f4c
    MsgDmaExtendedDescriptor[Id].ActualBytesTransferred=0x00000000  @0x00004f50
    MsgDmaExtendedDescriptor[Id].Status             =0x00000000  @0x00004f54
    MsgDmaExtendedDescriptor[Id].SequenceNumber     =0x00000000  @0x00004f5c
    MsgDmaExtendedDescriptor[Id].ReadBurstCount     =0x00000000  @0x00004f5e
    MsgDmaExtendedDescriptor[Id].WriteBurstCount    =0x00000000  @0x00004f5f
    MsgDmaExtendedDescriptor[Id].ReadStride         =0x00000000  @0x00004f60
    MsgDmaExtendedDescriptor[Id].WriteStride        =0x00000000  @0x00004f62
    MsgDmaExtendedDescriptor[Id].ReadAddressHigh    =0x00000000  @0x00004f64
    MsgDmaExtendedDescriptor[Id].WriteAddressHigh   =0x00000000  @0x00004f68
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh
                                                    =0x00000000  @0x00004f6c
    MsgDmaExtendedDescriptor[Id].Control            =0x00000000  @0x00004f7c

 === Descriptor at offset 30 ===
    MsgDmaExtendedDescriptor[Id].ReadAddressLow     =0x00000000  @0x00004f80
    MsgDmaExtendedDescriptor[Id].WriteAddressLow    =0x00000000  @0x00004f84
    MsgDmaExtendedDescriptor[Id].Length             =0x00000000  @0x00004f88
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerLow
                                                    =0x00000000  @0x00004f8c
    MsgDmaExtendedDescriptor[Id].ActualBytesTransferred=0x00000000  @0x00004f90
    MsgDmaExtendedDescriptor[Id].Status             =0x00000000  @0x00004f94
    MsgDmaExtendedDescriptor[Id].SequenceNumber     =0x00000000  @0x00004f9c
    MsgDmaExtendedDescriptor[Id].ReadBurstCount     =0x00000000  @0x00004f9e
    MsgDmaExtendedDescriptor[Id].WriteBurstCount    =0x00000000  @0x00004f9f
    MsgDmaExtendedDescriptor[Id].ReadStride         =0x00000000  @0x00004fa0
    MsgDmaExtendedDescriptor[Id].WriteStride        =0x00000000  @0x00004fa2
    MsgDmaExtendedDescriptor[Id].ReadAddressHigh    =0x00000000  @0x00004fa4
    MsgDmaExtendedDescriptor[Id].WriteAddressHigh   =0x00000000  @0x00004fa8
    MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh
                                                    =0x00000000  @0x00004fac
    MsgDmaExtendedDescriptor[Id].Control            =0x00000000  @0x00004fbc

 === Terminating Descriptor at offset 31 ===
    MsgDmaTerminatingDescriptor.ReadAddressLow      =0x00000000  @0x00004fc0
    MsgDmaTerminatingDescriptor.WriteAddressLow     =0x00000000  @0x00004fc4
    MsgDmaTerminatingDescriptor.Length              =0x00000000  @0x00004fc8
    MsgDmaTerminatingDescriptor.NextDescriptorPointerLow
                                                    =0x00000000  @0x00004fcc
    MsgDmaTerminatingDescriptor.ActualBytesTransferred =0x00000000  @0x00004fd0
    MsgDmaTerminatingDescriptor.Status              =0x00000000  @0x00004fd4
    MsgDmaTerminatingDescriptor.SequenceNumber      =0x00000000  @0x00004fdc
    MsgDmaTerminatingDescriptor.ReadBurstCount      =0x00000000  @0x00004fde
    MsgDmaTerminatingDescriptor.WriteBurstCount     =0x00000000  @0x00004fdf
    MsgDmaTerminatingDescriptor.ReadStride          =0x00000000  @0x00004fe0
    MsgDmaTerminatingDescriptor.WriteStride         =0x00000000  @0x00004fe2
    MsgDmaTerminatingDescriptor.ReadAddressHigh     =0x00000000  @0x00004fe4
    MsgDmaTerminatingDescriptor.WriteAddressHigh    =0x00000000  @0x00004fe8
    MsgDmaTerminatingDescriptor.NextDescriptorPointerHigh
                                                    =0x00000000  @0x00004fec
```

```
    MsgDmaTerminatingDescriptor.Control               =0x00000000  @0x00004ffc
```

3.1.5 ccurpmfc_regedit

This is an interactive test to display and write to local, configuration and physical memory.

Usage: ./ccurpmfc_regedit [-b board]
-b board: Board number -- default board is 0

Example display:

./ccurpmfc_regedit

```
Device Name: /dev/ccurpmfc0

  LOCAL REGION: Physical Addr=0xc4900000 Size=65536 (0x00010000)
 CONFIG REGION: Physical Addr=0xc4910000 Size=32768 (0x00008000)

         LOCAL: Register 0x7ffff7fe7000 Offset=0x0 Size=0x00010000
        CONFIG: Register 0x7ffff7fdf000 Offset=0x0 Size=0x00008000
        LIBPTR: Register 0x7ffff7fdd000 Offset=0x0 Size=0x00001070

Initialize_Board: Firmware Rev. 0x550003 successful

Virtual Address: 0x7ffff7fe7000
   1 = Create Physical Memory         2 = Destroy Physical memory
   3 = Display Channel Data           4 = Display Driver Information
   5 = Display Physical Memory Info   6 = Display Registers (CONFIG)
   7 = Display Registers (LOCAL)      8 = Dump Physical Memory
   9 = Reset Board                   10 = Write Register (LOCAL)
  11 = Write Register (CONFIG)       12 = Write Physical Memory

Main Selection ('h'=display menu, 'q'=quit)->
```

3.1.6 ccurpmfc_tst

This is an interactive test to exercise some of the driver features.

Usage: ./ccurpmfc_tst [-b board]
-b board: Board number -- default board is 0

Example display:

./ccurpmfc_tst

```
Device Name: /dev/ccurpmfc0

  LOCAL REGION: Physical Addr=0xc4900000 Size=65536 (0x00010000)
 CONFIG REGION: Physical Addr=0xc4910000 Size=32768 (0x00008000)

         LOCAL: Register 0x7ffff7fe7000 Offset=0x0 Size=0x00010000
        CONFIG: Register 0x7ffff7fdf000 Offset=0x0 Size=0x00008000
        LIBPTR: Register 0x7ffff7fdd000 Offset=0x0 Size=0x00001070
Initialize_Board: Firmware Rev. 0x550003 successful

  01 = add irq                       02 = disable pci interrupts
  03 = enable pci interrupts         04 = get device error
  05 = get driver info               06 = get physical memory
  07 = init board                    08 = mmap select
```

```
  09 = mmap(CONFIG registers)        10 = mmap(LOCAL registers)
  11 = mmap(physical memory)         12 = munmap(physical memory)
  13 = no command                    14 = read operation
  15 = remove irq                    16 = reset board
  17 = restore config registers      18 = write operation

Main Selection ('h'=display menu, 'q'=quit)->
```

3.1.7 ccurpmfc_wreg

This is a simple test to write to the local registers at the user specified offset.

```
Usage: ./ccurpmfc_wreg [-b Board] [-C] [-o Offset] [-s Size] [-v Value] [-x]
 -b Board   : Board selection --  default board is 0
 -C         : Select Config Registers instead of Local Registers
 -o Offset  : Hex offset to write to -- default offset is 0x0
 -s Size    : Number of bytes to write in decimal -- default size is 0x4
 -v Value   : Hex value to write at offset -- default value is 0x0
 -x         : Do not read back just written values -- default read back values
```

Example display:

./ccurpmfc_wreg -v12345678 -o0x8000 -s400

```
Device Name: /dev/ccurpmfc0

  LOCAL REGION: Physical Addr=0xc4900000 Size=65536 (0x00010000)
 CONFIG REGION: Physical Addr=0xc4910000 Size=32768 (0x00008000)

         LOCAL: Register 0x7ffff7fe7000 Offset=0x0 Size=0x00010000
        CONFIG: Register 0x7ffff7fdf000 Offset=0x0 Size=0x00008000
        LIBPTR: Register 0x7ffff7fdd000 Offset=0x0 Size=0x00001070

Writing 0x12345678 to offset 0x8000 for 400 bytes

#### LOCAL REGS #### (length=400)
+LCL+   0x8000   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8010   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8020   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8030   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8040   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8050   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8060   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8070   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8080   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8090   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x80a0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x80b0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x80c0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x80d0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x80e0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x80f0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8100   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8110   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8120   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8130   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8140   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8150   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8160   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8170   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+   0x8180   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
```

3.1.8 Flash/ccurpmfc_flash

This program is used to burn new firmware or update the license of an already installed firmware. It can also be used to reload the firmware on the card. This must only be done at the direction of Concurrent Real-Time support team, otherwise, they could render the board useless.

```
./ccurpmfc_flash -b[Board] -B –F[!] -i -L -q -Q -r[OutFile] -R -v -w[InFile] -X
-b [Board]        : Board number. Must be specified
-B                : Reload Base Level Firmware if MultiFirmware support present
-F                : Force Read Flash:  Overwrite output file if exists
-F                : Force Write Flash: Do not abort Flash burn for header label
                    mismatch
-F!               : Force Write Flash: Serious override required to continue
                    burning
-i                : Query chip, on-board flash and InFile if specified
-L                : Update License only. (default is to update entire firmware)
-q                : Quite (non-interactive) mode
-Q                : Quite (non-interactive) mode. Also dump FPGAWB message
-r                : Read Flash and write to output file created by
                    ./ccurpmfc_flash
-r [OutFile]      : Read Flash and write to output file 'OutFile'
-R                : Reload Firmware at sector address in Flash
-R [SectorNumber] : Reload Firmware at sector address 'SectorNumber'
-v                : Enable verbose mode
-w [InFile]       : Read input FPGA file and Flash the board
-X                : Use Full File. Do not truncate for firmware write

  ================================== Notes ====================================
  Board must be specifed. Use '-b' option
  Query option '-i' not allowed with '-B', '-R#', '-L', 'r' or '-X' options
  Firmware reload '-B' or '-R' not allowed with '-i', '-L', '-r', '-w' or '-X'
  options
  Firmware read flash '-r' not allowed with '-B', '-i', '-L', '-R', '-w' or '-X'
  options
  Base Run Level '-B' or '-R#' option not allowed with '-i', '-L', 'r', '-w' or
  '-X' options
  Must specify write flash option '-w' when License only option '-L' is specified
  License only option '-L' not allowed with '-B', '-i', '-R', '-w' or '-X'
  options
  Don't truncate file option '-X' cannot be selected with the license only update
  '-L' option
  Don't truncate file option '-X' can only be used with the '-w' option
  Inquiry '-i' can be used '-w' options
  ============================================================================

e.g. ./ccurpmfc_flash -b0                 (Query chip and on-board Flash)
     ./ccurpmfc_flash -b0 -i              (Query chip and on-board Flash)
     ./ccurpmfc_flash -b0 -i -w InFile    (Query chip, on-board Flash and InFile)
     ./ccurpmfc_flash -b0 -r OutFile      (On-board FPGA ===> OutFile)
     ./ccurpmfc_flash -b0 -w InFile       (InFile ===> On-board FPGA - use
                                           truncated file)
     ./ccurpmfc_flash -b0 -w InFile -v    (InFile ===> On-board FPGA - use
                                           truncated file - verbose)
     ./ccurpmfc_flash -b0 -w InFile -X    (InFile ===> On-board FPGA - use entire
                                           file)
     ./ccurpmfc_flash -b0 -w InFile -L    (InFile ===> On-board FPGA - only
                                           license updated - interactive)
     ./ccurpmfc_flash -b0 -w InFile -L -q (InFile ===> On-board FPGA - only
                                           license updated - non-interactive)
     ./ccurpmfc_flash -b0 -R              (Reload Firmware - i.e. power-cycle the
                                           card) - Run Level
     ./ccurpmfc_flash -b0 -B              (Reload Firmware - i.e. power-cycle the
```

```
                                        card) - Base Level
    ./ccurpmfc_flash -b0 -R 0          (Reload Firmware - i.e. power-cycle the
                                        card) - Base Level
    ./ccurpmfc_flash -b0 -R 200        (Reload Firmware - i.e. power-cycle the
                                        card) - at sector 200
```

If the installed firmware is a Multi-Level firmware and you are running at Base Level, then the only utility that will be able to access the card will be this ***ccurpmfc_flash*** *utility. You will need to switch to Run Level before un-restricted access is allowed to the card.*

3.1.9 Flash/ccurpmfc_label

This utility is only supplied for those customers that are creating their own firmware and need to install in a RedHawk system. In its simplest form, the customer will request a License file from Concurrent Real-Time for the option to burn their custom firmware. The license file (*.lic) supplied by Concurrent Real-Time, along with the customer firmware (*.rpd) file will be supplied to this utility to create a burnable FPGA file (*.cust), that will be supplied to the *ccurpmfc_flash* utility to burn the firmware on the card.

The user can also supply the '-x' option to additionally create a license only file (*.cust.liconly) file that is associated with the firmware (.rpd). This is useful if you only wish to update the license information of a card that already has the same firmware installed. This is similar to having a (*.cust) file and using the '-L' option when running the *ccurpmfc_flash* utility.

```
./ccurpmfc_label -d[OutputDirectory] -c[ChipName] -F -i[InputFile] -K[FpgawbKey]
                 -L[LicenseFile] -m[MemberCode] -o[OutputFile]
                 -S[RunLevelSectorAddress] -t[Tag] -x
-d [OutputDirectory]       : Directory to use for Output File
-c [ChipName]              : Chip Name. One of:
                             EPCQ16 EPCQ32 EPCQ64 EPCQ128 EPCQ256 EPCQ512
                             (This option is mandatory if not specified in license file)
-F                         : Force overwriting of output file if it exists
-i [InputFile]             : Raw input file. (.rpd extension)
-K [FpgawbKey]             : Fpgawb Key is required if license contains FPGA workbench
                             restriction
-L [LicenseFile]           : License file (.lic extension) to restrict firmware access (this
                             option is mandatory)
                             If '-i' option is not specified, the license file is dumped to
                             stderr
-m [MemberCode]            : Specify Member Code (A1,A3,A5,A7,B1,B3,B5,B7)
                             (This option is mandatory if not specified in license file)
-o [OutputFile]            : Use output file instead of the default file created by the
                             program
-S [RunLevelSectorAddress] : Run Level Sector Address. (This option is mandatory if not
                             specified in license file)
                           : S0=Base Level, S#=Run Level Number
-t [Tag]                   : Insert this tag name in the default file created by the program
-x                         : Create an additional License only file (*.liconly)

==== Notes ====
- Options '-L' is required. If option '-i' is not specified, license file is dumped
- Options 'c', '-m' and '-S' are required if they have not already been defined in
  LicenseFile
- You cannot specify a Run Level Sector '-S' with Single Level Firmware '-1' option
- Run Level Sector address of zero '-S0' represents the Base Level Firmware in Multi-
  Firmware support
- If option '-o' is not specified, the created customer FPGA file name will be as follows:
  <OutputDirectory>/<InputFile>_<Tag>_<Function>_<ChipName><MemberCode><RunLevel>.cust
- If the license file contains an FPGAWB restrict key, then the '-K' FpgawbKey is required

e.g. ./ccurpmfc_label -iraw_file.rpd -L LicenseFile.lic (in its simplest form)
       (output file created is: 'raw_file_<Function>_<ChipName><MemberCode><RunLevel>.cust')
     ./ccurpmfc_label -L LicenseFile.lic (this will display licensing information)
```

```
./ccurpmfc_label -iraw_RUN_file.rpd -ooutput_file.cust -S100 -L LicenseFile.lic
./ccurpmfc_label -iraw_SINGLE_file.rpd -L LicenseFile.lic
./ccurpmfc_label -iraw_RUN_file.rpd -ooutput_file.cust -S200 -L LicenseFile.lic
./ccurpmfc_label -iraw_BASE_file.rpd -S0 -L LicenseFile.lic
  (Will cause firmware to be loaded at start offset Base Run Level)
```

3.1.10 Flash/ccurpmfc_dump_license

This utility allows the customer to dump the license information from a firmware *(*.cust)* file or the *(*.liconly)* file.

```
Format: ./ccurpmfc_dump_license <Firmware file>

    This utility only dumps the license information from the *.cust or *.liconly files
    and not the *.lic license file

    e.g ./ccurpmfc_dump_license COS_IpCoreCOS_EPCQ256A5S100.cust
        ./ccurpmfc_dump_license COS_IpCoreCOS_EPCQ256A5S100.cust.liconly
```

3.2 Application Program Interface (API) Access Example Tests

These set of tests are in the *.../test/lib* directory and use the API.

3.2.1 lib/ccurpmfc_adc

This test performs validation of the Multi-Function ADC card.

```
Usage: ./ccurpmfc_adc [-A] [-a RollingAve] [-b BoardNo] [-C AdcUpdateClock]
                      [-d Delay] [-D DMAEngine] [-E ExpInpVolt] [-f DataFormat]
                      [-F DebugFile] [-i] [-l LoopCnt] [-m XferMode]
                      [-n NumChans] [-N] [-s InputSignal] [-t Compare]
                      [-T TestBus] [-V MaxBoardVolts]
 -A                      (Perform Auto Calibration first using reference voltage)
 -a RollingAve           (Rolling average -- default "=== None ===")
 -b BoardNo              (Board number -- default is 0)
 -C AdcUpdateClock       (select ADC update clock, 0..6 or 'n|N')
    -C 0,6               (Ch0..7=Clock0, Ch8..15=Clock6 at MAX SPS)
    -C 6@20000.0/n       (Ch0..7=Clock6 at 20000 SPS, Ch8..15=No Clock)
    -C 4                 (Ch0..15=Clock4 at MAX SPS)
    -C 4@150000.0        (Ch0..15=Clock4) at 150000 SPS
 -d Delay                (Delay between screen refresh -- default is 0
                                                          milli-seconds)
 -D DMA Engine           (DMA Engine number -- default = 1)
 -E <ExpInpVolts>@<Tol>  (Expected Input Volts@Tolerance -- default Tol=0.003000)
    +@<Tol>              (Positive Calibration Ref Volt@Tolerance)
    -@<Tol>              (Negative Calibration Ref Volt@Tolerance)
    s@<Tol>              (Requires '-s' input signal option to specify voltage
                          Volt@Tolerance)
                         (valid '-s' arguments are 'g','+','-','f','t')
 -f DataFormat           (select data format, '2' or 'b')
    -f b,2               (Ch0..7=Offset binary, Ch8..15=Two's complement)
    -f 2/b               (Ch0..7=Two's complement, Ch8..15=Offset binary)
    -f b                 (Ch0..15=Offset binary)
 -F DebugFile            (Debug file with menu display -- default "=== None ===")
    #DebugFile           (Debug file without display (only summary) -- default
                                                          "=== None ===")
    @DebugFile           (Debug file without display -- default "=== None ===")
    ~DebugFile           (For gnuplot, no header or summary -- default
                                                          "=== None ===")
    @, # or ~            (No debug file and no display -- default "=== None ===")
 -i                      (Enable Interrupts -- default = Disable)
 -l LoopCnt              (Loop count -- default is 0)
 -m XferMode             (Transfer Mode -- default = 'DMA Channel')
    -mdp                 (Driver:  (Channel Registers) PIO mode)
    -mdP                 (Driver:  (FIFO) PIO mode)
    -mlc                 (Library: (Channel Registers) program I/O Fast Memory
                                                                   Copy)
    -mld                 (Library: (Channel Registers) DMA mode)
    -mlD                 (Library: (FIFO) DMA mode)
    -mlp                 (Library: (Channel Registers) PIO)
    -mlP                 (Library: (FIFO) PIO mode)
 -n NumChans             (Number of channels -- default is 16)
 -N                      (Open device with O_NONBLOCK flag)
 -s InputSignal          (select input signal, 'e', 'g', '+', '-', 't', 'f',
                                                               '0..15')
    -s e,g               (Ch0..7=External input, Ch8..15=ground calibration)
    -s +/e               (Ch0..7=Postive calibration, Ch8..15=external reference)
    -s -                 (Ch0..15=Negative calibration)
    -s t                 (Ch0..15=2.5 volt calibration)
    -s e/f               (Ch0..7=external reference, Ch8..15=5 volt calibration)
```

```
   -s e/12              (Ch0..7=external reference, Ch8..15=DAC Channel 12)
-t Compare              (Compare two channels for +/- -- default is
                                                          "=== None ===")
   -t0,15               (Compare channel 0 and 15 for being in sync)
   -t5/7                (Compare channel 5 and 7 for being in sync)
   -t12,4@0.500         (Compare channel 4 and 12 for being in sync with 0.5V
                                                            tolerance)
-T TestBus              (Test Bus Control 'b' or 'o'. Exit after programming
                                                          this option)
   -T b                 (Calibration Bus Control)
   -T o                 (Open Bus Control)
-V MaxBoardVolts        (Voltage range 'b5' or 'b10')
   -V b5,b10            (Ch0..7=5V, Ch8..15=10V)
   -Vb10/b5             (Ch0..7=10V, Ch8..15=5V)
   -V b10               (Ch0..15=10V)

e.g. ./ccurpmfc_adc -A -C0@150000.0/1@1234.0 -se/+ (Autocal, ADC0=150000Hz
                                                    external input, ADC1=1234Hz
                                                    Positive Cal.)
     ./ccurpmfc_adc -A -C0 -s+ -E+                (Autocal, Max Clock, Positive
                                                    cal. input, validate result)
     ./ccurpmfc_adc -A -C0 -s- -t0,15 -a100       (Autocal, Max Clock, Negative
                                                    cal. input, compare ch0 and
                                                    ch15, rolling ave=100)
     ./ccurpmfc_adc -C0 -Vb10 -s- -Es             (Max Clock, -9.91V input,
                                                    validate against -9.91V)
     ./ccurpmfc_adc -C0 -Vb10 -st -Es             (Max Clock, +2.5V input,
                                                    validate against +2.5V)
```

Example display:

./ccurpmfc_adc -A -C0@150000.0/1@1234.0 -se/+

```
local_ptr=0x7ffff7fd7000
           Physical Memory Information:
             UserPID         =22341
             PhysMemPtr      =0x352d9000
             DriverVirtMemPtr=0xffff8800352d9000
             MmapedUserMemPtr=0x7ffff7fcc000
             PhysMemSize     =0x00001000
             PhysMemSizeFreed=0x00000000
             EntryInTxTbl    =0
             NumOfEntriesUsed=1
             Flags           =0x0000
Auto Calibration started...done. (2.357 seconds)

 Board Number          [-b]: 0
 Update Clock Selected [-C]: Ch00..07 OutputClock=0 (0x7) (150000.000 SPS)
                           : Ch08..15 OutputClock=1 (0x1) (1234.000 SPS)
 Delay                 [-d]: 0 milli-seconds
 DMA Engine            [-D]: 1
 Expected Input Volts  [-E]: === Not Specified ===
 Data Format           [-f]: Ch00..07 Offset binary (0x0)
                           : Ch08..15 Offset binary (0x0)
 Interrupts            [-i]: Disabled
 Loop Count            [-l]: ***Forever***
 Transfer Mode         [-m]: Library: (Channel Registers) DMA I/O
 Number of Channels    [-n]: 16
 Input Signal          [-s]: Ch00..07 [0]External Input
                           : Ch08..15 [1]Calibration Input (0x01: Positive 9.91)
 Voltage Range         [-V]: Ch00..07 +/-10 Volts (0x1)
```

```
                          : Ch08..15 +/-10 Volts (0x1)

 Scan Count                : 51661          (0:00:02:07)
 Read Duration (microsecs) : TotalDelta:     7.266 (min=   7.064/max=  74.005/ave=   7.390)

          ##### Raw Data #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       ====     ====     ====     ====     ====     ====     ====     ====     ====     ====
 [0]   7fff     8000     7fff     7fff     8000     8001     7fff     8000     fed6     fed6

 [1]   fed8     fed9     fed7     fed8     fed8     fed8

          ##### Volts #####
     [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
     =======  =======  =======  =======  =======  =======  =======  =======  =======  =======
 [0] -0.0003  +0.0000  -0.0003  -0.0003  +0.0000  +0.0003  -0.0003  +0.0000  +9.9091  +9.9091

 [1] +9.9097  +9.9100  +9.9094  +9.9097  +9.9097  +9.9097

=====================================================
                   Date: Mon Oct 25 12:43:14 2017
   Expected Input Volts: === Not Specified ===
           Scan Counter: 246540
   WorstMinChanVoltsHWM:  -0.001497 (Ch06)
   WorstMaxChanVoltsHWM:   9.914379 (Ch14)
=====================================================
      <-------- (volts) -------->
Chan  Min       Max       Ave      TolerExeededCnt
====  =======   =======   =======  ===============
 00   -0.0015    0.0006   -0.0002         -
 01   -0.0009    0.0009   -0.0000         -
 02   -0.0012    0.0000   -0.0003         -
 03   -0.0009    0.0003   -0.0002         -
 04   -0.0006    0.0021    0.0002         -
 05   -0.0006    0.0018    0.0001         -
 06   -0.0015    0.0021   -0.0002         -
 07   -0.0006    0.0003   -0.0000         -
 08    9.9075    9.9121    9.9100         -
 09    9.9081    9.9118    9.9101         -
 10    9.9078    9.9124    9.9099         -
 11    9.9078    9.9121    9.9101         -
 12    9.9069    9.9124    9.9098         -
 13    9.9081    9.9124    9.9101         -
 14    9.9078    9.9124    9.9101         -
 15    9.9078    9.9121    9.9101         -
=====================================================
```

3.2.2 lib/ccurpmfc_adc_calibrate

This test is useful for performing, saving and restoring ADC calibration.

```
Usage: ./ccurpmfc_adc_calibrate [-A] [-b board] [-i inCalFile] [-o outCalFile]
                                [-R]
 -A                    (perform Auto Calibration)
 -b <board>            (board #, default = 0)
 -i <In Cal File>      (input calibration file [input->board_reg])
 -o <Out Cal File>     (output calibration file [board_reg->output])
 -R                    (reset ADC calibration)

 e.g. ./ccurpmfc_adc_calibrate              (Dump calibration information to
                                             stdout)
```

```
      ./ccurpmfc_adc_calibrate -A -o Calfile (Perform Auto calibration and dump
                                              information to 'Calfile')
      ./ccurpmfc_adc_calibrate -i Calfile    (Update board calibration with
                                              supplied 'Calfile')
      ./ccurpmfc_adc_calibrate -R            (Reset ADC calibration)
```

Example display:

```
./ccurpmfc_adc_calibrate -A
Device Name    : /dev/ccurpmfc0
Board Serial No: 674459 (0x000a4a9b)
Auto Calibration started...done. (0.721 seconds)

===> Dump to 'stdout'
#Date           : Thu Jan 20 10:01:28 2022

#Chan   Negative                Offset                  Positive
#====   ========                ======                  ========
 ch00:  0.99787505948916077614  0.00030517578125000000  0.99782201414927840233
 ch01:  0.99794836621731519699 -0.00030517578125000000  0.99802144151180982590
 ch02:  0.99645603867247700691 -0.00030517578125000000  0.99649177165701985359
 ch03:  0.99683960061520338058 -0.00061035156250000000  0.99686906719580292702
 ch04:  0.99678667308762669563 -0.00030517578125000000  0.99685290874913334846
 ch05:  0.99729985604062676430 -0.00030517578125000000  0.99740492785349488258
 ch06:  0.99751370400190353394 -0.00091552734375000000  0.99759460566565394402
 ch07:  0.99769832380115985870 -0.00061035156250000000  0.99774155439808964729
 ch08:  0.99819327518343925476  0.00061035156250000000  0.99818982416763901711
 ch09:  0.99846281437203288078 -0.00030517578125000000  0.99854838801547884941
 ch10:  0.99665071209892630577  0.00030517578125000000  0.99664338957518339157
 ch11:  0.99724958604201674461  0.00030517578125000000  0.99725262960419058800
 ch12:  0.99790062708780169487  0.00030517578125000000  0.99792200094088912010
 ch13:  0.99731464916840195656  0.00030517578125000000  0.99733714759349822998
 ch14:  0.99746086820960044861  0.00000000000000000000  0.99751773616299033165
 ch15:  0.99614016432315111160  0.00030517578125000000  0.99612911511212587357
```

3.2.3 lib/ccurpmfc_adc_fifo

This test performs validation of the Multi-Function ADC FIFO operation of the card.

```
Usage: ./ccurpmfc_adc_fifo [-A] [-b BoardNo] [-c ChannelSelectMask]
                           [-C AdcUpdateClock] [-d Delay] [-D DMAEngine]
                           [-E ExpInpVolt] [-f DataFormat] [-F DebugFile] [-i]
                           [-l LoopCnt] [-m XferMode] [-N] [-s InputSignal]
                           [-S NumberOfSamples] [-T TestBus] [-V MaxBoardVolts]
 -A                       (Perform Auto Calibration first using reference voltage)
 -b BoardNo               (Board number -- default is 0)
 -c ChannelSelectMask     (Specify channel selection mask 0x0..0xffff)
 -C AdcUpdateClock        (select ADC update clock, 0..6 or 'n|N')
    -C 0,6                (Ch0..7=Clock0, Ch8..15=Clock6 at MAX SPS)
    -C 6@20000.0/n        (Ch0..7=Clock6 at 20000 SPS, Ch8..15=No Clock)
    -C 4                  (Ch0..15=Clock4 at MAX SPS)
    -C 4@150000.0         (Ch0..15=Clock4) at 150000 SPS
 -d Delay                 (Delay between screen refresh -- default is 0
                           milli-seconds)
 -D DMA Engine            (DMA Engine number -- default = 1)
 -E <ExpInpVolts>@<Tol>   (Expected Input Volts@Tolerance -- default Tol=0.003000)
    +@<Tol>               (Positive Calibration Ref Volt@Tolerance)
    -@<Tol>               (Negative Calibration Ref Volt@Tolerance)
    s@<Tol>               (Requires '-s' input signal option to specify voltage
                           Volt@Tolerance)
                          (valid '-s' arguments are 'g','+','-','f','t')
```

```
 -f DataFormat           (select data format, '2' or 'b')
    -f b,2               (Ch0..7=Offset binary, Ch8..15=Two's complement)
    -f 2/b               (Ch0..7=Two's complement, Ch8..15=Offset binary)
    -f b                 (Ch0..15=Offset binary)
 -F DebugFile            (Debug file with rate display -- default "=== None ===")
    @DebugFile           (Debug file without rate display -- default "=== None ===")
    @                    (No debug file and no rate display -- default "=== None ===")
 -i                      (Enable Interrupts -- default = Disable)
 -l LoopCnt              (Loop count -- default is 0)
 -m XferMode             (Transfer Mode -- default = Library DMA or MsgDma if
                          supported)
    -mdP                 (Driver:  (FIFO) PIO mode)
    -mlD                 (Library: (FIFO) DMA mode)
    -mlP                 (Library: (FIFO) PIO mode)
 -N                      (Open device with O_NONBLOCK flag for driver operations)
 -s InputSignal          (select input signal, 'e', 'g', '+', '-', 't', 'f',
                          '0..15')
    -s e,g               (Ch0..7=External input, Ch8..15=ground calibration)
    -s +/e               (Ch0..7=Postive calibration, Ch8..15=external reference)
    -s -                 (Ch0..15=Negative calibration)
    -s t                 (Ch0..15=2.5 volt calibration)
    -s e/f               (Ch0..7=external reference, Ch8..15=5 volt calibration)
    -s e/12              (Ch0..7=external reference, Ch8..15=DAC Channel 12)
 -S NumberOfSamples      (Number of Samples -- default is 49152)
 -T TestBus              (Test Bus Control 'b' or 'o'. Exit after programming
                          this option)
    -T b                 (Calibration Bus Control)
    -T o                 (Open Bus Control)
 -V MaxBoardVolts        (Voltage range 'b5' or 'b10')
    -V b5,b10            (Ch0..7=5V, Ch8..15=10V)
    -Vb10/b5             (Ch0..7=10V, Ch8..15=5V)
    -V b10               (Ch0..15=10V)

 e.g. ./ccurpmfc_adc_fifo -C0,1@100000 -se/+ (ADC0=300000Hz external input,
                                              ADC1=100000Hz Positive Cal.)
      ./ccurpmfc_adc_fifo -C0,1@100000 -Vb10 -s- -Es
                                             (Max Clock, -9.91V input,
                                              validate against -9.91V)
      ./ccurpmfc_adc_fifo -C0,1@100000 -Vb10 -st -Es
                                             (Max Clock, +2.5V input,
                                              validate against +2.5V)
```

Example display:

./ccurpmfc_adc_fifo -C0@300000,1@100000 -se/+ -l1000

```
local_ptr=0x7ffff7f98000
           Number of Samples =49152
           Transfer Mode     =Library DMA Mode
           Physical Memory Information:
             UserPID         =19842
             PhysMemPtr      =0x43500000
             DriverVirtMemPtr=0xffff979dc3500000
             MmapedUserMemPtr=0x7ffff7f18000
             PhysMemSize     =0x00080000
             PhysMemSizeFreed=0x00000000
             EntryInTxTbl    =0
             NumOfEntriesUsed=1
             Flags           =0x0000
   NumOfChannels=16, FirstChannel=0, LastChannel=15, NumAdc0Chans=8
   NumAdc1Chans=8
   Time in microseconds (TT=Total, WT=Work, FT=Free, RT=Read, mi=min, ma=max,
```

```
                          av=ave)
    Measuring how long it takes to collect 49152 samples...done. (15366.380
                          usecs)
   1000: TT=15360.68 WT=11899.23 FT=3461.45 RT=11097.57
         (min=11095.73/max=11163.35/ave=11097.25) 17.72 MBytes/Sec -
          EmptyCnt=92944 (71%)
============================================================
                   Date: Mon Oct 11 16:57:20 2021
   Expected Input Volts: === Not Specified ===
            Scan Counter: ADC0=4608000 ADC1=1536000
  Approx. Sample/Second: ADC0=299998 ADC1=99999
           NumberOfChans: ADC0=8 ADC1=8
   WorstMinChanVoltsHWM:  -0.002441 (Ch00)
   WorstMaxChanVoltsHWM:   9.936218 (Ch10)
============================================================
      <-------- (volts) -------->
Chan  Min       Max       Ave      DetectedCnt TolerExeededCnt
====  =======   =======   =======  =========== ===============
 00   -0.0024   -0.0003   -0.0010      4608000       -
 01   -0.0012    0.0012    0.0001      4608000       -
 02   -0.0018    0.0006   -0.0004      4608000       -
 03   -0.0012    0.0009   -0.0000      4608000       -
 04   -0.0012    0.0006   -0.0000      4608000       -
 05   -0.0018    0.0003   -0.0005      4608000       -
 06   -0.0012    0.0009    0.0003      4608000       -
 07   -0.0015    0.0006   -0.0003      4608000       -
 08    9.9133    9.9179    9.9156      1536000       -
 09    9.9139    9.9185    9.9162      1536000       -
 10    9.9316    9.9362    9.9339      1536000       -
 11    9.9271    9.9313    9.9293      1536000       -
 12    9.9158    9.9203    9.9180      1536000       -
 13    9.9179    9.9225    9.9200      1536000       -
 14    9.9185    9.9231    9.9209      1536000       -
 15    9.9240    9.9286    9.9264      1536000       -
============================================================

Below are the statistics for 49152 samples:
   Estimated time to collect samples:     15368.294 usecs
   Total work time breakdown:             15371.225 usecs
      Average time to fill FIFO:           3407.085 usecs (### User blocking for FIFO to
fill  ###)
      Average time to read samples:       11097.822 usecs (### This time excludes FIFO
fill time ###)
      Average time to process samples:      851.790 usecs
      Average time other:                    14.528 usecs
   Approximate free time available:        3418.682 usecs
```

3.2.4 lib/ccurpmfc_adc_sps

This is a useful tool to display the sample rate of various channels.

```
Usage: ./ccurpmfc_adc_sps [-b Board] [-c StartChan,StopChan] [-C AdcUpdateClock]
                          [-E ExpSPS@Tol] [-l LoopCnt] [-t TolerancePPT]
 -b Board                (Board number -- default is 0)
 -c StartChan,EndChan    (Select start and end channel numners -- default 0,15
    -c 4,13              (select channels 4 through 13 for processing
    -c 7                 (select channels 7 through 15 for processing
 -C AdcUpdateClock       (select ADC update clock, 0..6 or 'n|N')
    -C 0,6               (Ch0..7=Clock0, Ch8..15=Clock6 at MAX SPS)
    -C 6@20000.0/n       (Ch0..7=Clock6 at 20000 SPS, Ch8..15=No Clock)
    -C 4                 (Ch0..15=Clock4 at MAX SPS)
```

```
   -C 4@150000.0        (Ch0..15=Clock4) at 150000 SPS
 -E ExpSPS@Tol          (specify expected samples/second and tolerance for each
                         ADC)
   -E C                 (All ADC's to use clock samples/second and default
                         tolerance 0.010%)
   -E c@0.02,30000      (ADC 0 uses clock samples/second and tolerance 0.02%,
                         remaining use 30,000 SPS and default tolerance 0.010%)
   -E C@0.02,C          (All ADC's to use clock samples/second and default
                         tolerance except for ADC 0 tolerance of 0.02%)
   -E 10000,c           (ADC 0 to use 10000 SPS, rest of ADCs to use clock
                         samples/second. Default tolerance for all ADCs)
 -F DebugFile           (Debug file with menu display -- default "=== None ===")
   @DebugFile           (Debug file without menu display (only summary and rate
                         display) -- default "=== None ===")
   @                    (No debug file and no menu display (only summary and
                         rate display) -- default "=== None ===")
 -l LoopCnt             (Loop Count -- default is 10000000)
 -l 0                   (Loop forever)
 -t <TolerancePPT>      (Tolerance in Parts/Trillion -- default is 0.007000 PPT)

 e.g. ./ccurpmfc_adc_sps -C0@123456,1@78912 (ADC0 is 123456Hz, ADC1 is 78912Hz)
```

Example display:

./ccurpmfc_adc_sps -C0@123456,1@78912

```
local_ptr=0x7ffff7fd7000
           Physical Memory Information:
             UserPID         =26726
             PhysMemPtr      =0x4910000
             DriverVirtMemPtr=0xffff880004910000
             MmapedUserMemPtr=0x7ffff7fb0000
             PhysMemSize     =0x00010000
             PhysMemSizeFreed=0x00000000
             EntryInTxTbl    =0
             NumOfEntriesUsed=1
             Flags           =0x0000

Read: Size 65536, Count 8 (FIFO wait: 6425.7us, Read time/rate: 3701.4us/17.7MBPS)

          ============================== Samples/Second ==============================
          [0]       [1]       [2]       [3]       [4]       [5]       [6]       [7]
          ======    ======    ======    ======    ======    ======    ======    ======
 [00 07]  123455    123455    123455    123455    123455    123455    123455    123455
 [08 15]  078912    078912    078912    078912    078912    078912    078912    078912




================================
       <-- (Samples/Second) -->
Chan   Min      Max      Ave
====   ======   ======   ======
  0    123452   123462   123458
  1    123452   123462   123458
  2    123452   123462   123458
  3    123452   123462   123458
  4    123452   123462   123458
  5    123452   123462   123458
  6    123452   123462   123458
```

```
 7    123452   123462   123458
 8    078910   078916   078913
 9    078910   078916   078913
10    078910   078916   078913
11    078910   078916   078913
12    078910   078916   078913
13    078910   078916   078913
14    078910   078916   078913
15    078910   078916   078913
```

3.2.5 lib/ccurpmfc_check_bus

This is a simple test to check whether there is intereference from other cards that may be sharing the same bus. It simply computes the time it takes to perform hardware reads and computes the jitter. It must be run as *root*.

```
Usage: ./ccurpmfc_check_bus [-b Board] [-c CPU] [-l LoopCnt] [-t Tolerance]
 -b Board     (Board number -- default is 0)
 -c CPU       (CPU number -- default is 1)
 -l LoopCnt   (Loop Count -- default is 10000000)
 -l 0         (Loop forever)
 -t Tolernace (Tolerance -- default is 2.00 micro-seconds)
```

Example display:

sudo ./ccurpmfc_check_bus

```
local_ptr=0x7ffff7fd7000
10000000: usec/read: Cur=1.181 (Min=1.159 Max=1.794 Ave= 1.181335)
          [Bus Jitter (usec): 0.635 ===> LOW]
```

3.2.6 lib/ccurpmfc_clock

This is a useful tool to display information of the various clocks and also program them.

```
Usage: ./ccurpmfc_clock [-b BoardNo] [-C UpdateClock] [-d Delay] [-l LoopCnt]
                       [-R] [-t TolerancePPT]
 -b BoardNo              (Board number -- default is 0)
 -C <Clock>@<Frequency>  (set update clock '0..6' with frequency )
 -d Delay                (Delay between screen refresh -- default is 10 milli-
                          seconds)
 -l LoopCnt              (Loop count -- default is 0)
 -R                      (Reset/Clear all clocks)
 -t                      (Tolerance in Parts/Trillion -- default is 0.007000 PPT)

 e.g. ./ccurpmfc_clock -C 1@300000
                         (Set Clock 1 to 300000 SPS - do not change any other
                          running clocks)
      ./ccurpmfc_clock -R -C0@100000 -C4@12345 -t0.5
                         (Reset all clocks and then set Clock 0 to 100000 SPS and
                          Clock 4 to 12345 SPS and 0.5 PPT)
```

Example display:

./ccurpmfc_clock -R -C0@100000 -C4@12345

```
                  Board Number [-b]: 0
                        Delay [-d]: 10 milli-seconds
                    Loop Count [-l]: ***Forever***
                        Scan Count: 1258
```

```
                    _________ Clock Revision ________
                     Silicon Revision: A1
                     Base Part Number: 5341
                   Device Speed Grade: A
                      Device Revision: A

                    _________ Clock CSR _________
                      Clock Interface: Idle
                         Clock Output: Enabled
                          Clock State: Active

                    _________ Input Clock Status _________
                          Calibration: Not In-Progress
                        SMBUS Timeout: Not Timed Out
                             PLL Lock: Locked
                         Input Signal: Present
                        Input_0 Clock: Present
                        Input_1 Clock: *** Not Present ***
                        Input_2 Clock: *** Not Present ***
                       Input_FB Clock: Present
                      XAXB Input Clock: *** Not Present ***

          _____________________ Output Clock Setting _____________________
          User output clock frequency 0:   100000.000 Samples/Second/Channel
          User output clock frequency 1: *** Not Set ***
          User output clock frequency 2: *** Not Set ***
          User output clock frequency 3: *** Not Set ***
          User output clock frequency 4:    12345.000 Samples/Second/Channel
          User output clock frequency 5: *** Not Set ***
          User output clock frequency 6: *** Not Set ***
        SD-RAM output clock frequency 7: 10000000.000 Samples/Second/Channel
       External output clock frequency 8: 10000000.000 Samples/Second/Channel
      Feed-Back output clock frequency 9: 10000000.000 Samples/Second/Channel
      Feed-Back output clock frequency 9: 10000000.000 Samples/Second/Channel
```

3.2.7 lib/ccurpmfc_dac

This test is useful in programming the DAC interface and displaying the DAC registers.

```
Usage: ./ccurpmfc_dac [-A] [-a RollingAve] [-b BoardNo] [-C AdcUpdateClock]
                      [-d Delay] [-D DMAEngine] [-E ExpInpVolt] [-f DataFormat]
                      [-F DebugFile] [-l LoopCnt] [-n NumChans] [-o OutputSelect]
                      [-s InputSignal] [-v DacVoltage] [-V OutputRange] [-Z]
 -A                      (Perform DAC Auto Calibration first using reference
                          voltage)
 -a RollingAve           (Rolling average -- default "=== None ===")
 -b BoardNo              (Board number -- default is 0)
 -C AdcUpdateClock       (select ADC update clock, 0..6 or 'n')
    -C 0,6               (Ch0..7=Clock0, Ch8..15=Clock6 at MAX SPS)
    -C 6@20000.0/n       (Ch0..7=Clock6 at 20000 SPS, Ch8..15=No Clock)
    -C 4                 (Ch0..15=Clock4 at MAX SPS)
    -C 4@150000.0        (Ch0..15=Clock4) at 150000 SPS
 -d Delay                (Delay between screen refresh -- default is 0 milli-
                          seconds)
 -D DMA Engine           (DMA Engine number -- default = 1)
 -E <ExpInpVolts>@<Tol>  (Expected Input Volts@Tolerance -- default Tol=0.006000)
    +@<Tol>              (Positive Calibration Ref Volt@Tolerance)
    -@<Tol>              (Negative Calibration Ref Volt@Tolerance)
    c@<Tol>              (DAC Channel 0 Volt@Tolerance)
    s@<Tol>              (Requires '-s' input signal option to specify voltage
                          Volt@Tolerance)
```

```
                        (valid '-s' arguments are 'g','+','-','f','t')
   Note:                (For differential bipolar, even channels, voltage read
                         is half supplied)
                        (For differential bipolar, odd channels, voltage read is
                         neg. half supplied)
-f DataFormat           (select DAC data format, '2' or 'b')
   -f b,2               (Ch0..3=Offset binary, Ch4..15=Two's complement)
   -f 2/b,2             (Ch0..3 & Ch8..15=Two's complement, Ch4..7=Offset
                         binary)
   -f b                 (Ch0..15=Offset binary)
-F DebugFile            (Debug file with menu display -- default "=== None ===")
   #DebugFile           (Debug file without display (only summary) -- default
                         "=== None ===")
   @DebugFile           (Debug file without display -- default "=== None ===")
   @ or #               (No debug file and no display -- default "=== None ===")
-l LoopCnt              (Loop count -- default is 0)
-n NumChans             (Number of channels (1..16) -- default is 16)
-o OutputSelect         (DAC output select, 's' or 'd')
   -o d,s               (Ch0..3=differential, Ch4..15=single_ended)
   -o s/d,s             (Ch0..3 & Ch8..15=single_ended, Ch4..7=differential)
   -o d                 (Ch0..15=differential)
-s InputSignal          (ADC select input signal, 'a', 'e', 'g', '+', '-', 't',
                         'f', '0..15')
   -s a,e               (Ch0..7=All DAC Channels 0..7, Ch8..15=External ADC
                         Input)
   -s e,g               (Ch0..7=External ADC input, Ch8..15=ground calibration)
   -s +/e               (Ch0..7=Postive calibration, Ch8..15=External ADC input)
   -s -                 (Ch0..15=Negative calibration)
   -s t                 (Ch0..15=2.5 volt calibration)
   -s e/f               (Ch0..7=External ADC input, Ch8..15=5 volt calibration)
   -s e/12              (Ch0..7=External ADC input, Ch8..15=DAC Channel 12)
-v DacVoltage           (DAC Voltage. -10.0  to +20.0)
   -v 1.5,9.9           (Ch0..3=1.5 volts, Ch4..15= 9.9 volts)
   -v2.5/7.5,12.7       (Ch0..3=2.5 volts, 4..7=7.5 volts, 8..15=12.7 volts)
   -v 9.95              (Ch0..15=9.95 volts)
-V OutputRange          (Output Voltage range. u10, u20, b5, b10, b20)
                           'u10'  - Unipolar 10 volts (  +0 --> +10 )
                            single_ended/differential
                           'u20'  - Unipolar 20 volts (  +0 --> +20 )
                            single_ended/differential
                           'b5'   - Bipolar 5 volts   (  -5 --> +5  )
                            single_ended
                           'b10'  - Bipolar 10 volts  ( -10 --> +10 )
                            single_ended/differential
                           'b20'  - Bipolar 20 volts  ( -20 --> +20 )
                            differential
   -V u10,b10           (Ch0..3=UniPolar 10V, Ch4..15=BiPolar 10V)
   -Vb5/b10,u20         (Ch0..3=BiPolar 5V, 4..7=BiPolar 10V, 8..15=UniPolar
                         20V)
   -V b10               (Ch0..15=BiPolar 10V)
   -V b20               (Ch0..15=BiPolar 20V - differential)
-Z                      (Display Calibration Offset & Gain Channels)

e.g. ./ccurpmfc_dac -os -s7 -Vb5 -v4.5 -E4.5      (Internal Loopback Testing.
                                                   Generate 4.5V and compare)
     ./ccurpmfc_dac -os -se -Vb5 -v4.5 -E4.5      (External DAC/ADC Loopback
                                                   Testing. Generate 4.5V and
                                                   compare)
     ./ccurpmfc_dac -od -s2 -Vb5 -v5.0 -E2.5      (Internal Loopback Testing.
                                                   Generate 5.0V and compare
                                                   diff 2.5V)
```

```
                ./ccurpmfc_dac -od -s3 -Vb5 -v5.0 -E-2.5     (Internal Loopback Testing.
                                                              Generate 5.0V and compare
                                                              diff -2.5V)
                ./ccurpmfc_dac -os -sa -Vb5 -v1,2,3,4 -a100  (display all DAC 0..15
                                                              channels with rolling average
                                                              of 100)
                ./ccurpmfc_dac -os -sa -Vb5 -v3.5 -E3.5@0.01 (Internal Loopback Testing.
                                                              Generate 3.5V and compare
                                                              diff on all chans)
                ./ccurpmfc_dac -C0 -Vb10 -s- -Es             (Max Clock, -9.91V input,
                                                              validate against -9.91V)
                ./ccurpmfc_dac -C0 -Vb10 -st -Es             (Max Clock, +2.5V input,
                                                              validate against +2.5V)
```

Example display:

./ccurpmfc_dac -A -os -s7 -Vb5 -v4.5 -E4.5

```
local_ptr=0x7ffff7fd7000
           Physical Memory Information:
             UserPID         =27364
             PhysMemPtr      =0x1c2000
             DriverVirtMemPtr=0xffff8800001c2000
             MmapedUserMemPtr=0x7ffff7fcc000
             PhysMemSize     =0x00001000
             PhysMemSizeFreed=0x00000000
             EntryInTxTbl    =0
             NumOfEntriesUsed=1
             Flags           =0x0000
Auto Calibration started...done. (2.574 seconds)

 Board Number          [-b]: 0
 Update Clock Selected [-C]: Ch00..07 OutputClock=0 (0x7)
                           : Ch08..15 OutputClock=0 (0x7)
 Delay                 [-d]: 0 milli-seconds
 DMA Engine            [-D]: 1
 Expected Input Volts  [-E]: 4.500000 volts (Tolerance 0.006000 volts)
 DAC Data Format       [-f]: Ch00..03=Obin Ch04..07=Obin Ch08..11=Obin Ch12..15=Obin
 DAC Output Select     [-o]: Ch00..03=Sngl Ch04..07=Sngl Ch08..11=Sngl Ch12..15=Sngl
 ADC Input Signal      [-s]: Ch00..07 [1]Calibration Input (0x27: DAC Channel 7)
                           : Ch08..15 [1]Calibration Input (0x27: DAC Channel 7)
 DAC Voltage           [-v]: Ch00..03=4.50 Ch04..07=4.50 Ch08..11=4.50 Ch12..15=4.50
 DAC Voltage Range     [-V]: Ch00..03=b5 Ch04..07=b5 Ch08..11=b5 Ch12..15=b5

 Loop Count            [-l]: ***Forever***
 Number of Channels    [-n]: 16
 Scan Count                : 35843           (0:00:01:17)

 Tolerance Exceeded Count  : 0 (=== Passed ===)
 Read Duration (microsecs) :    20.739 (min=  20.582/max=  34.028/ave=  21.044)

          ##### Raw Data (DAC Channels) #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       ====     ====     ====     ====     ====     ====     ====     ====     ====     ====
 [0]   f333     f333     f333     f333     f333     f333     f333     f333     f333     f333
 [1]   f333     f333     f333     f333     f333     f333

          ##### Volts (DAC Channels) #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       =======  =======  =======  =======  =======  =======  =======  =======  =======  =======
 [0]   +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000
```

```
 [1]   +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000
-------------------------------------------------------------------------------------------------------

         ##### Raw Data (ADC Readback - 4.50v) #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       ====     ====     ====     ====     ====     ====     ====     ====     ====     ====
 0]    --       --       --       --       --       --       --      7ffd      --       --
 1]    --       --       --       --       --       --

         ##### Volts (ADC Readback - 4.50v) #####
         [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       =======  =======  =======  =======  =======  =======  =======  =======  =======  =======
 [0]     ---      ---      ---      ---      ---      ---      ---    -0.0005    ---      ---
 [1]     ---      ---      ---      ---      ---      ---

==================================================
                  Date: Mon Oct 25 14:40:18 2017
   Expected Input Volts: 4.500000 volts (Tolerance 0.006000 volts)
 Tolerence Exceed Count: 0
           Scan Counter: 827984
   WorstMinChanVoltsHWM:  -0.003695 (Ch07)
   WorstMaxChanVoltsHWM:   0.002013 (Ch07)
==================================================
      <-------- (volts) -------->
Chan  Min       Max       Ave      TolerExeededCnt
====  =======   =======   =======  ===============
 07   -0.0037    0.0020   -0.0008         -
==================================================
```

./ccurpmfc_dac -C0 -Z

```
local_ptr=0x7ffff7fd7000
          Physical Memory Information:
            UserPID         =27381
            PhysMemPtr      =0x86439000
            DriverVirtMemPtr=0xffff880086439000
            MmapedUserMemPtr=0x7ffff7fcc000
            PhysMemSize     =0x00001000
            PhysMemSizeFreed=0x00000000
            EntryInTxTbl    =0
            NumOfEntriesUsed=1
            Flags           =0x0000

 Board Number          [-b]: 0
 Update Clock Selected [-C]: Ch00..07 OutputClock=0 (0x7) (300000.000 SPS)
                           : Ch08..15 OutputClock=0 (0x7) (300000.000 SPS)
 Delay                 [-d]: 0 milli-seconds
 DMA Engine            [-D]: 1
 Expected Input Volts  [-E]: === Not Specified ===
 DAC Data Format       [-f]: Ch00..03=Obin Ch04..07=Obin Ch08..11=Obin Ch12..15=Obin
 DAC Output Select     [-o]: Ch00..03=Sngl Ch04..07=Sngl Ch08..11=Sngl Ch12..15=Sngl
 ADC Input Signal      [-s]: Ch00..07 [1]Calibration Input (0x20: DAC Channel 0)
                           : Ch08..15 [1]Calibration Input (0x20: DAC Channel 0)
 DAC Voltage           [-v]: Ch00..03=99.00 Ch04..07=99.00 Ch08..11=99.00 Ch12..15=99.00
 DAC Voltage Range     [-V]: Ch00..03=b5 Ch04..07=b5 Ch08..11=b5 Ch12..15=b5

 Loop Count            [-l]: ***Forever***
 Number of Channels    [-n]: 16
 Scan Count                : 36306           (0:00:11:04)
 Read Duration (microsecs) :    20.867 (min=  20.605/max=  35.622/ave=  21.058)
```

```
         ##### Raw Data (Offset Calibration DAC Channels) #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       ====     ====     ====     ====     ====     ====     ====     ====     ====     ====
 [0]   0000     0000     0000     0000     0000     0000     0000     0000     0000     0000
 [1]   0000     0000     0000     0000     0000     0000

         ##### Volts (Offset Calibration DAC Channels) #####
       [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]     [8]     [9]
       =======  =======  =======  =======  =======  =======  =======  =======  =======  =======
 [0]   +0.0000  +0.0000  +0.0000  +0.0000  +0.0000  +0.0000  +0.0000  +0.0000  +0.0000  +0.0000
 [1]   +0.0000  +0.0000  +0.0000  +0.0000  +0.0000  +0.0000
-------------------------------------------------------------------------------------------------
         ##### Raw Data (Gain Calibration DAC Channels) #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       ====     ====     ====     ====     ====     ====     ====     ====     ====     ====
 [0]   0012     0011     0008     0008     0009     000a     0014     0014     000a     000e
 [1]   000f     000f     0010     000e     0012     0011

         ##### Volts (Gain Calibration DAC Channels) #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       =======  =======  =======  =======  =======  =======  =======  =======  =======  =======
 [0]   +0.0014  +0.0013  +0.0006  +0.0006  +0.0007  +0.0008  +0.0015  +0.0015  +0.0008  +0.0011
 [1]   +0.0011  +0.0011  +0.0012  +0.0011  +0.0014  +0.0013
-------------------------------------------------------------------------------------------------
         ##### Raw Data (DAC Channels) #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       ====     ====     ====     ====     ====     ====     ====     ====     ====     ====
 [0]   f333     f333     f333     f333     f333     f333     f333     f333     f333     f333
 [1]   f333     f333     f333     f333     f333     f333

         ##### Volts (DAC Channels) #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       =======  =======  =======  =======  =======  =======  =======  =======  =======  =======
 [0]   +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000
 [1]   +4.5000  +4.5000  +4.5000  +4.5000  +4.5000  +4.5000
-------------------------------------------------------------------------------------------------
         ##### Raw Data (ADC Readback) #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       ====     ====     ====     ====     ====     ====     ====     ====     ====     ====
 [0]   f328      --       --       --       --       --       --       --       --       --
 [1]    --       --       --       --       --       --

         ##### Volts (ADC Readback) #####
       [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]      [9]
       =======  =======  =======  =======  =======  =======  =======  =======  =======  =======
 [0]   +4.4983    ---      ---      ---      ---      ---      ---      ---      ---      ---
 [1]     ---      ---      ---      ---      ---      ---

==================================================
                  Date: Mon Oct 25 14:42:38 2017
   Expected Input Volts: === Not Specified ===
           Scan Counter: 223036
   WorstMinChanVoltsHWM:   4.496932 (Ch00)
   WorstMaxChanVoltsHWM:   4.502127 (Ch00)
==================================================
      <-------- (volts) -------->
Chan  Min       Max       Ave      TolerExeededCnt
====  =======   =======   =======  ===============
 00    4.4969    4.5021    4.4997         -
==================================================
```

3.2.8 lib/ccurpmfc_dac_calibrate

This test is useful for performing, saving and restoring DAC calibration. If calibration '-A' is specified along with voltage '-V', the board voltage range will first be programmed prior to initiaiing calibration.

```
Usage: ./ccurpmfc_dac_calibrate [-A] [-b board] [-c ChanMask] [-f DataFormat]
                                [-i inCalFile] [-o outCalFile] [-R]
                                [-V MaxBoardVolts]
 -A                    (perform Auto Calibration)
 -b <board>            (board #, default = 0)
 -c <ChanMask>         (channel selection mask, default = all channels)
 -f DataFormat         (select DAC data format, '2' or 'b')
    -f b,2             (Ch0..3=Offset binary, Ch4..15=Two's complement)
    -f 2/b,2           (Ch0..3 & Ch8..15=Two's complement, Ch4..7=Offset binary)
    -f b               (Ch0..15=Offset binary)
 -i <In Cal File>      (input calibration file [input->board_reg])
 -o <Out Cal File>     (output calibration file [board_reg->output])
 -R                    (reset DAC calibration)
 -V MaxBoardVolts      (Voltage range)
    -V u10,b10         (Ch0..3=UniPolar 10V, Ch4..15=BiPolar 10V)
    -Vb5/b10,u20       (Ch0..3=BiPolar 5V, 4..7=BiPolar 10V, 8..15=UniPolar 20V)
    -V b10             (Ch0..15=BiPolar 10V)

 e.g. ./ccurpmfc_dac_calibrate              (Dump calibration information to
                                             stdout)
      ./ccurpmfc_dac_calibrate -A -o Calfile (Perform Auto calibration and dump
                                             information to 'Calfile')
      ./ccurpmfc_dac_calibrate -i Calfile    (Update board calibration with
                                             supplied 'Calfile')
      ./ccurpmfc_dac_calibrate -R            (Reset DAC calibration)
```

Example display:

./ccurpmfc_dac_calibrate -A -oOutputCal

```
Device Name    : /dev/ccurpmfc0
Board Serial No: 674459 (0x000a4a9b)
Auto Calibration started...done. (2.572 seconds)

===> Dump of 'OutputCal' file
#Date           : Wed Oct  2 13:43:12 2019

#Chan   Gain                    Offset
#====   ====                    ======
 ch00:  0.0027465820312500  0.0000762939453125
 ch01:  0.0027465820312500  0.0000953674316406
 ch02:  0.0024414062500000  0.0000190734863281
 ch03:  0.0025939941406250  0.0000572204589844
 ch04:  0.0025939941406250  0.0001335144042969
 ch05:  0.0027465820312500  0.0000572204589844
 ch06:  0.0030517578125000  0.0000762939453125
 ch07:  0.0030517578125000  0.0000572204589844
 ch08:  0.0025939941406250  0.0001144409179688
 ch09:  0.0024414062500000  0.0000953674316406
 ch10:  0.0027465820312500  0.0000953674316406
 ch11:  0.0025939941406250  0.0000190734863281
 ch12:  0.0022888183593750  0.0001144409179688
 ch13:  0.0022888183593750  0.0001144409179688
 ch14:  0.0035095214843750  0.0000953674316406
 ch15:  0.0032043457031250  0.0000953674316406
```

```
===> Board calibration data written to 'OutputCal' file
```

3.2.9 lib/ccurpmfc_dac_setchan

This test generates voltages on various Analog Output channels.

```
Usage: ./ccurpmfc_dac_setchan [-b board] [-c ChannelSelectMask]
                              [-C DacUpdateClock] [-D DMAEngine][-f format] [-i]
                              [-l LoopCnt] [-m WriteMode] [-N] [-o OutputSelect]
                              [-S NumSamples] [-u UpdateMode] [-v OutputVolts]
                              [-V OutputRange] [-w WaveType]
 -A                       (Perform DAC Auto Calibration first using reference
                           voltage)
 -b <board>               (board #, default = 0)
 -c <ChannelSelectMask>   (channel selection mask, default = all channels)
 -C DacUpdateClock        (select DAC update clock, 0..6 or 's|S')
    -C s                  (Ch0..15=Software Update)
    -C 6@20000.0          (Ch0..15=Clock6 at 20000 SPS
    -C 4                  (Ch0..15=Clock4 at MAX SPS)
 -D DMA Engine            (DMA Engine number -- default = 0)
 -f DataFormat            (select DAC data format, '2' or 'b')
    -f b,2                (Ch0..3=Offset binary, Ch4..15=Two's complement)
    -f 2/b,2              (Ch0..3 & Ch8..15=Two's complement, Ch4..7=Offset
                           binary)
    -f b                  (Ch0..15=Offset binary)
 -i                       (Enable Interrupts -- default = Disable)
 -l LoopCnt               (Loop count -- default is 0)
 -m <WriteMode>           (Write Mode)
    -mdp                     (Driver:  (Channel Registers) PIO mode)
    -mlc                     (Library: (Channel Registers) program I/O Fast Memory
                              Copy)
    -mld                     (Library: (Channel Registers) DMA mode)
    -mlp                     (Library: (Channel Registers) PIO mode
    -mup                     (User:    (Channel Registers) PIO mode)
    -mdP                     (Driver:  (FIFO) PIO mode)
    -mlD                     (Library: (FIFO) DMA mode)
    -mlP                     (Library: (FIFO) PIO mode)
 -N                       (Open device with O_NONBLOCK flag for driver operations)
 -o OutputSelect          (DAC output select, 's' or 'd')
    -o d,s                (Ch0..3=differential, Ch4..15=single_ended)
    -o s/d,s              (Ch0..3 & Ch8..15=single_ended, Ch4..7=differential)
    -o d                  (Ch0..15=differential)
 -S <NumSamples>          (Number of Samples per channel, default = 512)
 -u                       (Set DAC Update Mode)
    -ui                      (Set DAC Update Mode to Immediate Mode)
    -us                      (Set DAC Update Mode to Synchronized Mode)
 -v DacVoltage            (DAC Voltage. -10.0  to +20.0)
    -v 1.5,9.9            (Ch0..3=1.5 volts, Ch4..15= 9.9 volts)
    -v2.5/7.5,12.7        (Ch0..3=2.5 volts, 4..7=7.5 volts, 8..15=12.7 volts)
    -v 9.95               (Ch0..15=9.95 volts)
 -V OutputRange           (Output Voltage range. u10, u20, b5, b10, b20)
                             'u10'  - Unipolar 10 volts (  +0 --> +10 )
                              single_ended/differential
                             'u20'  - Unipolar 20 volts (  +0 --> +20 )
                              single_ended/differential
                             'b5'   - Bipolar 5 volts   (  -5 --> +5  )
                              single_ended
                             'b10'  - Bipolar 10 volts  ( -10 --> +10 )
                              single_ended/differential
                             'b20'  - Bipolar 20 volts  ( -20 --> +20 )
                              differential
```

```
   -V u10,b10           (Ch0..3=UniPolar 10V, Ch4..15=BiPolar 10V)
   -Vb5/b10,u20         (Ch0..3=BiPolar 5V, 4..7=BiPolar 10V, 8..15=UniPolar
                         20V)
   -V b10               (Ch0..15=BiPolar 10V)
   -V b20               (Ch0..15=BiPolar 20V - differential)
-w <WaveType>           (default  = 'c' Constant Voltage)
   -wc                     (Constant Voltage
   -wu                     (Saw Wave (up))
   -wd                     (Saw Wave (down))
   -ws                     (Sine Wave)
   -wx                     (Square Wave)
   -wX                     (Square Wave - Alternate Sample)
   -wy                     (Step Wave (down))
   -wz                     (Step Wave (up))
   -wt                     (Triangle Wave)
   -ww                     (All Wave (Sine/Square/StepUp/Triangle/StepDown))

e.g. ./ccurpmfc_dac_setchan -od,s,d,s -v20,10,5,1 -wx -mdP -Vb20 (dac0&2
                                          differential, dac1&3 single_ended)
     ./ccurpmfc_dac_setchan -ws -ui -od                        (sine wave,
                                                                immediate)
```

Example display:

./ccurpmfc_dac_setchan -ws -ui -od

```
Device Name    : /dev/ccurpmfc0
          Physical Memory Information:
            UserPID         =29005
            PhysMemPtr      =0x86b10000
            DriverVirtMemPtr=0xffff880086b10000
            MmapedUserMemPtr=0x7ffff7fbc000
            PhysMemSize     =0x00008000
            PhysMemSizeFreed=0x00000000
            EntryInTxTbl    =0
            NumOfEntriesUsed=1
            Flags           =0x0000
DAC 0....
           State = 0x0 (Idle)
      Power Down = 0x0 (Operational)
     Update Mode = 0x0 (Immediate)
     Data Format = 0x0 (Offset Binary)
   Output Select = 0x1 (Differential)
    Output Range = 0x1 (Unipolar 10 Volts - Differential)
DAC 1....
           State = 0x0 (Idle)
      Power Down = 0x0 (Operational)
     Update Mode = 0x0 (Immediate)
     Data Format = 0x0 (Offset Binary)
   Output Select = 0x1 (Differential)
    Output Range = 0x1 (Unipolar 10 Volts - Differential)
DAC 2....
           State = 0x0 (Idle)
      Power Down = 0x0 (Operational)
     Update Mode = 0x0 (Immediate)
     Data Format = 0x0 (Offset Binary)
   Output Select = 0x1 (Differential)
    Output Range = 0x1 (Unipolar 10 Volts - Differential)
DAC 3....
           State = 0x0 (Idle)
      Power Down = 0x0 (Operational)
```

```
      Update Mode = 0x0 (Immediate)
      Data Format = 0x0 (Offset Binary)
    Output Select = 0x1 (Differential)
     Output Range = 0x1 (Unipolar 10 Volts - Differential)

Write Mode: -mdp: Driver: (CHANNEL) PIO Mode
Generating a continuous Sine Wave on selected channels: <CTRL-C> to abort
Voltage Selection: 10.00/10.00/10.00/10.00, Channel Mask Selection: 0xffff,
Samples/Write=512
9.587 usec/write: 5.093 msec period, 196.343 Hz
```

3.2.10 lib/ccurpmfc_dio

This test generates, views and tests various digital channels.

```
Usage: ./ccurpmfc_dio [-b BoardNo] [-d Delay] [-F DebugFile] [-l LoopCnt]
                      [-m DIOMode] [-n NumChans] [-p PatternSelect]
                      [-r RunOption] [-s SkipChannelsMask]
 -b BoardNo               (Board number -- default is 0)
 -d Delay                 (Delay between screen refresh -- default is 100)
 -F DebugFile             (Debug file -- default "=== None ===")
 -l LoopCnt               (Loop count -- default is 0)
 -m DIOMode               (DIO mode -- default is 1)
    -m0                   (DIO Custom mode)
    -m1                   (DIO Normal mode)
 -n NumChans              (number of channels -- default is 96)
 -p PatternSelect         (DIO mode -- default is to sequence through all
                           patterns)
    -p0                   (Rolling Ones)
    -p1                   (Rolling Zeros)
    -p2                   (Adding Bit)
    -p3                   (Toggling 'A' & '5')
    -p@XXXXXXXX           (Fixed Pattern XXXXXXXX selection in Hex)
 -r RunOption             (Run option -- default is 0)
    -rd                   (Digital Isolators Test)
    -rD                   (Fast [no curses] Digital Isolators Test)
    -ri                   (Read DIO input channels)
    -rl                   (Internal Loopback DIO test)
    -rL                   (Fast [no curses] internal loopback DIO test)
    -re                   (External Loopback DIO test)
    -rE                   (Fast [no curses] external loopback DIO test)
    -ro                   (Write pattern to DIO output channels)
 -s SkipChannelsMask      (Skip channels mask -- default is 0@0x00000000
                           1@0x00000000 2@0x00000000)
    -s0@XXXXXXXX          (Channels 31..00=XXXXXXXX in Hex)
    -s1@XXXXXXXX          (Channels 63..32=XXXXXXXX in Hex)
    -s2@XXXXXXXX          (Channels 95..64=XXXXXXXX in Hex)
 e.g. ./ccurpmfc_dio -rl -s1@ffffffff (Internal Loopback Testing. Skip Channels
                                       32-63)
      ./ccurpmfc_dio -rE -s2@ffffffff (External Loopback w/o Curses Testing. Skip
                                       Channels 64-95)
```

Example display:

./ccurpmfc_dio -rl -s1@FFFFFFFF

In this example we are performing an internal loopback test. In this case, none of the DIO channels should be connected to any external lines, otherwise, the test will fail.

```
 Board Number          [-b]: 0
 Delay                 [-d]: 100 milli-seconds
```

```
 Loop Count            [-l]: ***Forever***
 DIO Mode              [-m]: 0x00000001 (DIO Normal Mode)
 Number of Channels    [-n]: 96
 Pattern Selection     [-p]: 2 (Adding Bit)
 Run Option            [-r]: 2 (Internal Loopback Test)
 Skip Channels Mask    [-s]: 31..0=00000000, 63..32=FFFFFFFF, 95..64=00000000
 Custom Channels Mask      : 31..0=00000000, 63..32=00000000, 95..64=00000000
 Channel Mismatch Count    : 0 (=== Passed ===)
 DIO Direction             : 0xFFFFFF (All Output)
 DIO Enable                : 0x00000001 (Enable)
 Input Snapshot            : 0x00000001 (Snapshot)
 Scan Count                : 183           (0:00:01:02)

 Write Duration (microsecs) :    1.396 (min=   1.330/max=   1.660/ave=   1.378)
 Read  Duration (microsecs) :    3.687 (min=   3.562/max=   3.771/ave=   3.683)

 Channels     Output    Input
 ===========  ========  ========
  31..00 [0]: FFFFFFFF  FFFFFFFF
  63..32 [1]: 00000000  00000000
  95..64 [2]: 003FFFFF  003FFFFF

           <------------------------ Input Channels ------------------------>
           [0]    [1]    [2]    [3]    [4]    [5]    [6]    [7]    [8]    [9]
           ===    ===    ===    ===    ===    ===    ===    ===    ===    ===
   [0]     +      +      +      +      +      +      +      +      +      +

   [1]     +      +      +      +      +      +      +      +      +      +

   [2]     +      +      +      +      +      +      +      +      +      +

   [3]     +      +    skip   skip   skip   skip   skip   skip   skip   skip

   [4]   skip   skip   skip   skip   skip   skip   skip   skip   skip   skip

   [5]   skip   skip   skip   skip   skip   skip   skip   skip   skip   skip

   [6]   skip   skip   skip   skip     +      +      +      +      +      +

   [7]     +      +      +      +      +      +      +      +      +      +

   [8]     +      +      +      +      +      +      .      .      .      .

   [9]     .      .      .      .      .      .

(*** Enter <CONTROL-C> to Terminate ***)

local_ptr=0x7ffff7fd7000
AvalonPtr=0x7ffff7fd7000

=== Test Passed ===
```

3.2.11 lib/ccurpmfc_dio_intr

This test is used to validate the DIO change-of-state interrupt detection.

```
Usage: ./ccurpmfc_dio_intr [-b Board] [-d Delay] [-F FallCh] [-l LoopCnt]
                           [-L LevelCh] [-R RiseCh] [-X DeleteCh]
 -b <board>            (board #, default = 0)
 -d Delay              (Delay between screen refresh -- default is 0
                        milli-seconds)
```

```
  -F FallCh             (Falling Edge Channel_List)
  -l LoopCnt            (Loop count -- default is 0)
  -L LevelCh            (Level State Channel_List)
  -R RiseCh             (Rising Edge Channel_List)
  -X DeleteCh           (Delete COS Channel_List)

   Examples of Channel_List. Unchanged channels default to Level State Channels:
      -F -              (set all DIO channels to falling edge)
      -R 1,2,7,9        (set channels 1,2,7,9 to rising edge, rest are level state
      -X 5,7-12         (delete channels 5, and 7 to 12. rest are level state
```

Example display:

In the example below, a signal generator is connected to digital input channels 0, 1, 32 and 64. A 15 KHz, +5/0 volts square wave is injected into the selected channels. The test is run using shielding and directing to CPU 4 and 5 and driver interrupt directed to CPU 2. The reason the display shows approximately 30 KHz rate is because the test defaults to performing level detection, i.e. detecting both rising and falling edge on all channels. Depending on other activities in the system, it is possible to see some Overflow conditions.

In order to run at such high rates without getting overflow or missing user callbacks requires proper shielding and real time running of the test. Additionally, the driver interrupt handler needs to be directed to a dedicated processor in order to minimize overflows. E.g.

```
# === as root ===
# shield -a 2, 4-5                      (shield processors 2, 4 and 5)
# cat /proc/ccurpmfc                    (get board irq – in this case it is 'irq=56')
# echo 4 > /proc/irq/56/smp_affinity    (direct board irq to be handled by processor 2)
# (if irq '56' is not present in the proc/irq directory, then you will need to start the test at least once to get it
   assigned by the kernel)
# run -b4-5 ./ccurpmfc_dio_intr
```

```
0001303506 - COS Interrupt Duration (usec):      34.89 min=22.65 max=44.09
                                                 run_ave=33.24 (30081.28 Hz)
     Driver Interrupt Response Time (usec):      17.96 min=15.79 max=27.27
                                                 run_ave=16.72
                 DIO COS Enable (Ch95..0): 0xffffffff 0xffffffff 0xffffffff
                   DIO COS Mode (Ch95..0): 0x00000000 0x00000000 0x00000000
              DIO COS Edge Sense (Ch95..0): 0x00000000 0x00000000 0x00000000
                  DIO COS Status (Ch95..0): 0x00000001 0x00000001 0x00000003
             DIO COS OVFL Status (Ch95..0): 0x00000000 0x00000000 0x00000000
                  Interrupts Occurred Mask: 0x00001c00
                     Wakeup Interrupt Mask: 0x00001c00

                Total DIO Interrupts Count:    6520825
                       User Callback Count:    6520825
                Missed User Callback Count:          0
                DIO COS Counts (group2..0):    6520825    6520825    6520825
       Overflow DIO COS Counts (group2..0):          0          0          0
```

```
    <---------------- Channels COS Status Count    ('+' Rise, '-' Fall, '.' Level, 'NC' No COS) ---------------->
          [0]       [1]       [2]       [3]       [4]       [5]       [6]       [7]       [8]       [9]
    ========== ========== ========== ========== ========== ========== ========== ========== ========== ==========
[0]   6520825.  6520825.        0.        0.        0.        0.        0.        0.        0.        0.
[1]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[2]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[3]         0.        0.  6520825.        0.        0.        0.        0.        0.        0.        0.
[4]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[5]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[6]         0.        0.        0.        0.  6520825.        0.        0.        0.        0.        0.
[7]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[8]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[9]         0.        0.        0.        0.        0.        0.

    <--------------- Channels COS Overflow Count    ('+' Rise, '-' Fall, '.' Level, 'NC' No COS) --------------->
```

```
          [0]       [1]       [2]       [3]       [4]       [5]       [6]       [7]       [8]       [9]
    ========== ========== ========== ========== ========== ========== ========== ========== ========== ==========
 [0]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [1]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [2]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [3]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [4]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [5]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [6]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [7]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [8]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [9]         0.        0.        0.        0.        0.        0.

(*** Enter <CONTROL-C> to Terminate ***)

 Rising Edge[-r]: ### No Channels Selected ###
Falling Edge[-f]: ### No Channels Selected ###
 Level State[-l]: Number of Channels = 96
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 Disable COS[-x]: ### No Channels Selected ###

driver_lib_ptr: 0x7ffff7fdd000
Deactivate DIO
WakeupInterruptMask = 0x00001c00
Activate DIO
### Test Terminated ###

Destroying User COS Interrupt Handler
--------------------------------------------------------------------------------
```

./ccurpmfc_dio_intr -F0,32 -R64

In this example, we are detecting the falling edge for channels 0 and 32, rising edge for channel 64 and level detection for the rest of the channels. This is why you will see the count for channel 1 (level detection) double that of channels 0, 32 and 64.

```
 0001236827 - COS Interrupt Duration (usec):      32.88 min=20.72 max=46.17
                                                  run_ave=33.24 (30080.95 Hz)
      Driver Interrupt Response Time (usec):      12.86 min=12.63 max=25.93
                                                  run_ave=13.38
                  DIO COS Enable (Ch95..0): 0xffffffff 0xffffffff 0xffffffff
                    DIO COS Mode (Ch95..0): 0x00000001 0x00000001 0x00000001
               DIO COS Edge Sense (Ch95..0): 0x00000001 0x00000000 0x00000000
                    DIO COS Status (Ch95..0): 0x00000001 0x00000001 0x00000002
              DIO COS OVFL Status (Ch95..0): 0x00000000 0x00000000 0x00000000
                   Interrupts Occurred Mask: 0x00001400
                      Wakeup Interrupt Mask: 0x00001c00

                 Total DIO Interrupts Count:    6684258
                        User Callback Count:    6684258
                 Missed User Callback Count:          0
                       DMA Counts (dma0..0):          0          0
                 DIO COS Counts (group2..0):    3342129    3342129    6684258
        Overflow DIO COS Counts (group2..0):          0          0          0

     <---------------- Channels COS Status Count    ('+' Rise, '-' Fall, '.' Level, 'NC' No COS) ---------------->
          [0]       [1]       [2]       [3]       [4]       [5]       [6]       [7]       [8]       [9]
     ========== ========== ========== ========== ========== ========== ========== ========== ========== ==========
 [0]    3342129-   6684258.        0.        0.        0.        0.        0.        0.        0.        0.
 [1]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [2]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [3]         0.        0.   3342129-        0.        0.        0.        0.        0.        0.        0.
 [4]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [5]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
 [6]         0.        0.        0.        0.   3342129+        0.        0.        0.        0.        0.
```

```
[7]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[8]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[9]         0.        0.        0.        0.        0.        0.

    <--------------- Channels COS Overflow Count    ('+' Rise, '-' Fall, '.' Level, 'NC' No COS) --------------->
          [0]       [1]       [2]       [3]       [4]       [5]       [6]       [7]       [8]       [9]
    ========== ========== ========== ========== ========== ========== ========== ========== ========== ==========
[0]         0-        0.        0.        0.        0.        0.        0.        0.        0.        0.
[1]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[2]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[3]         0.        0.        0-        0.        0.        0.        0.        0.        0.        0.
[4]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[5]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[6]         0.        0.        0.        0.        0+        0.        0.        0.        0.        0.
[7]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[8]         0.        0.        0.        0.        0.        0.        0.        0.        0.        0.
[9]         0.        0.        0.        0.        0.        0.


(*** Enter <CONTROL-C> to Terminate ***)

 Rising Edge[-r]: Number of Channels = 1
64
Falling Edge[-f]: Number of Channels = 2
 0 32
 Level State[-l]: Number of Channels = 93
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
52 53 54 55 56 57 58 59 60 61 62 63 65 66 67 68 69 70 71 72 73 74 75 76 77
78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 Disable COS[-x]: ### No Channels Selected ###

driver_lib_ptr: 0x7ffff7fdd000
Deactivate DIO
WakeupInterruptMask = 0x00001c00
Activate DIO
### Test Terminated ###
```

3.2.12 lib/ccurpmfc_disp

Useful program to display the local board registers. This program uses the *curses* library. This test is similar to the previous non-library test.

```
Usage: ./ccurpmfc_disp [-b Board] [-d Delay] [-D DMAEngineNo] [-H] [-i]
                       [-l LoopCnt] [-m XferMode] [-o Offset] [-P Pause]
                       [-s XferSize] [-S DispSize]
 -b Board        (Board number -- default board is 0)
 -d Delay        (Delay between screen refresh -- default is 0)
 -D DMAEngineNo  (DMA Engine number -- default = 1)
 -H              (Enable Hyper-Drive Mode -- default "=== Disabled ===")
 -i              (Enable Interrupts -- default = Disable)
 -l LoopCnt      (Loop Count - default = 0)
 -m XferMode     (Transfer Mode -- default = DMA)
    -md          (Avalon Memory: DMA mode)
    -mm          (Avalon Memory: Modular Scatter-Gather DMA mode)
    -mp          (Avalon Memory: Programmed I/O mode)
    -mS          (SDRAM Memory: DMA mode)
    -ms          (SDRAM Memory: Programmed I/O mode)
 -o Offset       (Hex offset to read from -- default is 0x0)
 -P Pause        (Microseconds to sleep in User Function loop -- default is 0)
 -s XferSize     (Number of bytes to transfer -- default is 0x1000)
 -S DispSize     (Number of bytes to display -- default is 0x200)
```

Example display:

./ccurpmfc_disp

```
local_ptr=0x7ffff7fe7000
          Physical Memory Information:
            UserPID         =18944
            PhysMemPtr      =0x79a50000
            DriverVirtMemPtr=0xffff880079a50000
            MmapedUserMemPtr=0x7ffff7fdc000
            PhysMemSize     =0x00001000
            PhysMemSizeFreed=0x00000000
            EntryInTxTbl    =0
            NumOfEntriesUsed=1
            Flags           =0x0000
----------------------------------------------------------------------
 Board Number          [-b]: 0
 Delay                 [-d]: 0 milli-seconds
 DMA Engine            [-D]: 1
 Hyper-Drive           [-H]: Disabled
 Interrupts            [-i]: Disabled
 Loop Count            [-l]: ***Forever***
 Transfer Mode         [-m]: Basic DMA I/O (Avalon Memory)
 Offset                [-o]: 0x00000000
 Transfer Size         [-s]: 0x00001000 (4096) bytes (  21.283 MBytes/Second)
 Display Size          [-S]: 0x00000200 (512) bytes

 ScanCount                 : 12029
 Read Duration (microsecs) :   192.451 (min= 192.276/max= 262.459/ave= 192.733)

         00       04       08       0C       10       14       18       1C
         ======== ======== ======== ======== ======== ======== ======== ========
000000   92900101 06192019 00040000 00000000 00000000 00000000 00000000 00000000
000020   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000040   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000060   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000080   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000a0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000c0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000e0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000100   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000120   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000140   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000160   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000180   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001a0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001c0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001e0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

3.2.13 lib/ccurpmfc_dma

This test transfers data from physical memory to the Local register area and back. There are three modes of operation. One is regular DMA, the second is Modular Scatter-Gather DMA and the third is programmed I/O. Depending on the number of DMA engines supported by the card, the user can select one of them to perform the DMA. Additionally, if the card supports Modular Scatter-Gather DMA, then they can also select that. Area select is one of three areas the user can specify. They represent the area in physical memory and local register where the transfer is to occur. The test automatically switches to a different area corresponding to the regular DMA engine supplied. If multiple copies of this application is run on the same card using the same DMA engine, then the user needs to manually select a different area '-A' so the data mismatch does not occur due to using the same area region.

```
Usage: ./ccurpmfc_dma [-A Area2Select] [-b Board] [-D DMAEngineNo] [-i]
                      [-l LoopCnt] [-m XferMode] [-s Size] [-v VerboseNo]
 -A Area2Select (Area to select -- default = -1)
```

```
 -b Board       (Board number -- default = 0)
 -D DMAEngineNo (DMA Engine number -- default = 0)
 -i             (Enable Interrupts -- default = Disable)
 -l LoopCnt     (Loop Count - default = 1000)
 -m XferMode    (Transfer Mode -- default = DMA)
    -md         (DMA mode)
    -mm         (MsgDma mode - '-D' option is ignored)
    -mp         (Programmed I/O mode)
 -s Size        (Transfer Size in bytes (multiple of byte width) –
                 default = 12288)
 -V VerboseNo   (verbose -- default = 0)

 e.g. ./ccurpmfc_dma -A1    (perform dma using DMA0 on area 1 )
      ./ccurpmfc_dma -i -D1 (perform dma using DMA1 with interrupts on area 0)
      ./ccurpmfc_dma -mm    (perform dma using MsgDMA on area 0)
```

Example display:

./ccurpmfc_dma

```
Device Name: /dev/ccurpmfc0
local_ptr=0x7ffff7fe7000
           Physical Memory Information:
             UserPID         =18950
             PhysMemPtr      =0x5ea00000
             DriverVirtMemPtr=0xffff88005ea00000
             MmapedUserMemPtr=0x7ffff70f4000
             PhysMemSize     =0x00200000
             PhysMemSizeFreed=0x00000000
             EntryInTxTbl    =0
             NumOfEntriesUsed=2
             Flags           =0x0000
### Avalon Address[A0]: 0x00001000 - 0x00004000
###    DMA Address[A0]: 0x00100400 - 0x00103400
###      Transfer Size: 12288 (0x00003000) bytes (DMA without Interrupts: DMA
                        Engine 0) ###
  1000: A2P: Total:  566.852us (  21.68 MB/s): first=0xface0000 last=0xface0bff

          (micro-seconds)          (MBytes/second)
       Min     Max     Ave      Min     Max     Ave
    ----------------------  ------------------------
P2A:  514.18  523.39  516.20     23.48   23.90   23.80
A2P:  566.28  594.08  567.11     20.68   21.70   21.67
```

./ccurpmfc_dma -b1 -mm *(board supports MsgDma)*

```
Device Name: /dev/ccurpmfc1
local_ptr=0x7ffff7f97000
           Physical Memory Information:
             UserPID         =25634
             PhysMemPtr      =0x35000000
             DriverVirtMemPtr=0xffff96d0f5000000
             MmapedUserMemPtr=0x7ffff7065000
             PhysMemSize     =0x00200000
             PhysMemSizeFreed=0x00000000
             EntryInTxTbl    =0
             NumOfEntriesUsed=2
             Flags           =0x0000
### Avalon Address[A0]: 0x00001000 - 0x00004000
###    DMA Address[A0]: 0x00100400 - 0x00103400
###      Transfer Size: 12288 (0x00003000) bytes (DMA without Interrupts: MsgDma
```

```
                        Engine) ###
  1000: A2P: Total:   43.757us ( 280.82 MB/s): first=0xface0000 last=0xface0bff

         (micro-seconds)          (MBytes/second)
       Min     Max     Ave      Min     Max     Ave
    ----------------------  ----------------------
P2A:   48.38   61.43   49.41    200.03  254.02  248.68
A2P:   43.70   49.04   43.88    250.59  281.18  280.05
```

3.2.14 lib/ccurpmfc_example

This test provides a simple example of programming ADC, DAC and DIO.

```
Usage: ./ccurpmfc_example [-b Board]
 -b Board     (Board number -- default is 0)
```

Example display:

./ccurpmfc_example (for card ***without*** Modular Scatter-Gather DMA)

```
local_ptr=0x7ffff7fe7000
           Physical Memory Information:
             UserPID         =18954
             PhysMemPtr      =0x8566b000
             DriverVirtMemPtr=0xffff88008566b000
             MmapedUserMemPtr=0x7ffff7fdb000
             PhysMemSize     =0x00001000
             PhysMemSizeFreed=0x00000000
             EntryInTxTbl    =1
             NumOfEntriesUsed=1
             Flags           =0x0000
### Configuring ADC ###
  - Activate ADC (disable followed by enable)
  - Configure ADC
  - Set Calibration to Positive Reference Voltage
  - Calibrate ADC
### Configuring DAC ###
  - Activate DAC (disable followed by enable)
  - Select Software Update
  - Configure DAC
  - Write 0 to DAC outputs
  - Make DAC operational
  - Calibrate DAC
### Programming Clocks ###
### Reading ADC Channels ###

     ==== ADC Channels - Using ccurPMFC_Transfer_Data() ==== (length=64)
     +DMP+        0   00007ed4  00007ed5  00007ed5  00007ed5 *..~...~...~...~.*
     +DMP+     0x10   00007ed4  00007ed4  00007ed6  00007ed6 *..~...~...~...~.*
     +DMP+     0x20   00007ed6  00007ed5  00007ed5  00007ed5 *..~...~...~...~.*
     +DMP+     0x30   00007ed5  00007ed5  00007ed6  00007ed6 *..~...~...~...~.*

### Reading ADC Channels using ccurPMFC_DMA_Configure()/ccurPMFC_DMA_Fire() ###
  - Convert Physical DMA Memory Address to Avalon Equivalent Address
  - Configure DMA
  - Fire DMA

     ==== ADC Channels - Using ccurPMFC_DMA_Fire() ==== (length=64)
     +DMP+        0   00007eda  00007edb  00007edc  00007eda *..~...~...~...~.*
     +DMP+     0x10   00007edb  00007eda  00007edb  00007eda *..~...~...~...~.*
     +DMP+     0x20   00007edc  00007eda  00007eda  00007eda *..~...~...~...~.*
```

```
     +DMP+     0x30   00007edb  00007eda  00007eda  00007edb *..~...~...~...~.*
### Writing DAC Channels ###
  DacCh00: 0x00000000 (0.000000 volts)
  DacCh01: 0x00000666 (0.499878 volts)
  DacCh02: 0x00000ccc (0.999756 volts)
  DacCh03: 0x00001333 (1.499939 volts)
  DacCh04: 0x00001999 (1.999817 volts)
  DacCh05: 0x00002000 (2.500000 volts)
  DacCh06: 0x00002666 (2.999878 volts)
  DacCh07: 0x00002ccc (3.499756 volts)
  DacCh08: 0x00003333 (3.999939 volts)
  DacCh09: 0x00003999 (4.499817 volts)
  DacCh10: 0x00004000 (5.000000 volts)
  DacCh11: 0x00004666 (5.499878 volts)
  DacCh12: 0x00004ccc (5.999756 volts)
  DacCh13: 0x00005333 (6.499939 volts)
  DacCh14: 0x00005999 (6.999817 volts)
  DacCh15: 0x00006000 (7.500000 volts)
### Configuring DIO ###
  - Activate DIO (disable followed by enable)
  - Set DIO for normal mode
  - Set DIO output sync mode
  - Set DIO input snapshot mode
  - Set DIO ports direction
### Reading DIO Channels 00..47 ###
CCURPMFC_DIO_CHAN_00_31=0xaaaaaaab
CCURPMFC_DIO_CHAN_32_63=0x0000aaab
### Writing DIO Channels 48..95 ###
CCURPMFC_DIO_CHAN_32_63=0xbabe0000
CCURPMFC_DIO_CHAN_64_95=0xfeedface
```

./ccurpmfc_example (for card ***with*** Modular Scatter-Gather DMA)

```
local_ptr=0x7ffff7f97000
           Physical Memory Information:
             UserPID         =32485
             PhysMemPtr      =0x35f7c000
             DriverVirtMemPtr=0xffff96d0f5f7c000
             MmapedUserMemPtr=0x7ffff7feb000
             PhysMemSize     =0x00001000
             PhysMemSizeFreed=0x00000000
             EntryInTxTbl    =1
             NumOfEntriesUsed=1
             Flags           =0x0000
### Configuring ADC ###
  - Activate ADC
  - Configure ADC
  - Set Calibration to Positive Reference Voltage
  - Calibrate ADC
### Configuring DAC ###
  - Activate DAC
  - Select Software Update
  - Configure DAC
  - Write 0 to DAC outputs
  - Make DAC operational
  - Calibrate DAC
### Programming Clocks ###
### Reading ADC Channels ###

     ==== ADC Channels - Using ccurPMFC_Transfer_Data() ==== (length=64)
     +DMP+        0   00007ed8  00007ed9  00007ed7  00007ed8 *..~...~...~...~.*
```

```
     +DMP+     0x10   00007ed8  00007ed9  00007ed8  00007ed9 *..~...~...~...~.*
     +DMP+     0x20   00007ed8  00007ed9  00007ed8  00007ed9 *..~...~...~...~.*
     +DMP+     0x30   00007ed8  00007ed9  00007ed9  00007ed9 *..~...~...~...~.*

### Reading ADC Channels using ccurPMFC_DMA_Configure()/ccurPMFC_DMA_Fire() ###
  - Convert Physical DMA Memory Address to Avalon Equivalent Address
  - Configure DMA
  - Fire DMA

     ==== ADC Channels - Using ccurPMFC_DMA_Fire() ==== (length=64)
     +DMP+        0   00007ed9  00007ed9  00007ed8  00007ed8 *..~...~...~...~.*
     +DMP+     0x10   00007ed9  00007ed9  00007ed9  00007eda *..~...~...~...~.*
     +DMP+     0x20   00007ed9  00007ed9  00007ed8  00007ed9 *..~...~...~...~.*
     +DMP+     0x30   00007ed8  00007ed9  00007ed8  00007ed9 *..~...~...~...~.*
### Writing DAC Channels ###
  DacCh00: 0x00000000 (0.000000 volts)
  DacCh01: 0x00000666 (0.499878 volts)
  DacCh02: 0x00000ccd (1.000061 volts)
  DacCh03: 0x00001333 (1.499939 volts)
  DacCh04: 0x0000199a (2.000122 volts)
  DacCh05: 0x00002000 (2.500000 volts)
  DacCh06: 0x00002666 (2.999878 volts)
  DacCh07: 0x00002ccd (3.500061 volts)
  DacCh08: 0x00003333 (3.999939 volts)
  DacCh09: 0x0000399a (4.500122 volts)
  DacCh10: 0x00004000 (5.000000 volts)
  DacCh11: 0x00004666 (5.499878 volts)
  DacCh12: 0x00004ccd (6.000061 volts)
  DacCh13: 0x00005333 (6.499939 volts)
  DacCh14: 0x0000599a (7.000122 volts)
  DacCh15: 0x00006000 (7.500000 volts)
### Configuring DIO ###
  - Activate DIO
  - Set DIO for normal mode
  - Set DIO output sync mode
  - Set DIO input snapshot mode
  - Set DIO ports direction
### Reading DIO Channels 00..47 ###
  - CCURPMFC_DIO_CHAN_00_31=0x00000000
  - CCURPMFC_DIO_CHAN_32_63=0x00000000
### Writing DIO Channels 48..95 ###
  - CCURPMFC_DIO_CHAN_32_63=0xbabe0000
  - CCURPMFC_DIO_CHAN_64_95=0xfeedface
### Single (one descriptor) Modular Scatter-Gather DMA ###
  - Allocating memory and seeding with pattern
  - Seizing MSGDMA
  - Configure Single MSGDMA (PCIe ==> Avalon)
  - Fire Single MSGDMA: Xfer 0x8000 bytes: Pcie ==> Avalon (@0x8000)
  - Validating data
  - Configure Single MSGDMA (Avalon ==> PCIe)
  - Fire Single MSGDMA: Xfer 0x8000 bytes: Avalon (@0x8000) ==> PCIe
  - Validating data
  - Releasing MSGDMA
### Multi (four descriptor) Modular Scatter-Gather DMA (Single-Shot) ###
  - Allocating memory and seeding with pattern)
  - Seizing MSGDMA
  - Configure multi MSGDMA (PCIe ==> Avalon ==> PCIe ==> Avalon ==> PCIe)
  - Setup Multi MSGDMA
  - Fire Multi MSGDMA (Single-Shot)
  - Validating data
  - Releasing MSGDMA
```

```
### Multi (four descriptor) Modular Scatter-Gather DMA (Clone) ###
  - Allocating memory and seeding with pattern)
  - Seizing MSGDMA
  - Configure multi MSGDMA (PCIe ==> Avalon ==> PCIe ==> Avalon ==> PCIe)
  - Setup Multi MSGDMA
  - Stop and Initialize Multi MSGDMA (Clone)
  - Fire Multi MSGDMA and wait one cycle (Clone: Once cycle wait)
  - Validating data
  - Releasing MSGDMA
```

3.2.15 lib/ccurpmfc_expires

This test is useful in displaying board expires information.

```
Usage: ./ccurpmfc_expires -[b Board] -[s]
          -b <board>            (board #, default = 0)
          -s                    (short display, default = verbose)
```

Example display:

./ccurpmfc_expires *(for card that has no restrictions)*

```
      Device Name: /dev/ccurpmfc0
   Board Serial No: 98765 (0x000181cd)

###############################################
###                                         ###
###         UNRESTRICTED FIRMWARE           ###
###                                         ###
###############################################
```

./ccurpmfc_expires *(for restricted card that has NO expiration date)*

```
          Device Name: /dev/ccurpmfc0
     Board Serial No: 98765 (0x000181cd)

###########################################
###                                     ###
###         RESTRICTED FIRMWARE         ###
###                                     ###
###########################################

=========================
=== No Expiration Date ===
=========================
```

./ccurpmfc_expires *(for restricted card that has expiration date)*

```
          Device Name: /dev/ccurpmfc0
     Board Serial No: 98765 (0x000181cd)

###########################################
###                                     ###
###         RESTRICTED FIRMWARE         ###
###                                     ###
###########################################

=====================================================================
Local Expiration Date: 03/11/2018 13:21:52
  GMT Expiration Date: 03/11/2018 17:21:52
   Duration to Expire: Days=122, Hours=2, Minutes=49, Seconds=20
```

```
================================================================
```
./ccurpmfc_expires -s *(for card that has no restructions)*

```
Unrestricted
```

./ccurpmfc_expires -s *(for restricted card that has NO expiration date)*

```
Restricted: No expiration date
```

./ccurpmfc_expires -s *(for restricted card that has expiration date)*

```
Restricted: Expire in 10550462 seconds
```

3.2.16 lib/ccurpmfc_identify

This test is useful in identifying a particular card by displaying its LED.

```
Usage: ./ccurpmfc_identify -[absx]
          -a                    (Identify all cards through a light sequence)
          -b <board>            (board #, default = 0)
          -s <seconds)          (Identify Board: ENABLED for number of seconds,
                                 default = 10)
          -s 0                  (Identify Board: DISABLED)
          -s <negative value>   (Identify Board: ENABLED forever)
          -x                    (silent)

If the '-a' option is selected, all other options are ignored. This option will
sequence through all the cards found in turn as follows:
  1) The first device number will flash its LED for 10 seconds
  2) The remaining devices numbers will be selected sequentially and flash their LEDs for
3 seconds
```

Example display:

./ccurpmfc_identify

```
Device Name       : /dev/ccurpmfc0
Board ID          : 9290
Board Type        : 02
Board Function    : 01
Board Serial No   : 674459 (0x000a4a9b)
Firmware Revision : 4.0 (Major.Minor)
MsgDma Support    : 0 descriptors (No)

Identify ENABLED on board 0 (LED should start flashing for 10 seconds)
Sleeping for 10 seconds...
Identify DISABLED on board 0 (LED should stop flashing)
```

./ccurpmfc_identify -a

```
TotalBoardCount=5
 # DNum IRQ MSI Bu:Sl:Fn VnID:Sub  BdID:Ty:Fu:Sub  FMaj.Min(mm:dd:yy hh:mm:ss)  MC
FmFlvCod FwbRev   IPCores Temp:C/F SerialNo RLS# Func
 0  0   166  Y  0c:00:00 1542:1542 9290.03.30:0100 1.1(03/18/21 13:57:02)       B7
46443131 20200201    4    30/ 86.0 690051   150  CustomIpc
 1  1   167  Y  0d:00:00 1542:1542 9290.03.30:0100 1.1(03/18/21 13:57:02)       B7
46443131 20200201    4    34/ 93.2 690055   150  CustomIpc
 2  2   168  Y  0e:00:00 1542:1542 9290.03.30:0100 1.1(03/18/21 13:57:02)       B7
46443131 20200201    4    36/ 96.8 674468   150  CustomIpc
```

```
 3  3   164  Y  05:00:00 1542:1542 9290.01.01:0100 4.0(06/19/19 00:00:00)       A5
00000000 00000000    0    35/ 95.0 696690   100  MultiFunc
 4  4   165  Y  06:00:00 1542:1542 9290.01.01:0100 4.0(06/19/19 00:00:00)       A5
00000000 00000000    0    31/ 87.8 696689   100  MultiFunc

Device Numbers: (enter <CTRL-C> to terminate)
============================================
 0*   1    2    3    4
```

3.2.17 lib/ccurpmfc_info

This test is useful in getting information for all the *ccurpmfc* devices in the system.

```
Usage: ./ccurpmfc_info -[b Board] -[l] -[v]
          -b <board>            (board #, default = 0)
          -l                    (long display, default = short)
          -v                    (long display and verbose, default = no verbose)
          -l -v                 (long display and verbose, default = no verbose)
```

Example display:

./ccurpmfc_info

```
# IRQ MSI Bu:Sl:Fn VnID:Sub  BdID:Ty:Fu:Sub  FMaj.Min(mm:dd:yy hh:mm:ss)  MC FmFlvCod FwbRev   IPCores
Temp:C/F SerialNo RLS# Func
 0  56  Y  06:00:00 1542:1542 9290.01.01:0100 4.0(06/19/19 00:00:00)       A5 00000000 00000000    0
54/129.2 11223344   100  MultiFunc
 1  58  Y  07:00:00 1542:1542 9290.02.30:0100 1.1(05/28/19 15:49:33)       B3 45523031 20190200    2
45/113.0 668603   100  CustomIpc
```

./ccurpmfc_info -l

```
        ###################### Board 0 ######################
                        Version:  2022.1.0
                          Build:  Thu Jan 20 08:44:36 EST 2022
                         Module:  ccurpmfc
                    Board Index:  0 (PCIe-CCUR_FPGA_PMFC)
                            Bus:  0x06
                           Slot:  0x00
                           Func:  0x00
                      Vendor ID:  0x1542
                  Sub-Vendor ID:  0x1542
                     Board Info:  0x92900101 (id=9290, type=0x01, func=0x01 (MultiFunc))
                    Member Code:  1 (A5)
                  Sub-Device ID:  0x0100
              Firmware Date/Time:  0x06192019 0x00000000 (06/19/2019 00:00:00)
               Firmware Revision:  0x00040000 (4.0)
                 Fpgawb Revision:  0x00000000 (0000.00-00) (Not Supported)
           Firmware Flavor Code:  0x00000000 (0) (****)
     Number of Advanced IP Cores:  0x00000000 (0)
            Board Serial Number:  0x00ab4130 (11223344)
          Board SPROM Revision:  0x0000 (0)
         FPGA Chip Temperature:  0x36 (54 degree C, 129.2 degree F)
        Run Level Sector Number:  0x64 (100)
         Multi-Firmware Support:  0x1 (Yes)
                    MSI Support:  Enabled
      Scatter-Gather DMA Support:  No
            Double-Word Support:  No
                      IRQ Level:  56
                 Cloning Support:  0 (Cloning is not supported)
           Calibration Reference:  9.91 Volts
```

```
###################### Board 1 ######################
                  Version:  2022.1.0
                    Build:  Thu Jan 20 08:44:36 EST 2022
                   Module:  ccurpmfc
              Board Index:  0 (PCIe-CCUR_FPGA_PMFC)
                      Bus:  0x07
                     Slot:  0x00
                     Func:  0x00
                Vendor ID:  0x1542
            Sub-Vendor ID:  0x1542
               Board Info:  0x92900230 (id=9290, type=0x02, func=0x30 (CustomIpc))
              Member Code:  2 (B3)
            Sub-Device ID:  0x0100
         Firmware Date/Time:  0x05282019 0x00154933 (05/28/2019 15:49:33)
          Firmware Revision:  0x00010001 (1.1)
            Fpgawb Revision:  0x20190200 (2019.02-00) (Supported)
       Firmware Flavor Code:  0x45523031 (1163014193) (ER01)
Number of Advanced IP Cores:  0x00000002 (2)
        Board Serial Number:  0x000a33bb (668603)
       Board SPROM Revision:  0x0000 (0)
      FPGA Chip Temperature:  0x2d (45 degree C, 113.0 degree F)
   Run Level Sector Number:  0x64 (100)
    Multi-Firmware Support:  0x1 (Yes)
               MSI Support:  Enabled
 Scatter-Gather DMA Support:  Yes
Number of MSG DMA Descriptors:  31
         Double-Word Support:  No
                  IRQ Level:  58
             Cloning Support:  3 (Cloning is supported. Region addressing allowed for any user)
      Calibration Reference:  9.91 Volts
```

./ccurpmfc_info -l –v

```
###################### Board 0 ######################
                  Version:  2022.1.0
                    Build:  Thu Jan 20 08:44:36 EST 2022
                   Module:  ccurpmfc
              Board Index:  0 (PCIe-CCUR_FPGA_PMFC)
                      Bus:  0x06
                     Slot:  0x00
                     Func:  0x00
                Vendor ID:  0x1542
            Sub-Vendor ID:  0x1542
               Board Info:  0x92900101 (id=9290, type=0x01, func=0x01 (MultiFunc))
              Member Code:  1 (A5)
            Sub-Device ID:  0x0100
         Firmware Date/Time:  0x06192019 0x00000000 (06/19/2019 00:00:00)
          Firmware Revision:  0x00040000 (4.0)
            Fpgawb Revision:  0x00000000 (0000.00-00) (Not Supported)
       Firmware Flavor Code:  0x00000000 (0) (****)
Number of Advanced IP Cores:  0x00000000 (0)
        Board Serial Number:  0x00ab4130 (11223344)
       Board SPROM Revision:  0x0000 (0)
      FPGA Chip Temperature:  0x36 (54 degree C, 129.2 degree F)
   Run Level Sector Number:  0x64 (100)
    Multi-Firmware Support:  0x1 (Yes)
               MSI Support:  Enabled
 Scatter-Gather DMA Support:  No
         Double-Word Support:  No
                  IRQ Level:  56
             Cloning Support:  0 (Cloning is not supported)
      Calibration Reference:  9.91 Volts

          ---ADC Information---
    Maximum Voltage Range:  10 Volts
           Number of ADCs:  2
  Number of ADC Channels:  16
Number of ADC Resolution:  16 Bits
```

```
          All ADC Channels Mask:  0x0000ffff
      Maximum ADC Fifo Threshold:  0x00020000

             ---DAC Information---
          Maximum Voltage Range:  20 Volts
                 Number of DACs:  4
        Number of DAC Channels:  16
     Number of DAC Resolution:  16 Bits
        All DAC Channels Mask:  0x0000ffff

             ---DIO Information---
        Number of DIO Channels:  96
           Number of DIO Ports:  24
  Number of DIO Channels/Port:  4
       Number of DIO Registers:  3
Number of DIO Channels/Register:  32

             ---DMA Information---
               Driver DMA Size:  524288
      Num of Trans Tbl Entries:  8
              Avalon Page Bits:  20
              Avalon Page Size:  1048576
             TX Interface Base:  8388608
          DMA Maximum Engines:  2
        DMA Maximum Burst Size:  1024
      DMA Maximum Transactions:  32
              DMA Maximum Size:  1048576
            DMA Width in Bytes:  4
              DMA Fire Command:  140

          ---Analog/DMA Interrupt Information---
               Interrupt Count:  0
                   DMA 0 Count:  0
                   DMA 1 Count:  0
                 MSG DMA Count:  0
     Interrupts Occurred Mask:  0x00000000
        Wakeup Interrupt Mask:  0x00000000
               Timeout Seconds:  0
                   DMA Control:  0x00000000

           ---DIO COS Interrupt Information---
            DIO Interrupt Count:  0
          DIO Group 0 COS Count:  0
          DIO Group 1 COS Count:  0
          DIO Group 2 COS Count:  0
    DIO Group 0 COS OVFL Count:  0
    DIO Group 1 COS OVFL Count:  0
    DIO Group 2 COS OVFL Count:  0
      Interrupts Occurred Mask:  0x00000000
         Wakeup Interrupt Mask:  0x00000000
        DIO Group 0 COS Status:  0x00000000
        DIO Group 1 COS Status:  0x00000000
        DIO Group 2 COS Status:  0x00000000
   DIO Group 0 COS Ovfl Status:  0x00000000
   DIO Group 1 COS Ovfl Status:  0x00000000
   DIO Group 2 COS Ovfl Status:  0x00000000

          ---Memory Regions Information---
                      Region 0:  Addr=0xbd340000  Size=32768 (0x8000)
                      Region 2:  Addr=0xbd300000  Size=262144 (0x40000)

        ###################### Board 1 ######################
                       Version:  2022.1.0
                         Build:  Thu Jan 20 08:44:36 EST 2022
                        Module:  ccurpmfc
                   Board Index:  0 (PCIe-CCUR_FPGA_PMFC)
                           Bus:  0x07
                          Slot:  0x00
                          Func:  0x00
```

```
                    Vendor ID:  0x1542
                Sub-Vendor ID:  0x1542
                   Board Info:  0x92900230 (id=9290, type=0x02, func=0x30 (CustomIpc))
                  Member Code:  2 (B3)
                Sub-Device ID:  0x0100
           Firmware Date/Time:  0x05282019 0x00154933 (05/28/2019 15:49:33)
            Firmware Revision:  0x00010001 (1.1)
              Fpgawb Revision:  0x20190200 (2019.02-00) (Supported)
         Firmware Flavor Code:  0x45523031 (1163014193) (ER01)
  Number of Advanced IP Cores:  0x00000002 (2)
          Board Serial Number:  0x000a33bb (668603)
         Board SPROM Revision:  0x0000 (0)
        FPGA Chip Temperature:  0x2d (45 degree C, 113.0 degree F)
      Run Level Sector Number:  0x64 (100)
       Multi-Firmware Support:  0x1 (Yes)
                  MSI Support:  Enabled
   Scatter-Gather DMA Support:  Yes
Number of MSG DMA Descriptors:  31
         Double-Word Support:  No
                    IRQ Level:  58
           Cloning Support:  3 (Cloning is supported. Region addressing allowed for any user)
      Calibration Reference:  9.91 Volts

          ---ADC Information---
      Maximum Voltage Range:  10 Volts
             Number of ADCs:  2
    Number of ADC Channels:  16
  Number of ADC Resolution:  16 Bits
     All ADC Channels Mask:  0x0000ffff
 Maximum ADC Fifo Threshold:  0x00020000

          ---DAC Information---
      Maximum Voltage Range:  20 Volts
             Number of DACs:  4
    Number of DAC Channels:  16
  Number of DAC Resolution:  16 Bits
     All DAC Channels Mask:  0x0000ffff

          ---DIO Information---
    Number of DIO Channels:  96
       Number of DIO Ports:  24
Number of DIO Channels/Port:  4
    Number of DIO Registers:  3
Number of DIO Channels/Register:  32

          ---DMA Information---
        Driver DMA Size:  524288
  Num of Trans Tbl Entries:  8
         Avalon Page Bits:  20
         Avalon Page Size:  1048576
        TX Interface Base:  8388608
      DMA Maximum Engines:  2
    DMA Maximum Burst Size:  1024
  DMA Maximum Transactions:  32
         DMA Maximum Size:  1048576
       DMA Width in Bytes:  4
         DMA Fire Command:  140

      ---Analog/DMA Interrupt Information---
           Interrupt Count:  0
               DMA 0 Count:  0
               DMA 1 Count:  0
            MSG DMA Count:  0
  Interrupts Occurred Mask:  0x00000000
     Wakeup Interrupt Mask:  0x00000000
           Timeout Seconds:  0
               DMA Control:  0x00000000

      ---DIO COS Interrupt Information---
```

```
        DIO Interrupt Count:  0
      DIO Group 0 COS Count:  0
      DIO Group 1 COS Count:  0
      DIO Group 2 COS Count:  0
 DIO Group 0 COS OVFL Count:  0
 DIO Group 1 COS OVFL Count:  0
 DIO Group 2 COS OVFL Count:  0
   Interrupts Occurred Mask:  0x00000000
      Wakeup Interrupt Mask:  0x00000000
     DIO Group 0 COS Status:  0x00000000
     DIO Group 1 COS Status:  0x00000000
     DIO Group 2 COS Status:  0x00000000
DIO Group 0 COS Ovfl Status:  0x00000000
DIO Group 1 COS Ovfl Status:  0x00000000
DIO Group 2 COS Ovfl Status:  0x00000000

       ---Ip Core Information---
           IpCore Code [00]:  0x0000000a ([10]: CCURPMFC_IPCODE_PWM_OUTPUT - PWM Output)
       IpCore Revision [00]:  0x00010000 (1.0)
         IpCore Offset [00]:  0x00012200
    IpCore Information [00]:  0x00000001
 IpCore Mapped Pointer [00]:  0x7ffff7fa9200
           IpCore Code [01]:  0x00000009 ([9]: CCURPMFC_IPCODE_PWM_INPUT - PWM Input)
       IpCore Revision [01]:  0x00010000 (1.0)
         IpCore Offset [01]:  0x00012800
    IpCore Information [01]:  0x00000001
 IpCore Mapped Pointer [01]:  0x7ffff7fa9800

       ---Memory Regions Information---
                   Region 0:  Addr=0xbd240000  Size=32768 (0x8000)
                   Region 2:  Addr=0xbd200000  Size=262144 (0x40000)
```

3.2.18 lib/ccurpmfc_msgdma

This test performs a modular scatter-gather DMA test on boards that support it. Additionally, it displays performance information for each mode of operation.

```
Usage: ./ccurpmfc_msgdma [-a AddrOff,ToAddrOff] [-b Board] [-d NumDesc]
                         [-f Input,Output] [-i] [-l LoopCnt] [-m Mode]
                         [-s TotalXferSize] [-v] [-X]
          -a <AddrOff,ToAddrOff> (First Avalon Address Offset, default DiagRam offset)
                                 (Second 'ToAddrOff' only for Avalon2Avalon mode)
          -b <Board>             (board #, default = 0)
          -d <NumDesc>           (Number of Descriptors, default = 1)
          -f <Input>,<#Output>   (Use input file as input data. default None)
                                 (Use Output file to write 'to' data. default None)
                                 (Prepend with '#' to remove comments and address)
          -i                     (Use interrupts, default is poll)
          -l <LoopCnt>           (Loop Count, default = 1000)
          -m <Mode>              (Mode of Operation, default = all)
             'a2p'               (Avalon memory address to Pci memory address)
             'p2a'               (Pci memory address to Avalon memory address)
             'p2p'               (Pci memory address to Pci memory address)
             'a2a'               (Avalon memory address to Avalon memory address)
             'all'               (All above modes with only memory addresses)

             'A2p'               (Avalon FIFO address to Pci memory address - specify
                                                          FIFO address '-a')
             'p2A'               (Pci memory address to Avalon FIFO address - specify
                                                          FIFO address '-a')
             'A2A'               (Avalon FIFO address to Avalon FIFO address - specify
                                                          FIFO address '-a')
          -s <TotalXferSize>     (Total Transfer Size in bytes, default size of
                                  DiagRam)
                                 (Maximum transfer size is 0x3FFFF)
          -v                     (Verbose operation. default is quiet)
```

```
            -X                    (Skip Data Validation, default is to validate)

Notes:
     1) For modes 'p2a' or 'a2p' only the first address 'AddrOff' is used in option '-a'
     2) For modes 'a2a' the first address 'AddrOff' is "FROM" and second address
        'ToAddrOff' is "TO"
     3) If Input file is specified in the '-f' option, its contents is used to seed
        input
     4) If '-X' option is specified, no pattern is written to input, unless '-f Input'
        option is specified
     5) Multiple '-m' options can be specified on a single command line
     6) When address '-a' option is not specified, DiagRam offset is used for Analog
        input/output
     7) Normal running process if no arguments specified is as follows:
          a) Incrementing pattern written to the input using programmed I/O and readback
             validated
          b) Output written with 'baadbeef' pattern using programmed I/O
          c) Scatter-Gather DMA performed from Input to Output
          d) Data is read back from both Input and Output using programmed I/O and
             compared
     8) An upper case 'A' in the -m option represents an Avalon FIFO address, while
        a lower case 'a' in the -m option represents a regular Avalon memory address
     9) If a regular memory Avalon address is specified as an Avalon FIFO address and
        vice-versa results will be unpredictable
    10) When either input or output Avalon address is pointing to a FIFO, then data
        validation is skipped
    11) If a size is specified for a memory or FIFO address that is greater than it can
        handle, the result will be unpredictable. You will need to reset the firmware to
        restore proper operation

e.g. ./ccurpmfc_msgdma -mall                    (Run all transfer modes with validation)
     ./ccurpmfc_msgdma -a0x8000 -s0x100         (Run all modes with Avalon Address
                                                 0x8000 and size 0x100)
     ./ccurpmfc_msgdma -a0xA000 -s0x200 -ma2a   (Run a2a with Avalon Address 0xA000 and
                                                 size 0x200)
     ./ccurpmfc_msgdma -mp2a -l1 -d1 -fHexFile_16K -a0x10004 -X
                               (Transfer Input file to Avalon memory at 0x10004)
     ./ccurpmfc_msgdma -ma2p -l1 -d1 -f,OutFile -s0x4000 -a0x10004 -X
                               (Transfer Avalon memory at 0x10004 to output file
                                'OutFile')
     ./ccurpmfc_msgdma -mA2p -l10000 -s0x20000 -d16 -a0x18010
                               (Transfer Avalon FIFO at 0x18010 to PCI memory with 16
                                Descriptors where each descriptor has a transfer size
                                of 0x2000 bytes. No validation will be performed)
```

Example display:

./ccurpmfc_msgdma

```
### TotalXferSize = 0x00008000, individual descriptor length=0x008000 ###
    1000: P2P Total: Size 0x8000, Fire= 129.40us/ 253.23MB/s
              (mi/ma/av:  238.96/ 260.60/ 252.05 MB/s, 125.74/ 137.13/ 130.01 us)
              LastWord=0x007cffff
    1000: A2A Total: Size 0x4000, Fire= 102.79us/ 159.40MB/s
              (mi/ma/av:  153.79/ 160.72/ 159.35 MB/s, 101.94/ 106.53/ 102.82 us)
              LastWord=0x003e7fff
    1000: P2A Total: Size 0x8000, Fire= 117.02us/ 280.01MB/s
              (mi/ma/av:  268.80/ 282.28/ 279.66 MB/s, 116.08/ 121.91/ 117.17 us)
              LastWord=0x007cffff
    1000: A2P Total: Size 0x8000, Fire= 106.55us/ 307.54MB/s
              (mi/ma/av:  295.75/ 309.16/ 307.33 MB/s, 105.99/ 110.80/ 106.62 us)
              LastWord=0x007cffff
```

./ccurpmfc_msgdma -C *(Cloning Option)*

```
### Cloning Option Selected ###
### TotalXferSize = 0x00008000, individual descriptor length=0x008000 ###
    1000: P2P Total: Size 0x8000, Fire= 126.16us/ 259.74MB/s
              (mi/ma/av:  257.47/ 259.79/ 259.70 MB/s, 126.13/ 127.27/ 126.18 us)
              LastWord=0x007cffff
    1000: A2A Total: Size 0x4000, Fire= 103.55us/ 158.22MB/s
              (mi/ma/av:  153.12/ 158.25/ 158.18 MB/s, 103.53/ 107.00/ 103.58 us)
              LastWord=0x003e7fff
    1000: P2A Total: Size 0x8000, Fire= 113.95us/ 287.57MB/s
              (mi/ma/av:  277.85/ 287.61/ 287.48 MB/s, 113.93/ 117.94/ 113.98 us)
              LastWord=0x007cffff
    1000: A2P Total: Size 0x8000, Fire= 102.76us/ 318.88MB/s
              (mi/ma/av:  310.06/ 318.95/ 318.75 MB/s, 102.74/ 105.68/ 102.80 us)
              LastWord=0x007cffff
```

3.2.19 lib/ccurpmfc_msgdma_clone

Cloning is an optional feature of this card that can be purchased separately. The basic cloning option is an extremely powerful tool that gives the user the ability to continuously transfer the contents of a region on the card to a physical memory entirely under hardware control, once cloning has commenced. This continuous transfer is performed at MsgDma speeds. A more advanced cloning option that can be purchased is known as Region Addressing. This option allows the board to Clone any MsgDma location to another MsgDma location, i.e. the source and destination locations can be any valid MsgDma able physical address in the system.

Only one Cloning or MsgDma operation can be active at a given time. Additionally, it is meaningless to perform Cloning on a FIFO region for two reasons. Firstly, each data in a FIFO is synchronous, however, the Cloned region is accessed asynchronously. Secondly, when the FIFO runs empty *(underflow)* or cannot accept more data *(overflow)* the results are unpredictable.

<u>Caution</u>: Since physical addresses are supplied to this test, care must be taken to ensure that the supplied addresses are valid and that while cloning is in progress, the memory regions must not be freed or made inactive, otherwise, the results could be unpredictable and could lead to the possible corruption of the system.

This test show the capabilities of this new cloning option.

```
Usage: ./ccurpmfc_msgdma_clone [-a FromAddr,ToAddr] [-b Board] [-d Delay]
                               [-F DebugFile] [-l LoopCnt] [-P] [-q] [-s XferSize]
                               [-S DisplaySize] [-v Delay] [-X] [-Z]
          -a <FromAddr,ToAddr> (Clone address space From/To address in Hex)
                               (If address less than board size, board offset used)
          -b <Board>           (board #, default = 0)
          -d <Delay>           (Delay between screen refresh -- default is 0
                                milli-seconds)
          -F DebugFile         (Debug file -- default "=== None ===")
             @DebugFile        (Debug file and no display)
             @                 (No Debug file and no display)
          -l LoopCnt           (Loop count -- default is 0)
          -P                   (Program Board)
          -q                   (Quite (non-interactive) mode)
          -s <XferSize>        (Transfer size bytes)
          -S <DisplaySize>     (Display size bytes)
          -v Delay             (Verify data. Add additional one cycle delay in
                                micro-seconds if specified)
          -X                   (Disable region protection)
          -Z                   (Disable address cache - default is enabled)
```

```
e.g.
  ./ccurpmfc_msgdma_clone                          (Clone DiagRam to physical memory
                                                    created by this program)
  ./ccurpmfc_msgdma_clone -a,-1                    (Clone DiagRam to physical memory
                                                    created by this program)
  ./ccurpmfc_msgdma_clone -a-1,-1 -v               (Clone physical memory to physical
                                                    memory created by this program
                                                    and verify)
  ./ccurpmfc_msgdma_clone -a,c000 -s0x1000 -v      (Clone DiagRam at 0x8000 to board
                                                    DiagRam at 0xc000)
  ./ccurpmfc_msgdma_clone -a8000,c000 -s0x4000 -v  (Clone DiagRam at 0x8000 to
                                                    DiagRam at 0xC000 and verify)
  ./ccurpmfc_msgdma_clone -a8000,-1 -s0x4000 -v    (Clone DiagRam at 0x8000 to
                                                    physical memory created and
                                                    verify)
  ./ccurpmfc_msgdma_clone -a-1,8000 -s0x4000 -v    (Clone physical memory created to
                                                    DiagRam at 0x8000 and verify)
  ./ccurpmfc_msgdma_clone -a,0xbd308000 -X         (Clone DiagRam to some other board
                                                    at specified physical address)
```

Example display:

./ccurpmfc_msgdma_clone -b1

```
Device Name                 : /dev/ccurpmfc1
Board ID                    : 9290
Board Type                  : 02
Board Function              : 30
Board Serial No             : 668603 (0x000a33bb)
Number of MsgDMA Descriptors: 31

Local Region  (BAR2) Size   : 0x00040000
Local Region  (BAR2) Address: 0xbd200000
Config Region (BAR0) Size   : 0x00008000
Config Region (BAR0) Address: 0xbd240000


>>>####### Processing 'From' Address (0x00008000) ######<<<

From_____________________________________
  TranslationRequired         = 0
  UserSuppliedPhysicalAddress = 0x00008000
  AvalonEquivalentAddress     = 0x00008000
  PhysicalMemoryToAttach      = 0xbd208000
  PhysicalMemorySize          = 0x00004000
  VirtualUserAddress          = 0x7ffff7f9c000
  Flags                       = 0x0000

>>>####### Processing 'To' Address (0xfffffffffffffffc) ######<<<
           Physical Memory Information:
             UserPID          =7996
             PhysMemPtr       =0x340ec000
             DriverVirtMemPtr =0xffff96d0f40ec000
             MmapedUserMemPtr =0x7ffff7fe9000
             PhysMemSize      =0x00004000
             PhysMemSizeFreed =0x00000000
             EntryInTxTbl     =0
             NumOfEntriesUsed =1
             Flags            =0x0000

To_______________________________________
```

```
  TranslationRequired         = 1
  UserSuppliedPhysicalAddress = 0x340ec000
  AvalonEquivalentAddress     = 0x008ec000
  PhysicalMemoryToAttach      = 0x340ec000
  PhysicalMemorySize          = 0x00004000
  VirtualUserAddress          = 0x7ffff7fe9000
  Flags                       = 0x0000

 Physical Address      [-a]: 0xBD208000/0x340EC000 (From/To)
 Board Number          [-b]: 1
 Delay                 [-d]: 0       (milli-seconds)
 Loop Count            [-l]: 0       (forever)
 Program Board         [-P]: 0       (no)
 Quiet Mode            [-q]: 0       (interactive)
 Transfer Size         [-s]: 16384   (bytes)
 Display Size          [-S]: 256     (bytes)
 Verify Data           [-v]: no
 Region Protection     [-X]: 1       (enabled)
 Address Cache         [-Z]: 0       (enabled)
 One Cycle Time            : 52.115  (micro-seconds)

 From_____________________________________  To_____________________________________
 TranslationRequired      = 0               TranslationRequired      = 1
 UserSuppliedPhysicalAddr = 0x00008000      UserSuppliedPhysicalAddr = 0x340ec000
 AvalonEquivalentAddress  = 0x00008000      AvalonEquivalentAddress  = 0x008ec000
 PhysicalMemoryToAttach   = 0xbd208000      PhysicalMemoryToAttach   = 0x340ec000
 PhysicalMemorySize       = 0x00004000      PhysicalMemorySize       = 0x00004000
 VirtualUserAddress       = 0x7ffff7f9c000  VirtualUserAddress       = 0x7ffff7fe9000
 Flags                    = 0x0000          Flags                    = 0x0000

ScanCount=1187 (XferSize=16384, DisplaySize=256) (CopyTime: From 12899.666 usecs,
                                                          To    3.836 usec)

___From___  ___00___ ___04___ ___08___ ___0C___ ___10___ ___14___ ___18___ ___1C___
0x00008000  007d0000 007d0001 007d0002 007d0003 007d0004 007d0005 007d0006 007d0007
0x00008020  007d0008 007d0009 007d000a 007d000b 007d000c 007d000d 007d000e 007d000f
0x00008040  007d0010 007d0011 007d0012 007d0013 007d0014 007d0015 007d0016 007d0017
0x00008060  007d0018 007d0019 007d001a 007d001b 007d001c 007d001d 007d001e 007d001f
0x00008080  007d0020 007d0021 007d0022 007d0023 007d0024 007d0025 007d0026 007d0027
0x000080a0  007d0028 007d0029 007d002a 007d002b 007d002c 007d002d 007d002e 007d002f
0x000080c0  007d0030 007d0031 007d0032 007d0033 007d0034 007d0035 007d0036 007d0037
0x000080e0  007d0038 007d0039 007d003a 007d003b 007d003c 007d003d 007d003e 007d003f

____To____  ___00___ ___04___ ___08___ ___0C___ ___10___ ___14___ ___18___ ___1C___
0x340ec000  007d0000 007d0001 007d0002 007d0003 007d0004 007d0005 007d0006 007d0007
0x340ec020  007d0008 007d0009 007d000a 007d000b 007d000c 007d000d 007d000e 007d000f
0x340ec040  007d0010 007d0011 007d0012 007d0013 007d0014 007d0015 007d0016 007d0017
0x340ec060  007d0018 007d0019 007d001a 007d001b 007d001c 007d001d 007d001e 007d001f
0x340ec080  007d0020 007d0021 007d0022 007d0023 007d0024 007d0025 007d0026 007d0027
0x340ec0a0  007d0028 007d0029 007d002a 007d002b 007d002c 007d002d 007d002e 007d002f
0x340ec0c0  007d0030 007d0031 007d0032 007d0033 007d0034 007d0035 007d0036 007d0037
0x340ec0e0  007d0038 007d0039 007d003a 007d003b 007d003c 007d003d 007d003e 007d003f
```

```
./ccurpmfc_msgdma_clone -b1 -a8000,-1 -s0x4000 -v
Device Name                 : /dev/ccurpmfc1
Board ID                    : 9290
Board Type                  : 02
Board Function              : 30
Board Serial No             : 668603 (0x000a33bb)
Number of MsgDMA Descriptors: 31
```

```
Local Region  (BAR2) Size   : 0x00040000
Local Region  (BAR2) Address: 0xbd200000
Config Region (BAR0) Size   : 0x00008000
Config Region (BAR0) Address: 0xbd240000


>>>####### Processing 'From' Address (0x00008000) ######<<<

From_____________________________________
  TranslationRequired         = 0
  UserSuppliedPhysicalAddress = 0x00008000
  AvalonEquivalentAddress     = 0x00008000
  PhysicalMemoryToAttach      = 0xbd208000
  PhysicalMemorySize          = 0x00004000
  VirtualUserAddress          = 0x7ffff7f9c000
  Flags                       = 0x0000

>>>####### Processing 'To' Address (0xfffffffffffffffc) ######<<<
           Physical Memory Information:
             UserPID          =8510
             PhysMemPtr       =0x35340000
             DriverVirtMemPtr =0xffff96d0f5340000
             MmapedUserMemPtr =0x7ffff7fe9000
             PhysMemSize      =0x00004000
             PhysMemSizeFreed =0x00000000
             EntryInTxTbl     =0
             NumOfEntriesUsed =1
             Flags            =0x0000

To_______________________________________
  TranslationRequired         = 1
  UserSuppliedPhysicalAddress = 0x35340000
  AvalonEquivalentAddress     = 0x00840000
  PhysicalMemoryToAttach      = 0x35340000
  PhysicalMemorySize          = 0x00004000
  VirtualUserAddress          = 0x7ffff7fe9000
  Flags                       = 0x0000

 Physical Address      [-a]: 0xBD208000/0x35340000 (From/To)
 Board Number          [-b]: 1
 Delay                 [-d]: 0       (milli-seconds)
 Loop Count            [-l]: 0       (forever)
 Program Board         [-P]: 0       (no)
 Quiet Mode            [-q]: 0       (interactive)
 Transfer Size         [-s]: 16384   (bytes)
 Display Size          [-S]: 256     (bytes)
 Verify Data           [-v]: 0.000   (Additional One Cycle Delay in micro-seconds)
 Region Protection     [-X]: 1       (enabled)
 Address Cache         [-Z]: 0       (enabled)
 One Cycle Time            : 52.260  (micro-seconds)

 From_____________________________________  To_____________________________________
 TranslationRequired      = 0               TranslationRequired      = 1
 UserSuppliedPhysicalAddr = 0x00008000      UserSuppliedPhysicalAddr = 0x35340000
 AvalonEquivalentAddress  = 0x00008000      AvalonEquivalentAddress  = 0x00840000
 PhysicalMemoryToAttach   = 0xbd208000      PhysicalMemoryToAttach   = 0x35340000
 PhysicalMemorySize       = 0x00004000      PhysicalMemorySize       = 0x00004000
 VirtualUserAddress       = 0x7ffff7f9c000  VirtualUserAddress       = 0x7ffff7fe9000
 Flags                    = 0x0000          Flags                    = 0x0000

FailCount=0          (==== passed ====)
```

```
ScanCount=405 (XferSize=16384, DisplaySize=256) (WaitAfterPatternWrite: 52.560 usecs)

___From___  ___00___ ___04___ ___08___ ___0C___ ___10___ ___14___ ___18___ ___1C___
0x00008000  00194000 00194001 00194002 00194003 00194004 00194005 00194006 00194007
0x00008020  00194008 00194009 0019400a 0019400b 0019400c 0019400d 0019400e 0019400f
0x00008040  00194010 00194011 00194012 00194013 00194014 00194015 00194016 00194017
0x00008060  00194018 00194019 0019401a 0019401b 0019401c 0019401d 0019401e 0019401f
0x00008080  00194020 00194021 00194022 00194023 00194024 00194025 00194026 00194027
0x000080a0  00194028 00194029 0019402a 0019402b 0019402c 0019402d 0019402e 0019402f
0x000080c0  00194030 00194031 00194032 00194033 00194034 00194035 00194036 00194037
0x000080e0  00194038 00194039 0019403a 0019403b 0019403c 0019403d 0019403e 0019403f

____To____  ___00___ ___04___ ___08___ ___0C___ ___10___ ___14___ ___18___ ___1C___
0x35340000  00194000 00194001 00194002 00194003 00194004 00194005 00194006 00194007
0x35340020  00194008 00194009 0019400a 0019400b 0019400c 0019400d 0019400e 0019400f
0x35340040  00194010 00194011 00194012 00194013 00194014 00194015 00194016 00194017
0x35340060  00194018 00194019 0019401a 0019401b 0019401c 0019401d 0019401e 0019401f
0x35340080  00194020 00194021 00194022 00194023 00194024 00194025 00194026 00194027
0x353400a0  00194028 00194029 0019402a 0019402b 0019402c 0019402d 0019402e 0019402f
0x353400c0  00194030 00194031 00194032 00194033 00194034 00194035 00194036 00194037
0x353400e0  00194038 00194039 0019403a 0019403b 0019403c 0019403d 0019403e 0019403f
```

3.2.20 lib/ccurpmfc_msgdma_info

This test provides useful modular scatter-gather DMA information for cards that support it.

```
Usage: ./ccurpmfc_msgdma_info [-b Board] [-l]
          -b <Board>            (board #, default = 0)
          -l                    (long format)
```

Example display:

./ccurpmfc_msgdma_info

```
============= Dispatcher ==============
 Status                     = 0x0000000a
 Control                    = 0x0000000c
 ReadFillLevel              = 0x00000000
 WriteFillLevel             = 0x00000000
 ResponseFillLevel          = 0x00000000
 ReadSequenceNumber         = 0x00000001
 WriteSequenceNumber        = 0x00000001

 ============= Prefetcher ==============
 Status                     = 0x00000000
 Control                    = 0x00000000
 NextDescriptorPointer      = 0xbaadbeef00004800 (### Descriptor ID 1 ###)
 DescriptorPollingFrequency = 0x00000000

========================================== Descriptors ==========================================
 ID  Addr   ReadAddr WritAddr Length Stat Control  SeqN Rbct Wbct Rstr Wstr ActBytXfr NextPtr
 ==  ====   ======== ======== ====== ==== ======== ==== ==== ==== ==== ==== ========= ===========
  1 (4800): 00008000 00870000 008000 0000 80000000 00   00   00   0000 0000 00000000  00004fc0
                                                                                      (Terminator)
  2 (4840): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
  3 (4880): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
  4 (48c0): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
  5 (4900): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
  6 (4940): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
  7 (4980): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
  8 (49c0): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
  9 (4a00): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
 10 (4a40): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
```

```
11 (4a80): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
12 (4ac0): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
13 (4b00): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
14 (4b40): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
15 (4b80): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
16 (4bc0): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
17 (4c00): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
18 (4c40): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
19 (4c80): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
20 (4cc0): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
21 (4d00): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
22 (4d40): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
23 (4d80): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
24 (4dc0): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
25 (4e00): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
26 (4e40): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
27 (4e80): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
28 (4ec0): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
29 (4f00): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
30 (4f40): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
31 (4f80): 00000000 00000000 000000 0000 00000000 00   00   00   0000 0000 00000000  00000000
```

./ccurpmfc_msgdma_info -l

```
============= Dispatcher ==============
 Status                     = 0x0000000a
 Control                    = 0x0000000c
 ReadFillLevel              = 0x00000000
 WriteFillLevel             = 0x00000000
 ResponseFillLevel          = 0x00000000
 ReadSequenceNumber         = 0x00000004
 WriteSequenceNumber        = 0x00000004

 ============= Prefetcher ==============
 Status                     = 0x00000001
 Control                    = 0x00000000
 NextDescriptorPointer      = 0x0000000000004800 (### Descriptor ID 1 ###)
 DescriptorPollingFrequency = 0x00000000

 ============ Descriptor ID 1 (address: 0x4800) =============
 ReadAddress                = 0x0000000000b80000
 WriteAddress               = 0x0000000000008000
 NextDescriptorPointer      = 0x0000000000004840 (### Descriptor ID 2 ###)
 Status                     = 0x0000
 Control                    = 0x80000000
 SequenceNumber             = 0x0001     (1)
 Length                     = 0x00003000 (12288)
 ReadBurstCount             = 0x00       (0)
 WriteBurstCount            = 0x00       (0)
 ReadStride                 = 0x0000     (0)
 WriteStride                = 0x0000     (0)
 ActualBytesTransferred     = 0x00000000 (0)

 ============ Descriptor ID 2 (address: 0x4840) =============
 ReadAddress                = 0x0000000000008000
 WriteAddress               = 0x0000000000b83000
 NextDescriptorPointer      = 0x0000000000004880 (### Descriptor ID 3 ###)
 Status                     = 0x0000
 Control                    = 0x80000000
 SequenceNumber             = 0x0002     (2)
 Length                     = 0x00003000 (12288)
 ReadBurstCount             = 0x00       (0)
 WriteBurstCount            = 0x00       (0)
```

```
 ReadStride                 = 0x0000     (0)
 WriteStride                = 0x0000     (0)
 ActualBytesTransferred     = 0x00000000 (0)

 ============ Descriptor ID 3 (address: 0x4880) =============
 ReadAddress                = 0x0000000000b83000
 WriteAddress               = 0x000000000000b000
 NextDescriptorPointer      = 0x00000000000048c0 (### Descriptor ID 4 ###)
 Status                     = 0x0000
 Control                    = 0x80000000
 SequenceNumber             = 0x0003     (3)
 Length                     = 0x00003000 (12288)
 ReadBurstCount             = 0x00       (0)
 WriteBurstCount            = 0x00       (0)
 ReadStride                 = 0x0000     (0)
 WriteStride                = 0x0000     (0)
 ActualBytesTransferred     = 0x00000000 (0)

============ Descriptor ID 4 (address: 0x48c0) =============
 ReadAddress                = 0x000000000000b000
 WriteAddress               = 0x0000000000b86000
 NextDescriptorPointer      = 0x0000000000004fc0 (### Terminator ###)
 Status                     = 0x0000
 Control                    = 0x80000000
 SequenceNumber             = 0x0004     (4)
 Length                     = 0x00003000 (12288)
 ReadBurstCount             = 0x00       (0)
 WriteBurstCount            = 0x00       (0)
 ReadStride                 = 0x0000     (0)
 WriteStride                = 0x0000     (0)
 ActualBytesTransferred     = 0x00000000 (0)

 ============ Descriptor ID 5 (address: 0x4900) =============
 ReadAddress                = 0x0000000000000000
 WriteAddress               = 0x0000000000000000
 NextDescriptorPointer      = 0x0000000000000000
 Status                     = 0x0000
 Control                    = 0x00000000
 SequenceNumber             = 0x0000     (0)
 Length                     = 0x00000000 (0)
 ReadBurstCount             = 0x00       (0)
 WriteBurstCount            = 0x00       (0)
 ReadStride                 = 0x0000     (0)
 WriteStride                = 0x0000     (0)
 ActualBytesTransferred     = 0x00000000 (0)
.
.
.
============ Descriptor ID 30 (address: 0x4f40) =============
 ReadAddress                = 0x0000000000000000
 WriteAddress               = 0x0000000000000000
 NextDescriptorPointer      = 0x0000000000000000
 Status                     = 0x0000
 Control                    = 0x00000000
 SequenceNumber             = 0x0000     (0)
 Length                     = 0x00000000 (0)
 ReadBurstCount             = 0x00       (0)
 WriteBurstCount            = 0x00       (0)
 ReadStride                 = 0x0000     (0)
 WriteStride                = 0x0000     (0)
 ActualBytesTransferred     = 0x00000000 (0)
```

```
============ Descriptor ID 31 (address: 0x4f80) =============
ReadAddress                = 0x0000000000000000
WriteAddress               = 0x0000000000000000
NextDescriptorPointer      = 0x0000000000000000
Status                     = 0x0000
Control                    = 0x00000000
SequenceNumber             = 0x0000     (0)
Length                     = 0x00000000 (0)
ReadBurstCount             = 0x00       (0)
WriteBurstCount            = 0x00       (0)
ReadStride                 = 0x0000     (0)
WriteStride                = 0x0000     (0)
ActualBytesTransferred     = 0x00000000 (0)
```

3.2.21 lib/ccurpmfc_msgdma_multi_clone

This test is a more powerful version of the *ccurpmfc_msgdma_clone* test above. It allows the users to specify multiple source and destination addresses during the cloning operation. There is a limit on the number of physical memory that can be created or mapped. When that limit is reached, the tests fail to run. Additionally, there is a limit to the number ofMsgDmadescriptors that can be specified. Once again, if that limit is exceed the test will fail to run.

<u>Caution</u>: Since physical addresses are supplied to this test, care must be taken to ensure that the supplied addresses are valid and that while cloning is in progress, the memory regions must not be freed or made inactive, otherwise, the results could be unpredictable and could lead to the possible corruption of the system.

```
Usage: ./ccurpmfc_msgdma_multi_clone [-a FromAddr,ToAddr,Size] [-b Board] [-d Delay]
                                     [-F DebugFile] [-l LoopCnt] [-P] [-q] [-s XferSize]
                                     [-S DisplaySize] [-t|T FromAddr,Size] [-X] [-Z]
          -a <FromAddr,ToAddr,Size> (Clone address space From/To address (hex) and
                                     size (bytes))
                                    (If address less than board size, board offset used)
          -b <Board>                (board #, default = 0)
          -d <Delay>                (Delay between screen refresh -- default is
                                     5 milli-seconds)
          -F DebugFile              (Debug file -- default "=== None ===")
             @DebugFile             (Debug file and no display)
             @                      (No Debug file and no display)
          -l LoopCnt                (Loop count -- default is 0)
          -P                        (Program Board)
          -q                        (Quite (non-interactive) mode)
          -s <XferSize>             (Transfer size in bytes)
          -S <DisplaySize>          (Display size in bytes)
          -t <FromAddr,Size>        (Perform debug firmware testing - From address (hex)
                                     and size (bytes) - non-verbose)
          -T <FromAddr,Size>        (Perform debug firmware testing - From address (hex)
                                     and size (bytes) - verbose)
          -X                        (Disable region protection)
          -Z                        (Disable address cache - default is enabled)

Note: If the size is not specifed in the '-a' option, then the default size or that
      specifed in the '-s' option is used
      If debug firmware is installed, you can use the '-t|T' option
      The debug firmware uses Clock 3 and its value is programmed by the 't|T' option

e.g.
  ./ccurpmfc_msgdma_multi_clone                    (Clone DiagRam to physical memory
```

```
                                                    created by this program)
  ./ccurpmfc_msgdma_multi_clone -a,-1               (Clone DiagRam to physical memory
                                                    created by this program)
  ./ccurpmfc_msgdma_multi_clone -a-1,-1             (Clone physical memory to physical
                                                    memory created by this program)
  ./ccurpmfc_msgdma_multi_clone -a,c000 -s0x1000    (Clone DiagRam at 0x8000 to board
                                                    DiagRam at 0xc000)
  ./ccurpmfc_msgdma_multi_clone -a8000,c000 -s0x4000 (Clone DiagRam at 0x8000 to DiagRam
                                                    at 0xC000)
  ./ccurpmfc_msgdma_multi_clone -a,0xbd308000 -X    (Clone DiagRam to some other board
                                                    at specified physical address)
  ./ccurpmfc_msgdma_multi_clone -a8000,8800 -a8800,9000 -a9000,9800 -a9800,a000
                               -aa000,a800 -aa800,b000 -ab000,b800 -ab800,c000
                               -ac000,c800 -s256
                                                    (Clone through all 9 descriptors)
  ./ccurpmfc_msgdma_multi_clone -a8000,8800 -a8800,9000 -a9000,9800 -a9800,a000,128
                              -aa000,a800 -aa800,b000 -ab000,b800 -ab800,c000
                              -ac000,c800,288 -s256 -S288
                                                    (Clone through all 9 descriptors
                                                    with different sizes)
  ./ccurpmfc_msgdma_multi_clone -t                  (Perform non-verbose Debug Firmware
                                                    testing using default address
                                                    0x6000 and size 256 (64 samples))
  ./ccurpmfc_msgdma_multi_clone -T,256 -F/tmp/LOG   (Perform verbose Debug Firmware
                                                    testing using default address, 256
                                                    bytes and send output to /tmp/LOG)
```

Example display:

./ccurpmfc_msgdma_multi_clone

```
 Physical Address      [-a]: 0xC0608000/0x35FC8000 (From/To)
 Board Number          [-b]: 0
 Delay                 [-d]: 5       (milli-seconds)
 Loop Count            [-l]: 0       (forever)
 Program Board         [-P]: 0       (no)
 Quiet Mode            [-q]: 0       (interactive)
 Transfer Size         [-s]: 16384   (bytes)
 Display Size          [-S]: 256     (bytes)
 Region Protection     [-X]: 1       (enabled)
 Address Cache         [-Z]: 0       (enabled)
 One Cycle Time            : 52.962  (micro-seconds)
 Current Descriptor        : 0

 From_____________________________________  To_____________________________________
 TranslationRequired      = 0               TranslationRequired      = 1
 UserSuppliedPhysicalAddr = 0x00008000      UserSuppliedPhysicalAddr = 0x35fc8000
 UserSuppliedSize         = 0x00004000      UserSuppliedSize         = 0x00004000
 AvalonEquivalentAddress  = 0x00008000      AvalonEquivalentAddress  = 0x008c8000
 PhysicalMemoryToAttach   = 0xc0608000      PhysicalMemoryToAttach   = 0x35fc8000
 PhysicalMemorySize       = 0x00004000      PhysicalMemorySize       = 0x00004000
 VirtualUserAddress       = 0x7ffff7f9c000  VirtualUserAddress       = 0x7ffff7fe9000
 Flags                    = 0x0000          Flags                    = 0x0000

ScanCount=475        (XferSize=16384, DisplaySize=256) (CopyTime: From 13284.239 usec,
                                                                  To    3.681 usec)

___From___  ___00___ ___04___ ___08___ ___0C___ ___10___ ___14___ ___18___ ___1C___
0x00008000  00135000 00135001 00135002 00135003 00135004 00135005 00135006 00135007
0x00008020  00135008 00135009 0013500a 0013500b 0013500c 0013500d 0013500e 0013500f
```

```
0x00008040  00135010 00135011 00135012 00135013 00135014 00135015 00135016 00135017
0x00008060  00135018 00135019 0013501a 0013501b 0013501c 0013501d 0013501e 0013501f
0x00008080  00135020 00135021 00135022 00135023 00135024 00135025 00135026 00135027
0x000080a0  00135028 00135029 0013502a 0013502b 0013502c 0013502d 0013502e 0013502f
0x000080c0  00135030 00135031 00135032 00135033 00135034 00135035 00135036 00135037
0x000080e0  00135038 00135039 0013503a 0013503b 0013503c 0013503d 0013503e 0013503f

____To____  ___00___ ___04___ ___08___ ___0C___ ___10___ ___14___ ___18___ ___1C___
0x35fc8000  00135000 00135001 00135002 00135003 00135004 00135005 00135006 00135007
0x35fc8020  00135008 00135009 0013500a 0013500b 0013500c 0013500d 0013500e 0013500f
0x35fc8040  00135010 00135011 00135012 00135013 00135014 00135015 00135016 00135017
0x35fc8060  00135018 00135019 0013501a 0013501b 0013501c 0013501d 0013501e 0013501f
0x35fc8080  00135020 00135021 00135022 00135023 00135024 00135025 00135026 00135027
0x35fc80a0  00135028 00135029 0013502a 0013502b 0013502c 0013502d 0013502e 0013502f
0x35fc80c0  00135030 00135031 00135032 00135033 00135034 00135035 00135036 00135037
0x35fc80e0  00135038 00135039 0013503a 0013503b 0013503c 0013503d 0013503e 0013503f

Enter 'c|C' to clear the pattern
      'w|W' toggle pattern write  (Pattern Write Enabled)
      'q|Q' ->
```

3.2.22 lib/ccurpmfc_smp_affinity

This test provides a useful mechanism to display or set the IRQ to specific set of CPUs. This is useful when we want to make sure that the driver interrupts are not being interfered with other CPU activity.

```
Usage: ./ccurpmfc_smp_affinity [-b Board] [-c CpuMask]
 -b Board           (Board number -- default is 0)
 -c CpuMask         (CPU mask in HEX -- default is none)

 e.g. ./ccurpmfc_smp_affinity        (display IRQ CPU mask for selected board)
      ./ccurpmfc_smp_affinity -c2    (set IRQ CPU for cpu 1)
      ./ccurpmfc_smp_affinity -c4    (set IRQ CPU for cpu 2)
      ./ccurpmfc_smp_affinity -c0x8  (set IRQ CPU for cpu 3)
      ./ccurpmfc_smp_affinity -cE2   (set IRQ CPU for cpu 1,5,6,7)
```

Example display:

./ccurpmfc_smp_affinity
(IRQ57) fc user f8 actual

./ccurpmfc_smp_affinity -b1 -c8
(IRQ57) 08 user 08 actual

3.2.23 lib/ccurpmfc_transfer

This test performs various DMA and Programmed I/O transfers between the board components and the PCI memory.

```
Usage: ./ccurpmfc_transfer [-b Board] [-c CaseNum] [-i] [-l LoopCnt]
                           [-s XferSize]
 -b Board     (Board number -- default is 0)
 -c CaseNum   (Select Case Numbers -- default = ALL CASES)
    -c 4,1,7-9 select case 1,4,7,8,9)
    -c 8-      select case 8 to end)
    -c -3      select case 1,2,3)
 -i           (Enable Interrupts -- default = Disable)
 -l LoopCnt   (Loop Count -- default is 100)
 -s XferSize  (Avalon Ram Xfer Size in bytes -- default is 32768)
```

Example display:

./ccurpmfc_transfer (for cards with modular scatter-gather DMA support)

```
local_ptr=0x7ffff7f97000
Size of Avalon RAM = 32768 (0x00008000)
           Physical Memory Information:
             UserPID         =20130
             PhysMemPtr      =0x70198000
             DriverVirtMemPtr=0xffff95bbb0198000
             MmapedUserMemPtr=0x7ffff7fe5000
             PhysMemSize     =0x00008000
             PhysMemSizeFreed=0x00000000
             EntryInTxTbl    =0
             NumOfEntriesUsed=1
             Flags           =0x0000

  1: Memory -> Avalon RAM (DMA0) (Size=0x8000):           100 (1074.47 us, 30.50 MBytes/Sec)
  2: Memory -> Avalon RAM (DMA1) (Size=0x8000):           100 (1074.77 us, 30.49 MBytes/Sec)
  3: Memory -> Avalon RAM (MSGDMA) (Size=0x8000):         100 (109.54 us, 299.14 MBytes/Sec)
  4: Memory -> Avalon RAM  (PIO) (Size=0x8000):           100 (987.92 us, 33.17 MBytes/Sec)
  5: Avalon RAM -> Memory (DMA0) (Size=0x8000):           100 (1854.17 us, 17.67 MBytes/Sec)
  6: Avalon RAM -> Memory (DMA1) (Size=0x8000):           100 (1854.26 us, 17.67 MBytes/Sec)
  7: Avalon RAM -> Memory (MSGDMA) (Size=0x8000):         100 (108.60 us, 301.73 MBytes/Sec)
  8: Avalon RAM -> Memory  (PIO) (Size=0x8000):           100 (7346.86 us, 4.46 MBytes/Sec)
  9: Memory -> Avalon ADC Calibration (DMA0) (Size=0x40): 10000 (8.88 us, 7.21 MBytes/Sec))
 10: Memory -> Avalon ADC Calibration (DMA1) (Size=0x40): 10000 (8.80 us, 7.27 MBytes/Sec)
 11: Memory -> Avalon ADC Calibration  (PIO) (Size=0x40): 10000 (0.69 us, 92.85 MBytes/Sec)
 12: Avalon ADC Calibration -> Memory (DMA0) (Size=0x40): 10000 (8.37 us, 7.65 MBytes/Sec)
 13: Avalon ADC Calibration -> Memory (DMA1) (Size=0x40): 10000 (8.58 us, 7.46 MBytes/Sec)
 14: Avalon ADC Calibration -> Memory  (PIO) (Size=0x40): 10000 (19.91 us, 3.21 MBytes/Sec)
 15: Memory -> Avalon SDRAM(FIFO) (DMA0) (Size=0x8000):   100 (1074.60 us, 30.49 MBytes/Sec)
 16: Memory -> Avalon SDRAM(FIFO) (DMA1) (Size=0x8000):   100 (1074.81 us, 30.49 MBytes/Sec)
 17: Memory -> Avalon SDRAM(FIFO)  (PIO) (Size=0x8000):   100 (997.24 us, 32.86 MBytes/Sec))
 18: Avalon SDRAM(FIFO) -> Memory (DMA0) (Size=0x8000):   100 (1996.39 us, 16.41 MBytes/Sec)
 19: Avalon SDRAM(FIFO) -> Memory (DMA1) (Size=0x8000):   100 (1996.46 us, 16.41 MBytes/Sec)
 20: Avalon SDRAM(FIFO) -> Memory  (PIO) (Size=0x8000):   100 (11406.91 us, 2.87 MBytes/Sec)
**** Test Passed ****
```

./ccurpmfc_transfer (for cards without modular scatter-gather DMA support)

```
local_ptr=0x7ffff7fd7000
local_ptr=0x7ffff7f97000
Size of Avalon RAM = 32768 (0x00008000)
           Physical Memory Information:
             UserPID         =20625
             PhysMemPtr      =0x70198000
             DriverVirtMemPtr=0xffff95bbb0198000
             MmapedUserMemPtr=0x7ffff7fe5000
             PhysMemSize     =0x00008000
             PhysMemSizeFreed=0x00000000
             EntryInTxTbl    =0
             NumOfEntriesUsed=1
             Flags           =0x0000

  1: Memory -> Avalon RAM (DMA0) (Size=0x8000):           100 (1137.05 us, 28.82 MBytes/Sec)
  2: Memory -> Avalon RAM (DMA1) (Size=0x8000):           100 (1136.70 us, 28.83 MBytes/Sec)
  3: Memory -> Avalon RAM  (PIO) (Size=0x8000):           100 (695.42 us, 47.12 MBytes/Sec)
  4: Avalon RAM -> Memory (DMA0) (Size=0x8000):           100 (1857.79 us, 17.64 MBytes/Sec)
  5: Avalon RAM -> Memory (DMA1) (Size=0x8000):           100 (1857.76 us, 17.64 MBytes/Sec)
  6: Avalon RAM -> Memory  (PIO) (Size=0x8000):           100 (4715.11 us, 6.95 MBytes/Sec)
  7: Memory -> Avalon ADC Calibration (DMA0) (Size=0x40): 10000 (8.67 us, 7.38 MBytes/Sec)
  8: Memory -> Avalon ADC Calibration (DMA1) (Size=0x40): 10000 (8.64 us, 7.41 MBytes/Sec)
  9: Memory -> Avalon ADC Calibration  (PIO) (Size=0x40): 10000 (0.11 us, 589.24 MBytes/Sec)
 10: Avalon ADC Calibration -> Memory (DMA0) (Size=0x40): 10000 (8.69 us, 7.37 MBytes/Sec)
 11: Avalon ADC Calibration -> Memory (DMA1) (Size=0x40): 10000 (8.69 us, 7.36 MBytes/Sec)
 12: Avalon ADC Calibration -> Memory  (PIO) (Size=0x40): 10000 (10.77 us, 5.94 MBytes/Sec)
```

```
 13: Memory -> Avalon SDRAM(FIFO) (DMA0) (Size=0x8000):   100 (1136.93 us, 28.82 MBytes/Sec)
 14: Memory -> Avalon SDRAM(FIFO) (DMA1) (Size=0x8000):   100 (1136.76 us, 28.83 MBytes/Sec)
 15: Memory -> Avalon SDRAM(FIFO)  (PIO) (Size=0x8000):   100 (1412.14 us, 23.20 MBytes/Sec)
 16: Avalon SDRAM(FIFO) -> Memory (DMA0) (Size=0x8000):   100 (2002.71 us, 16.36 MBytes/Sec)
 17: Avalon SDRAM(FIFO) -> Memory (DMA1) (Size=0x8000):   100 (2002.95 us, 16.36 MBytes/Sec)
 18: Avalon SDRAM(FIFO) -> Memory  (PIO) (Size=0x8000):   100 (11354.51 us, 2.89 MBytes/Sec)
**** Test Passed ****
```

3.2.24 lib/ccurpmfc_tst_lib

This is an interactive test that accesses the various supported API calls.

```
Usage: ./ccurpmfc_tst_lib [-b board]
 -b board: board number -- default board is 0
```

Example display:

./ccurpmfc_tst_lib

```
Device Name: /dev/ccurpmfc0
  01 = Abort DMA                            02 = Clear Driver Error
  03 = Clear Library Error                  04 = Display BOARD Registers
  05 = Display CONFIG Registers             06 = Dump Physical Memory List
  07 = Get All Boards Driver Information    08 = Get Board CSR
  09 = Get Board Information                10 = Get Driver Error
  11 = Get Driver Information               12 = Get Library Error
  13 = Get Mapped Config Pointer            14 = Get Mapped Driver/Library Pointer
  15 = Get Mapped Local Pointer             16 = Get Physical Memory
  17 = Get Test Bus Control                 18 = Get Value
  19 = Initialize Board                     20 = MMap Physical Memory
  21 = Munmap Physical Memory               22 = Reload Firmware
  23 = Reset Board                          24 = Set Board CSR
  25 = Set Test Bus Control                 26 = Set Value
  27 = ### ADC MENU ###                     28 = ### CALIBRATION MENU ###
  29 = ### CLOCK GENERATOR MENU ###         30 = ### DAC MENU ###
  31 = ### DIO MENU ###                     32 = ### INTERRUPT MENU ###
  33 = ### IP CORE MENU ###                 34 - ### SDRAM MENU ###
  35 = ### SPROM MENU ###

Main Selection ('h'=display menu, 'q'=quit)->
```

```
Main Selection ('h'=display menu, 'q'=quit)-> 27
   Command: ADC_menu()
  01 = ADC Activate                         02 = ADC Disable
  03 = ADC Reset (disable followed by enable)  04 = ADC Driver Read Operation
  05 = ADC Get CSR                          06 = ADC Get Driver Read Mode
  07 = ADC Get FIFO Channel Select          08 = ADC Get FIFO Information
  09 = ADC Get FIFO Threshold               10 = ADC Read Channels
  11 = ADC Reset FIFO                       12 = ADC Set CSR
  13 = ADC Set Driver Read Mode             14 = ADC Set FIFO Channel Select
  15 = ADC Set FIFO Threshold

ADC Selection ('h'=display menu, 'q'=quit)->
```

```
Main Selection ('h'=display menu, 'q'=quit)-> 28
   Command: calibration_menu()
  01 = ADC: Get Calibrated Values           02 = ADC: Perform Auto Calibration
  03 = ADC: Perform External Negative Calib.   04 = ADC: Perform External Offset Calib.
  05 = ADC: Perform External Positive Calib.   06 = ADC: Perform Negative Calibration
  07 = ADC: Perform Offset Calibration      08 = ADC: Perform Positive Calibration
  09 = ADC: Read Channels Calibration       10 = ADC: Reset Calibration
  11 = ADC: Write Channels Calibration      12 = DAC: Get Calibrated Values
  13 = DAC: Perform Auto Calibration        14 = DAC: Perform Gain Calibration
  15 = DAC: Perform Offset Calibration      16 = DAC: Read Channels Calibration
```

```
  17 = DAC: Reset Calibration                 18 = DAC: Write Channels Calibration
  19 = Get Calibration CSR                    20 = Set Calibration CSR

Calibration Selection ('h'=display menu, 'q'=quit)->
```

```
Main Selection ('h'=display menu, 'q'=quit)-> 29
   Command: clock_generator_menu()
  01 = Clock Get Generator CSR                02 = Clock Get Generator Dividers
  03 = Clock Get Generator Information        04 = Clock Get Generator Input Clock Enable
  05 = Clock Get Generator Input Clock Select 06 = Clock Get Generator Input Clock Status
  07 = Clock Get Generator Output Config      08 = Clock Get Generator Output Format
  09 = Clock Get Generator Output Mode        10 = Clock Get Generator Output Mux
  11 = Clock Get Generator P-Divider Enable   12 = Clock Get Generator Revision
  13 = Clock Get Generator Value              14 = Clock Get Generator Voltage Select
  15 = Clock Get Generator Zero Delay         16 = Clock Set Generator CSR
  17 = Clock Set Generator Dividers           18 = Clock Set Generator Input Clock Enable
  19 = Clock Set Generator Input Clock Select 20 = Clock Set Generator Output Config
  21 = Clock Set Generator Output Format      22 = Clock Set Generator Output Mode
  23 = Clock Set Generator Output Mux         24 = Clock Set Generator P-Divider Enable
  25 = Clock Set Generator Value              26 = Clock Set Generator Voltage Select
  27 = Clock Set Generator Zero Delay         28 = Compute All Output Clocks
  29 = Program All Output Clocks              30 = Read Clock Registers
  31 = Reset Clock (Hardware)                 32 = Soft Reset
  33 = Update Clock Generator Divider         34 = Write Clock Registers

Clock Generator Selection ('h'=display menu, 'q'=quit)->
```

```
Main Selection ('h'=display menu, 'q'=quit)-> 30
   Command: DAC_menu()
  01 = DAC Activate                           02 = DAC Disable
  03 = DAC Reset (disable followed by enable) 04 = DAC Driver Write Operation
  05 = DAC Get CSR                            06 = DAC Get Driver Write Mode
  07 = DAC Get FIFO Channel Select            08 = DAC Get FIFO Information
  09 = DAC Get FIFO Threshold                 10 = DAC Get FIFO Write Count
  11 = DAC Get Update Select                  12 = DAC ReadBack Channels
  13 = DAC Read Channels                      14 = DAC Reset FIFO
  15 = DAC Set CSR                            16 = DAC Set Driver Write Mode
  17 = DAC Set FIFO Channel Select            18 = DAC Set FIFO Threshold
  19 = DAC Set FIFO Write Count               20 = DAC Set Update Select
  21 = DAC Write Channels

DAC Selection ('h'=display menu, 'q'=quit)->
```

```
Main Selection ('h'=display menu, 'q'=quit)-> 31
   Command: DIO_menu()
  01 = DIO Activate                           02 = DIO Disable
  03 = DIO Reset (disable followed by enable) 04 = DIO Get Channels Polarity
  05 = DIO Get COS Channels Edge Sense        06 = DIO Get COS Channels Enable
  07 = DIO Get COS Channels Mode              08 = DIO Get COS Channels Overflow
  09 = DIO Get COS Channels Status            10 = DIO Get Input Channels Filter
  11 = DIO Get Input Snapshot                 12 = DIO Get Mode
  13 = DIO Get Output Sync                    14 = DIO Get Ports Direction
  15 = DIO Information                        16 = DIO Read Custom Channels
  17 = DIO Read Input Channels                18 = DIO Read Output Channels
  19 = DIO Set Channels Polarity              20 = DIO Set COS Channels Edge Sense
  21 = DIO Set COS Channels Enable            22 = DIO Set COS Channels Mode
  23 = DIO Set Input Channels Filter          24 = DIO Set Input Snapshot
  25 = DIO Set Mode                           26 = DIO Set Output Sync
  27 = DIO Set Ports Direction                28 = DIO Set Ports Direction to Inputs
  29 = DIO Set Ports Direction to Outputs     30 = DIO Write Output Channels
  31 = DIO Write Output Channels High         32 = DIO Write Output Channels Low

DIO Selection ('h'=display menu, 'q'=quit)->
```

```
Main Selection ('h'=display menu, 'q'=quit)-> 32
   Command: interrupt_menu()
  01 = Add Irq                                02 = Disable Pci Interrupts
  03 = Enable Pci Interrupts                  04 = Get Interrupt Status
  05 = Get Interrupt Timeout                  06 = Remove Irq
```

```
  07 = Set Interrupt Status                   08 = Set Interrupt Timeout

Interrupt Selection ('h'=display menu, 'q'=quit)->
_____________________________________________________________________________________
Main Selection ('h'=display menu, 'q'=quit)-> 33
   Command: IPCORE_menu()
  01 = IpCore COS Activate                    02 = IpCore COS Configure
  03 = IpCore COS Disable                     04 = IpCore COS Get Information
  05 = IpCore COS Read                        06 = IpCore COS Read (AGAIN)
  07 = IpCore COS Start/Stop Capture          08 = IpCore Get Ip Information
  09 = IpCore Get Ip Mapped Pointer

IP Core Selection ('h'=display menu, 'q'=quit)->
_____________________________________________________________________________________
Main Selection ('h'=display menu, 'q'=quit)-> 34
   Command: SDRAM_menu()
  01 = SDRAM Activate                         02 = SDRAM Disable
  03 = SDRAM Get CSR                          04 = SDRAM Read
  05 = SDRAM Set CSR                          06 = SDRAM Write

SDRAM Selection ('h'=display menu, 'q'=quit)->
_____________________________________________________________________________________
Main Selection ('h'=display menu, 'q'=quit)-> 35
   Command: SPROM_menu()
  01 = Clear Serial Prom                      02 = Read Serial Prom
  03 = Read Serial Prom Item                  04 = Write Override
  05 = Write Serial Prom                      06 = Write Serial Prom Item

SPROM Selection ('h'=display menu, 'q'=quit)->
_____________________________________________________________________________________
```

3.2.25 lib/IpCore/ccurpmfc_ipcore_cos

This program demonstrates the features of the IP Core Change of State card.

```
Usage: ./ccurpmfc_ipcore_cos [-b Board] [-c ChannelMask] [-d Delay]
                             [-D DMAEngineNo] [-F DebugFile] [-l LoopCnt]
                             [-m XferMode] [-n Element2Display] [-T TestNumber]
                             [-V VerifyPattern] [-w Control] [-x]
 -b Board           (Board number -- default board is 0)
 -c ChannelsMask    (select channels mask)
    -c0@XXXXXXXX    (Channels 31..00=XXXXXXXX in Hex)
    -c1@XXXXXXXX    (Channels 63..32=XXXXXXXX in Hex)
 -d Delay           (Delay in milli-secs between screen refresh -- default is 0)
 -D DMAEngineNo     (DMA Engine number -- default = 1)
 -F DebugFile       (Debug file with menu display -- default "=== None ===")
    @DebugFile      (Debug file without display)
    @               (No debug file and no display)
 -l LoopCnt         (Loop Count - default = 0)
 -m XferMode        (Transfer Mode -- default = DMA)
    -md             (Library ccurPMFC_Transfer_Data():   DMA mode)
    -mp             (Library ccurPMFC_Transfer_Data():   Progammed I/O mode)
 -n Element2Display (Number of Elements to Display -- default is 128)
 -T TestNumber      (Execute Self test 0=alternate or 1=increment pattern)
    -T0             (Atlernate pattern test -- AAAAAAAA/55555555)
    -T1             (Increment pattern test)
 -V VerifyPattern   (Verify capture rate, increment or alternate test pattern)
    -Vr             (compute capture rate)
    -Vp             (verify increment or alternate test pattern (-winc or -walt))
 -w Control         (select control: Normal/Test(increment/alternate pattern,
                     div=1,2,4,8,16,32,64,128), Start/Stop
    -wte -wsta      (select 'test; mode and 'start' capture)
    -wn  -wStop     (select 'normal' mode and 'stop' capture)
    -walt -wtest    (select 'alternation - 5555/AAAA' and 'test' mode capture)
    -winc -wtest    (select 'increment - 5555/AAAA' and 'test' mode capture)
```

```
    -winc -w128 -wtest
                    (select 'increment - 5555/AAAA' and 'test' mode capture with
                     time divider of 128)
 -x                 (Skip decoding of timestamp)

 e.g. ./ccurpmfc_ipcore_cos -c0@ffff -c1@ffff0000 -wtest -wstart (select channels
                             00..15 and 63..48, test mode, start capture)
      ./ccurpmfc_ipcore_cos -c0@ffffffff -c1@ffffffff -wtest -walt -wstart -Vp
                             (verify alternate 5555/AAAA pattern)
      ./ccurpmfc_ipcore_cos -T0 (verify alternate 5555/AAAA pattern)
      ./ccurpmfc_ipcore_cos -T1 (verify increment pattern test)
```

Example display:

./ccurpmfc_ipcore_cos –b1 –T1

```
                Board Number [-b]: 1
                       Delay [-d]: 20 milli-seconds
                  DMA Engine [-D]: 1
                 Loop Count  [-l]: ***Forever***
               Transfer Mode [-m]: (-d) Library ccurPMFC_Transfer_Data(): DMA I/O
Number of Elements to Display [-n]: 24
              Verify Pattern [-V]: Incrementing
                COS Control: 0x0000070f (Enable, Test Mode[Increment, Div=128], Start
                                         Capture)
                 COS Status: 0x9a500004 (Enabled, No Overflow, FIFO not Full)
           COS Channel Mask: 0xffffffff 0xffffffff (63..32 31..0)
             COS FIFO Count: 0x0000007c (124)
Number of Overflows Occurred: 0
                  ScanCount: 1218
 Number of Elements Returned:  31,   Total COS Occurred: 37706

   Read Duration (microsecs):    34.674 (min=  17.104/max=  42.532/ave=  34.769)
 Element TimeStamp (Raw)     ChannelMask         TimeStamp (Decoded)
         63....32 31....00   63....32 31....00   Day:HH:MM:SS:Mil.MicroSec
    1    00000000 49958001 - 0000932c 0000932c - 000:00:00:24:690.688020
    2    00000000 49960001 - 0000932d 0000932d - 000:00:00:24:691.343380
    3    00000000 49968001 - 0000932e 0000932e - 000:00:00:24:691.998740
    4    00000000 49970001 - 0000932f 0000932f - 000:00:00:24:692.654100
    5    00000000 49978001 - 00009330 00009330 - 000:00:00:24:693.309460
    6    00000000 49980001 - 00009331 00009331 - 000:00:00:24:693.964820
    7    00000000 49988001 - 00009332 00009332 - 000:00:00:24:694.620180
    8    00000000 49990001 - 00009333 00009333 - 000:00:00:24:695.275540
    9    00000000 49998001 - 00009334 00009334 - 000:00:00:24:695.930900
   10    00000000 499a0001 - 00009335 00009335 - 000:00:00:24:696.586260
   11    00000000 499a8001 - 00009336 00009336 - 000:00:00:24:697.241620
   12    00000000 499b0001 - 00009337 00009337 - 000:00:00:24:697.896980
   13    00000000 499b8001 - 00009338 00009338 - 000:00:00:24:698.552340
   14    00000000 499c0001 - 00009339 00009339 - 000:00:00:24:699.207700
   15    00000000 499c8001 - 0000933a 0000933a - 000:00:00:24:699.863060
   16    00000000 499d0001 - 0000933b 0000933b - 000:00:00:24:700.518420
   17    00000000 499d8001 - 0000933c 0000933c - 000:00:00:24:701.173780
   18    00000000 499e0001 - 0000933d 0000933d - 000:00:00:24:701.829140
   19    00000000 499e8001 - 0000933e 0000933e - 000:00:00:24:702.484500
   20    00000000 499f0001 - 0000933f 0000933f - 000:00:00:24:703.139860
   21    00000000 499f8001 - 00009340 00009340 - 000:00:00:24:703.795220
   22    00000000 49a00001 - 00009341 00009341 - 000:00:00:24:704.450580
   23    00000000 49a08001 - 00009342 00009342 - 000:00:00:24:705.105940
   24    00000000 49a10001 - 00009343 00009343 - 000:00:00:24:705.761300
```

3.2.26 lib/Sprom/ccurpmfc_sprom

This is a simple program to demonstrate sprom access.

```
Usage: ./ccurpmfc_sprom [-b board] [-C] [-D] [-S serialNo]
 -b <board>            (Board #, default = 0)
```

```
 -C                    (Clear ENTIRE serial PROM first)
 -D                    (Dump entire serial prom)
 -S <serialNo>         (Program board serial number)

   e.g.  ./ccurpmfc_sprom -C                 -> Clear Entire Serial Prom First
         ./ccurpmfc_sprom -D                 -> Dump Entire Serial Prom
         ./ccurpmfc_sprom -S 12345678        -> Write Serial Number
```

Example display:

./Sprom/ccurpmfc_sprom

```
Device Name:          /dev/ccurpmfc0
Board Serial Number:     12345 (0x00003039)
Serial PROM Revision:        0 (0x0000)
```

This page intentionally left blank