

# Software Interface

## CCURPMFC (WC-CP-FIO)

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

<i>Driver</i>	ccurpmfc (WC-CP-FIO)	v 22.7_2
<i>OS</i>	RedHawk	5.4, 6.0, 6.3, 6.5, 7.0, 7.2 and 7.3
<i>Vendor</i>	Concurrent Real-Time	
<i>Hardware</i>	PCIe Programmable Multi-Function Card (CP-FPGA-Ax)	
<i>Date</i>	January 8 <sup>th</sup> , 2018	



*This page intentionally left blank*

# Table of Contents

<b>1. INTRODUCTION</b> .....	<b>8</b>
1.1 Related Documents .....	8
<b>2. SOFTWARE SUPPORT</b> .....	<b>8</b>
2.1 Direct Driver Access.....	8
2.1.1 open(2) system call .....	8
2.1.2 ioctl(2) system call.....	9
2.1.3 mmap(2) system call.....	11
2.1.4 read(2) system call .....	12
2.1.5 write(2) system call.....	12
2.2 Application Program Interface (API) Access .....	13
2.2.1 ccurPMFC_Abort_DMA().....	17
2.2.2 ccurPMFC_ADC_Activate() .....	17
2.2.3 ccurPMFC_ADC_Get_CSR().....	17
2.2.4 ccurPMFC_ADC_Get_Driver_Read_Mode() .....	18
2.2.5 ccurPMFC_ADC_Get_Fifo_Channel_Select().....	18
2.2.6 ccurPMFC_ADC_Get_Fifo_Info() .....	19
2.2.7 ccurPMFC_ADC_Get_Fifo_Threshold() .....	20
2.2.8 ccurPMFC_ADC_Get_Negative_Cal() .....	20
2.2.9 ccurPMFC_ADC_Get_Offset_Cal().....	21
2.2.10 ccurPMFC_ADC_Get_Positive_Cal().....	21
2.2.11 ccurPMFC_ADC_Perform_Auto_Calibration() .....	22
2.2.12 ccurPMFC_ADC_Perform_External_Negative_Calibration() .....	22
2.2.13 ccurPMFC_ADC_Perform_External_Offset_Calibration().....	22
2.2.14 ccurPMFC_ADC_Perform_External_Positive_Calibration().....	23
2.2.15 ccurPMFC_ADC_Perform_Negative_Calibration().....	23
2.2.16 ccurPMFC_ADC_Perform_Offset_Calibration() .....	24
2.2.17 ccurPMFC_ADC_Perform_Positive_Calibration() .....	24
2.2.18 ccurPMFC_ADC_Read_Channels ().....	25
2.2.19 ccurPMFC_ADC_Read_Channels_Calibration() .....	26
2.2.20 ccurPMFC_ADC_Reset_Fifo() .....	26
2.2.21 ccurPMFC_ADC_Set_CSR() .....	26
2.2.22 ccurPMFC_ADC_Set_Driver_Read_Mode().....	27
2.2.23 ccurPMFC_ADC_Set_Fifo_Channel_Select() .....	28
2.2.24 ccurPMFC_ADC_Set_Fifo_Threshold() .....	28
2.2.25 ccurPMFC_ADC_Set_Negative_Cal() .....	29
2.2.26 ccurPMFC_ADC_Set_Offset_Cal() .....	29
2.2.27 ccurPMFC_ADC_Set_Positive_Cal() .....	30
2.2.28 ccurPMFC_ADC_Write_Channels_Calibration() .....	30
2.2.29 ccurPMFC_Add_Irq().....	31
2.2.30 ccurPMFC_BoardExpirationTimeRemaining() .....	31
2.2.31 ccurPMFC_Clear_Driver_Error().....	32
2.2.32 ccurPMFC_Clear_Lib_Error().....	32
2.2.33 ccurPMFC_Clock_Generator_Soft_Reset() .....	33
2.2.34 ccurPMFC_Clock_Get_Generator_CSR().....	33
2.2.35 ccurPMFC_Clock_Get_Generator_Info() .....	33
2.2.36 ccurPMFC_Clock_Get_Generator_Input_Clock_Enable() .....	36
2.2.37 ccurPMFC_Clock_Get_Generator_Input_Clock_Select().....	37
2.2.38 ccurPMFC_Clock_Get_Generator_Input_Clock_Status().....	37
2.2.39 ccurPMFC_Clock_Get_Generator_M_Divider() .....	38
2.2.40 ccurPMFC_Clock_Get_Generator_N_Divider() .....	38

2.2.41	ccurPMFC_Clock_Get_Generator_Output_Config()	39
2.2.42	ccurPMFC_Clock_Get_Generator_Output_Format()	40
2.2.43	ccurPMFC_Clock_Get_Generator_Output_Mode()	40
2.2.44	ccurPMFC_Clock_Get_Generator_Output_Mux()	41
2.2.45	ccurPMFC_Clock_Get_Generator_P_Divider()	42
2.2.46	ccurPMFC_Clock_Get_Generator_P_Divider_Enable()	42
2.2.47	ccurPMFC_Clock_Get_Generator_R_Divider()	43
2.2.48	ccurPMFC_Clock_Get_Generator_Revision()	43
2.2.49	ccurPMFC_Clock_Get_Generator_Value()	44
2.2.50	ccurPMFC_Clock_Get_Generator_Voltage_Select()	44
2.2.51	ccurPMFC_Clock_Get_Generator_Zero_Delay()	45
2.2.52	ccurPMFC_Clock_ReturnOutputFrequency()	45
2.2.53	ccurPMFC_Clock_Set_Generator_CSR()	45
2.2.54	ccurPMFC_Clock_Set_Generator_Input_Clock_Enable()	46
2.2.55	ccurPMFC_Clock_Set_Generator_Input_Clock_Select()	46
2.2.56	ccurPMFC_Clock_Set_Generator_M_Divider()	47
2.2.57	ccurPMFC_Clock_Set_Generator_N_Divider()	47
2.2.58	ccurPMFC_Clock_Set_Generator_Output_Config()	48
2.2.59	ccurPMFC_Clock_Set_Generator_Output_Format()	49
2.2.60	ccurPMFC_Clock_Set_Generator_Output_Mode()	49
2.2.61	ccurPMFC_Clock_Set_Generator_Output_Mux()	50
2.2.62	ccurPMFC_Clock_Set_Generator_P_Divider()	51
2.2.63	ccurPMFC_Clock_Set_Generator_P_Divider_Enable()	51
2.2.64	ccurPMFC_Clock_Set_Generator_R_Divider()	52
2.2.65	ccurPMFC_Clock_Set_Generator_Value()	52
2.2.66	ccurPMFC_Clock_Set_Generator_Voltage_Select()	53
2.2.67	ccurPMFC_Clock_Set_Generator_Zero_Delay()	53
2.2.68	ccurPMFC_Close()	54
2.2.69	ccurPMFC_Compute_All_Output_Clocks()	54
2.2.70	ccurPMFC_Convert_Physmem2avmm_Address()	55
2.2.71	ccurPMFC_Create_UserDioCosInterruptHandler()	55
2.2.72	ccurPMFC_Create_UserProcess()	57
2.2.73	ccurPMFC_DAC_Activate()	58
2.2.74	ccurPMFC_DAC_Get_CSR()	58
2.2.75	ccurPMFC_DAC_Get_Driver_Write_Mode()	59
2.2.76	ccurPMFC_DAC_Get_Fifo_Channel_Select()	60
2.2.77	ccurPMFC_DAC_Get_Fifo_Info()	60
2.2.78	ccurPMFC_DAC_Get_Fifo_Threshold()	61
2.2.79	ccurPMFC_DAC_Get_Fifo_Write_Count()	62
2.2.80	ccurPMFC_DAC_Get_Gain_Cal()	62
2.2.81	ccurPMFC_DAC_Get_Offset_Cal()	62
2.2.82	ccurPMFC_DAC_Get_Update_Source_Select()	63
2.2.83	ccurPMFC_DAC_Perform_Auto_Calibration()	63
2.2.84	ccurPMFC_DAC_Perform_Gain_Calibration()	64
2.2.85	ccurPMFC_DAC_Perform_Offset_Calibration()	64
2.2.86	ccurPMFC_DAC_Read_Channels_Calibration()	65
2.2.87	ccurPMFC_DAC_ReadBack_Channels()	65
2.2.88	ccurPMFC_DAC_Read_Channels()	66
2.2.89	ccurPMFC_DAC_Reset_Fifo()	67
2.2.90	ccurPMFC_DAC_Set_CSR()	68
2.2.91	ccurPMFC_DAC_Set_Driver_Write_Mode()	69
2.2.92	ccurPMFC_DAC_Set_Fifo_Channel_Select()	69
2.2.93	ccurPMFC_DAC_Set_Fifo_Threshold()	70
2.2.94	ccurPMFC_DAC_Set_Fifo_Write_Count()	70
2.2.95	ccurPMFC_DAC_Set_Gain_Cal()	71
2.2.96	ccurPMFC_DAC_Set_Offset_Cal()	71

2.2.97	ccurPMFC_DAC_Set_Update_Source_Select().....	72
2.2.98	ccurPMFC_DAC_Wait_For_Channel_Idle() .....	72
2.2.99	ccurPMFC_DAC_Wait_For_Fifo_To_Drain().....	73
2.2.100	ccurPMFC_DAC_Write_Channels() .....	73
2.2.101	ccurPMFC_DataToVolts().....	74
2.2.102	ccurPMFC_Destroy_AllUserProcess().....	75
2.2.103	ccurPMFC_Destroy_UserDioCosInterruptHandler() .....	75
2.2.104	ccurPMFC_Destroy_UserProcess() .....	75
2.2.105	ccurPMFC_DIO_Activate().....	76
2.2.106	ccurPMFC_DIO_Get_Channels_Polarity() .....	76
2.2.107	ccurPMFC_DIO_Get_COS_Channels_Edge_Sense().....	78
2.2.108	ccurPMFC_DIO_Get_COS_Channels_Enable().....	79
2.2.109	ccurPMFC_DIO_Get_COS_Channels_Mode().....	81
2.2.110	ccurPMFC_DIO_Get_COS_Channels_Overflow().....	83
2.2.111	ccurPMFC_DIO_Get_COS_Channels_Status() .....	84
2.2.112	ccurPMFC_DIO_Get_Input_Channels_Filter().....	86
2.2.113	ccurPMFC_DIO_Get_Input_Snapshot() .....	87
2.2.114	ccurPMFC_DIO_Get_Mode() .....	88
2.2.115	ccurPMFC_DIO_Get_Output_Sync() .....	88
2.2.116	ccurPMFC_DIO_Get_Ports_Direction() .....	88
2.2.117	ccurPMFC_DIO_Read_Custom_Channel_Registers().....	89
2.2.118	ccurPMFC_DIO_Read_Input_Channel_Registers().....	91
2.2.119	ccurPMFC_DIO_Read_Output_Channel_Registers() .....	93
2.2.120	ccurPMFC_DIO_Set_Channels_Polarity().....	95
2.2.121	ccurPMFC_DIO_Set_COS_Channels_Edge_Sense() .....	96
2.2.122	ccurPMFC_DIO_Set_COS_Channels_Enable().....	98
2.2.123	ccurPMFC_DIO_Set_COS_Channels_Mode() .....	99
2.2.124	ccurPMFC_DIO_Set_Input_Channels_Filter() .....	101
2.2.125	ccurPMFC_DIO_Set_Input_Snapshot() .....	103
2.2.126	ccurPMFC_DIO_Set_Mode().....	103
2.2.127	ccurPMFC_DIO_Set_Output_Sync().....	104
2.2.128	ccurPMFC_DIO_Set_Ports_Direction().....	104
2.2.129	ccurPMFC_DIO_Set_Ports_Direction_To_Input() .....	105
2.2.130	ccurPMFC_DIO_Set_Ports_Direction_To_Output().....	106
2.2.131	ccurPMFC_DIO_Write_Output_Channel_Registers() .....	107
2.2.132	ccurPMFC_DIO_Write_Output_Channel_High_Registers() .....	109
2.2.133	ccurPMFC_DIO_Write_Output_Channel_Low_Registers().....	110
2.2.134	ccurPMFC_Disable_Pci_Interrupts().....	112
2.2.135	ccurPMFC_DMA_Configure().....	112
2.2.136	ccurPMFC_DMA_Fire().....	113
2.2.137	ccurPMFC_Enable_Pci_Interrupts().....	113
2.2.138	ccurPMFC_Fast_Memcpy() .....	114
2.2.139	ccurPMFC_Fast_Memcpy_Unlocked().....	114
2.2.140	ccurPMFC_Fast_Memcpy_Unlocked_FIFO() .....	114
2.2.141	ccurPMFC_Fraction_To_Hex().....	115
2.2.142	ccurPMFC_Get_All_Boards_Driver_Info() .....	115
2.2.143	ccurPMFC_Get_Board_CSR() .....	117
2.2.144	ccurPMFC_Get_Board_Info() .....	117
2.2.145	ccurPMFC_Get_Calibration_CSR() .....	118
2.2.146	ccurPMFC_Get_Driver_Error().....	118
2.2.147	ccurPMFC_Get_Driver_Info() .....	119
2.2.148	ccurPMFC_Get_Interrupt_Status().....	124
2.2.149	ccurPMFC_Get_Interrupt_Timeout_Seconds().....	125
2.2.150	ccurPMFC_Get_Lib_Error().....	125
2.2.151	ccurPMFC_Get_Library_Info().....	127
2.2.152	ccurPMFC_Get_Mapped_Config_Ptr().....	128

2.2.153	ccurPMFC_Get_Mapped_Driver_Library_Ptr()	129
2.2.154	ccurPMFC_Get_Mapped_Local_Ptr()	129
2.2.155	ccurPMFC_Get_Open_File_Descriptor()	130
2.2.156	ccurPMFC_Get_Physical_Memory()	130
2.2.157	ccurPMFC_Get_RunCount_UserProcess()	131
2.2.158	ccurPMFC_Get_TestBus_Control()	131
2.2.159	ccurPMFC_Get_Value()	131
2.2.160	ccurPMFC_Hex_To_Fraction()	132
2.2.161	ccurPMFC_Initialize_Board()	132
2.2.162	ccurPMFC_IpCore_COS_Activate()	132
2.2.163	ccurPMFC_IpCore_COS_Configure()	133
2.2.164	ccurPMFC_IpCore_COS_Decode_Timestamp()	134
2.2.165	ccurPMFC_IpCore_COS_Get_Info()	134
2.2.166	ccurPMFC_IpCore_COS_Read()	135
2.2.167	ccurPMFC_IpCore_COS_Start_Stop()	136
2.2.168	ccurPMFC_IpCore_Get_Info()	137
2.2.169	ccurPMFC_IpCore_Get_Mapped_Ptr()	137
2.2.170	ccurPMFC_MMap_Physical_Memory()	138
2.2.171	ccurPMFC_Munmap_Physical_Memory()	139
2.2.172	ccurPMFC_NanoDelay()	139
2.2.173	ccurPMFC_Open()	139
2.2.174	ccurPMFC_Pause_UserProcess()	140
2.2.175	ccurPMFC_Program_All_Output_Clocks()	140
2.2.176	ccurPMFC_Read()	142
2.2.177	ccurPMFC_Reload_Firmware()	142
2.2.178	ccurPMFC_Remove_Irq()	143
2.2.179	ccurPMFC_Reset_Board()	143
2.2.180	ccurPMFC_Reset_Clock()	143
2.2.181	ccurPMFC_Resume_UserProcess()	144
2.2.182	ccurPMFC_Return_Board_Info_Description()	144
2.2.183	ccurPMFC_SDRAM_Activate()	144
2.2.184	ccurPMFC_SDRAM_Get_CSR()	145
2.2.185	ccurPMFC_SDRAM_Read()	145
2.2.186	ccurPMFC_SDRAM_Set_CSR()	146
2.2.187	ccurPMFC_SDRAM_Write()	146
2.2.188	ccurPMFC_Set_Board_CSR()	147
2.2.189	ccurPMFC_Set_Calibration_CSR()	147
2.2.190	ccurPMFC_Set_Interrupt_Status()	148
2.2.191	ccurPMFC_Set_Interrupt_Timeout_Seconds()	149
2.2.192	ccurPMFC_Set_TestBus_Control()	149
2.2.193	ccurPMFC_Set_Value()	149
2.2.194	ccurPMFC_SPROM_Read()	150
2.2.195	ccurPMFC_SPROM_Read_Item()	150
2.2.196	ccurPMFC_SPROM_Write()	151
2.2.197	ccurPMFC_SPROM_Write_Item()	151
2.2.198	ccurPMFC_SPROM_Write_Override()	152
2.2.199	ccurPMFC_Transfer_Data()	152
2.2.200	ccurPMFC_Update_Clock_Generator_Divider()	153
2.2.201	ccurPMFC_UserProcess_Command()	154
2.2.202	ccurPMFC_VoltsToData()	154
2.2.203	ccurPMFC_Wait_For_Interrupt()	155
2.2.204	ccurPMFC_Write()	156

### 3. TEST PROGRAMS..... 157

3.1	Direct Driver Access Example Tests	157
3.1.1	ccurpmfc_disp	157

3.1.2	ccurpmfc_dma .....	158
3.1.3	ccurpmfc_dump .....	159
3.1.4	ccurpmfc_rdreg .....	161
3.1.5	ccurpmfc_reg .....	162
3.1.6	ccurpmfc_regedit .....	169
3.1.7	ccurpmfc_tst .....	169
3.1.8	ccurpmfc_wreg .....	170
3.1.9	Flash/ccurpmfc_flash.....	171
3.1.10	Flash/ccurpmfc_label.....	172
3.1.11	Flash/ccurpmfc_dump_license .....	172
3.2	Application Program Interface (API) Access Example Tests .....	174
3.2.1	lib/ccurpmfc_adc .....	174
3.2.2	lib/ccurpmfc_adc_calibrate.....	176
3.2.3	lib/ccurpmfc_adc_fifo.....	177
3.2.4	lib/ccurpmfc_clock .....	179
3.2.5	lib/ccurpmfc_dac .....	180
3.2.6	lib/ccurpmfc_dac_calibrate.....	184
3.2.7	lib/ccurpmfc_dac_setchan .....	185
3.2.8	lib/ccurpmfc_dio.....	187
3.2.9	lib/ccurpmfc_dio_intr .....	188
3.2.10	lib/ccurpmfc_disp .....	191
3.2.11	lib/ccurpmfc_dma .....	192
3.2.12	lib/ccurpmfc_example .....	193
3.2.13	lib/ccurpmfc_expires .....	194
3.2.14	lib/ccurpmfc_identify.....	195
3.2.15	lib/ccurpmfc_info .....	195
3.2.16	lib/ccurpmfc_transfer.....	199
3.2.17	lib/ccurpmfc_tst_lib .....	200
3.2.18	lib/IpCore/ccurpmfc_ipcore_cos.....	202
3.2.19	lib/Sprom/ccurpmfc_sprom .....	204

*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). For additional information on programming, please refer to the *Concurrent Real-Time PCIe Programmable Multi-Function I/O Cards (MIOC) Design Specification (No. 0610104)* document.

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 Card I/O (MIOC) Design Specification (No. 0610104) by Concurrent Real-Time.

## 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. This driver allows multiple applications to open the same board by specifying the 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



used 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\_CCURAOCC\_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. 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_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_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_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()
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_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_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 will fail. The user can also use this call to return the current state of the ADC without any change.

```

/*****
  _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_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
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;
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)
    *****/

```

```
typedef struct {
    uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
    double  Float[CCURPMFC_MAX_ADC_CHANNELS];
} ccurpmfc_adc_cal_t;
```

## 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)
*****/

typedef struct {
    uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
    double  Float[CCURPMFC_MAX_ADC_CHANNELS];
} ccurpmfc_adc_cal_t;
```

## 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)
*****/

typedef struct {
    uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
    double  Float[CCURPMFC_MAX_ADC_CHANNELS];
} ccurpmfc_adc_cal_t;
```

### 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\_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\_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 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)
*****/

```

### 2.2.20 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)
*****/

```

### 2.2.21 ccurPMFC\_ADC\_Set\_CSR()

This call sets the DAC 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_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: 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)
*****/
```

### 2.2.22 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)
```

\*\*\*\*\*/

### 2.2.23 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)

\*\*\*\*\*/

### 2.2.24 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)
    *****/

```

## 2.2.25 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)
    *****/

```

```

typedef struct {
    uint          Raw[CCURPMFC_MAX_ADC_CHANNELS];
    double        Float[CCURPMFC_MAX_ADC_CHANNELS];
} ccurpmfc_adc_cal_t;

```

## 2.2.26 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)
*****/

typedef struct {
    uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
    double  Float[CCURPMFC_MAX_ADC_CHANNELS];
} ccurpmfc_adc_cal_t;

```

## 2.2.27 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)
*****/

typedef struct {
    uint    Raw[CCURPMFC_MAX_ADC_CHANNELS];
    double  Float[CCURPMFC_MAX_ADC_CHANNELS];
} ccurpmfc_adc_cal_t;

```

## 2.2.28 ccurPMFC\_ADC\_Write\_Channels\_Calibration()

This call allows the user to write the calibration registers from a user supplied calibration file.

```

/*****
_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)
*****/

```

### 2.2.29 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)
*****/

```

### 2.2.30 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.*

If 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 an 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.

### 2.2.31 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)
  */
```

### 2.2.32 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)
  */
```



### 2.2.33 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)
*/
```

### 2.2.34 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)
*/
```

### 2.2.35 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_voltssel_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          Pxab
    _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)
*****/

```

## 2.2.36 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)
*****/

```

### 2.2.37 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_INF8

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)
*****/

```

### 2.2.38 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 *ClkStatus (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)
*****/

```

### 2.2.39 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)
*****/

```

### 2.2.40 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)
*****/
```

## 2.2.41 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)
```

\*\*\*\*\*/

## 2.2.42 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)
```

\*\*\*\*\*/

## 2.2.43 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)
*****/

```

## 2.2.44 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)
*****/

```

## 2.2.45 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)
*****/

```

## 2.2.46 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)
*****/

```

## 2.2.47 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)
*****/

```

## 2.2.48 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
        _convert_base_part_number_t BasePartNumber;
        _convert_base_part_number_t
        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)
*****/

```

## 2.2.49 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)
*****/

```

## 2.2.50 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)
*****/

```

## 2.2.51 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)
*/
```

## 2.2.52 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
*/
```

## 2.2.53 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_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

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)
*****

```

## 2.2.54 ccurPMFC\_Clock\_Set\_Generator\_Input\_Clock\_Enable()

This call sets the input clock status for the input clocks.

```

/*****
    _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_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

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)
*****

```

## 2.2.55 ccurPMFC\_Clock\_Set\_Generator\_Input\_Clock\_Select()

This call sets the input clock selection.

```

/*****
    _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_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_INF8

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\_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.

```

/*****
_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)
*****/

```

### 2.2.57 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.

```

/*****
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)
         *****/
```

## 2.2.58 ccurPMFC\_Clock\_Set\_Generator\_Output\_Config()

This call sets the clock generator Output Configuration for the selected output.

```
/*
 *
 */
_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_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

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\_Format()

This call sets the clock generator Output Format for the selected output.

```

/*****
    _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_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

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\_Mode()

This call sets the clock generator Output Mode for the selected output.

```

/*****
    _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_outmode_common_t         common
      # CCURPMFC_CG_OUTPUT_COMMON_CMOS
      # CCURPMFC_CG_OUTPUT_COMMON_LVDS
      # CCURPMFC_CG_OUTPUT_COMMON_LVPECL
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\_Mux()

This call sets the clock generator Output Mux for the selected output.

```

/*****
_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_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
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\_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.

```

/*****
_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)
*****/

```

## 2.2.63 ccurPMFC\_Clock\_Set\_Generator\_P\_Divider\_Enable()

This call sets the state of the clock generator P-Divider.

```

/*****
_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)
*****/

```

## 2.2.64 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.

```

/*****
_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)
*****/

```

## 2.2.65 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.

```

/*****
_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)
```

\*\*\*\*\*/

### 2.2.66 ccurPMFC\_Clock\_Set\_Generator\_Voltage\_Select()

Program the clock generator voltage selection.

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

```
_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)
```

\*\*\*\*\*/

### 2.2.67 ccurPMFC\_Clock\_Set\_Generator\_Zero\_Delay()

Program the clock generator zero delay.

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

```
_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)
*****/

```

## 2.2.68 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)
*****/

```

## 2.2.69 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)
*****/

```

### 2.2.70 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)
*****/

```

### 2.2.71 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 will 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;

```

## 2.2.72 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 access 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.

```

/*****
_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;

```

### 2.2.73 ccurPMFC\_DAC\_Activate()

This call must be the first call to activate the DAC. Without activation, all other calls will fail. The user can also use this call to return the current state of the DAC without any change.

```

/*****
_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_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)
*****/

```

### 2.2.74 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)

*****/

```

## 2.2.75 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)
*****/

```

## 2.2.76 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)
*****/

```

## 2.2.77 ccurPMFC\_DAC\_Get\_Fifo\_Info()

This call returns DAC FIFO information to the user.

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

/*****
_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;
uint 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)
*****/

```

## 2.2.78 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)
*****/

```

### 2.2.79 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)
*****/

```

### 2.2.80 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)
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)
*****/

```

### 2.2.81 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)
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\_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)
*****/
```

## 2.2.83 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)
```

```

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)
*****/

```

## 2.2.84 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)

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)
*****/

```

## 2.2.85 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)

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)
*****/

```



## 2.2.86 ccurPMFC\_DAC\_Read\_Channels\_Calibration()

This routine reads the DAC channel calibration registers and dumps 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)
*****/
```

## 2.2.87 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)
*****/

```

## 2.2.88 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)
*****/

```

## 2.2.89 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
                        _ccurpmfc_dac_fifo_reset_t *Handle,
                        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)

```

\*\*\*\*\*/

## 2.2.90 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
                    _ccurpmfc_dac_mask_t *Handle,
                    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_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: 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\_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)
*****/

```

## 2.2.92 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)
*****/

```

### 2.2.93 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)
*****/

```

### 2.2.94 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)
*****/

```

## 2.2.95 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_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)

```

```

*****/

```

## 2.2.96 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_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)

```

```

*****/

```

## 2.2.97 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)
*****/
```

## 2.2.98 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)
*****/

```

## 2.2.99 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)
*****/

```

## 2.2.100 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)
*****/

```

### 2.2.101 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*.

```

/*****
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)
*****/

```

### 2.2.102 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.

```

/*****
_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)
*****/

```

### 2.2.103 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)
*****/

```

### 2.2.104 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.

```

/*****
_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)
*****/

```

### 2.2.105 ccurPMFC\_DIO\_Activate()

This call must be the first call to activate the DIO. Without activation, all other calls will fail. The user can also use this call to return the current state of the DIO without any change.

```

/*****
_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_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)
*****/

```

### 2.2.106 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
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
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)
*****/

```

### 2.2.107 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
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
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)
*****/

```

## 2.2.108 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

```

```

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

```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.



```

# 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
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)
*****/

```

## 2.2.109 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
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
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\_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  
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
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\_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
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

```

```

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\_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

```

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
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\_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, then 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)
*****/

```

### 2.2.114 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)
*****/

```

### 2.2.115 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)
*****/

```

### 2.2.116 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)
*****/

```

### 2.2.117 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 of 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)
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_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
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\_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\_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 selects the *snapshot* operation, 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* option 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 DIO being enabled prior to proceeding. If this option is set to *CCURPMFC\_TRUE*, then no test is. If the DIO has not been enabled, input reads will be invalid. 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.

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_DO_NOT_CHANGE
        ccurpmfc_bool skip_dio_disabled_check (skip dio disabled check)
        # CCURPMFC_TRUE
        # CCURPMFC_FALSE

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_channels_t ChannelSelectMask (input
channel selection)
        # NULL (select
all channels)
        # 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
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\_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)
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_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
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\_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_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
```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

# 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

```

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)

```

\*\*\*\*\*/

## 2.2.121 `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)

```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.



```

# 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_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

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)
*****/

```

## 2.2.122 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_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

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)
*****/

```

### 2.2.123 ccurPMFC\_DIO\_Set\_COS\_Channels\_Mode()

This call sets the change-of-state mode registers 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_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

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\_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 '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_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_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

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\_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()* an option to set it call.

If the user wants to collect data in the continuous mode, then 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 *CCURPMF\_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\_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\_SYNC* option for *dio\_snapshot*. There is really no need to issue this 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)
*****/
```

## 2.2.126 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)
*****/

```

## 2.2.127 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\_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)
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)
*****/

```

## 2.2.128 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)
*****/
```

### 2.2.129 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)
*****/

```

### 2.2.130 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 grouping 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)
*****/

```

### 2.2.131 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\_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 test is. 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_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_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

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\_Write\_Output\_Channel\_High\_Registers()

This call writes a selected set of channels to high outputs. Rest of the channels is 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 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\_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 test is. 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_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_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

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\_Write\_Output\_Channel\_Low\_Registers()

This call writes a selected set of channels to low outputs. Rest of the channels is 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_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 test is. 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_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_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

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\_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, uint interrupt_mask)

Description: Disable interrupts being generated by the board.

Input:   void           *Handle           (Handle pointer)
         uint           interrupt_mask    (interrupt 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_IOCTL_FAILED      (driver ioctl call
                                         failed)
*****/

```

### 2.2.135 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)
*****/

```

### 2.2.136 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)
*****/

```

### 2.2.137 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, uint interrupt_mask)

Description: Enable interrupts being generated by the board.

Input:  void          *Handle          (Handle pointer)

```

```

uint          interrupt_mask      (interrupt 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_IOCTL_FAILED  (driver ioctl call failed)
*****/

```

### 2.2.138 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.

```

/*****
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)
Output:  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)
*****/

```

### 2.2.139 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.

```

/*****
void
ccurPMFC_Fast_Memcpy_Unlocked(volatile void *Destination,
                              volatile void *Source,
                              int           SizeInBytes)

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)
Output:  volatile void *Destination    (pointer to destination buffer)
Return:  none
*****/

```

### 2.2.140 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.

```

/*****
void
ccurPMFC_Fast_Memcpy_Unlocked_FIFO(volatile void *Destination,
                                   volatile void *Source,
                                   int           SizeInWords,

```

```
int PioControl)
```

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)
Output: volatile void *Destination    (pointer to destination buffer)
Return: none
```

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

### 2.2.141 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)
```

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

### 2.2.142 ccurPMFC\_Get\_All\_Boards\_Driver\_Info()

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

```
ccurPMFC_Get_All_Boards_Driver_Info()
_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 board_id
        int board_type
        int sub_device_id
        u_int BoardInfo
        u_int FirmwareDate
```

```

u_int          FirmwareRevision
int            msi_support
int            irqlevel
int            firmware
double        calibration_reference_voltage
int            driver_dma_size
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
ccurpmfc_driver_int_t interrupt
uint          InterruptsOccurredMask
uint          WakeupInterruptMask
int           timeout_seconds
int           DmaControl
ccurpmfc_dio_channels_t DIO_COS_ChannelsStatus
ccurpmfc_dio_channel_t DIO_COS_ChannelsStatusX
ccurpmfc_dio_channels_t DIO_COS_ChannelsOverflow
ccurpmfc_dio_channel_t DIO_COS_ChannelsOverflowX
ccurpmfc_interrupt_counters_t counters
    unsigned long long count
    unsigned long long dma_count[CCURPMFC_DMA_MAX_ENGINES]
    long long unsigned
        DIO_COS_ChannelsCount[CCURPMFC_DIO_MAX_REGISTERS]
int          Ccurpmfc_Max_Region
ccurpmfc_dev_region_t mem_region[CCURPMFC_MAX_REGION]
    uint      physical_address
    uint      size
    uint      flags
    uint      *virtual_address
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
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
ccurpmfc_driver_sdram_info_t sdram_info
int           sdram_max_size_in_words
    _ccurpmfc_clock_generator_output_t
        sdram_output_clock
        sdram_output_clock_frequency
double
ccurpmfc_driver_clock_info_t clock_info
    _ccurpmfc_cg_input_clock_select_register_t
        default_input_clock
        default_input_clock_frequency
double
double        default_clock_tolerance_ppt

```

```

        ccurpmfc_sprom_header_t sprom_header
            u_int32_t                board_serial_number
            u_short                   sprom_revision
        char                          fpga_chip_temperature
        u_int                          FirmwareTime
        u_int                          FirmwareFlavorCode
        u_int                          NumberAdvancedIPCores
        u_short                        RunLevelSectorNumber
        char                          FirmwareReloadFailed
        char                          MultiFirmwareSupport
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)
*****/

```

## 2.2.143 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)
*****/

```

## 2.2.144 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)
*****/

```

## 2.2.145 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)
*****/

```

## 2.2.146 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)

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,
};

```

## 2.2.147 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	
	int	board_id	
	int	board_type	
	int	sub_device_id	
	u_int	BoardInfo	
	{		
	u_char	Function	
	u_char	Type	
	u_short	Id	
	}		
	u_int	FirmwareDate	
	{		
	u_short	Year	
	u_char	Day	
	u_char	Month	
	}		
	u_int	FirmwareRevision	
	{		
	u_short	Minor	
	u_short	Major	
	}		
	int	msi_support	
	int	irqlevel	
	int	firmware	
	double	calibration_reference_voltage	
	int	driver_dma_size	
	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	
	ccurpmfc_driver_int_t	interrupt	
	uint	InterruptsOccurredMask	
	uint	WakeupInterruptMask	
	int	timeout_seconds	
	int	DmaControl	
	ccurpmfc_dio_channels_t	DIO_COS_ChannelsStatus	
	ccurpmfc_dio_channel_t	DIO_COS_ChannelsStatusX	
	ccurpmfc_dio_channels_t	DIO_COS_ChannelsOverflow	
	ccurpmfc_dio_channel_t	DIO_COS_ChannelsOverflowX	
	ccurpmfc_interrupt_counters_t	counters	
	unsigned long long	count	
	unsigned long long	dma_count[CCURPMFC_DMA_MAX_ENGINES]	



```

        long long unsigned
        DIO_COS_ChannelsCount[CCURPMFC_DIO_MAX_REGISTERS]
int      Ccurpmfc_Max_Region
ccurpmfc_dev_region_t mem_region[CCURPMFC_MAX_REGION]
    uint   physical_address
    uint   size
    uint   flags
    uint   *virtual_address
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
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
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
ccurpmfc_driver_sdram_info_t sdram_info
int      sdram_max_size_in_words
    _ccurpmfc_clock_generator_output_t
        double   sdram_output_clock
        double   sdram_output_clock_frequency
ccurpmfc_driver_clock_info_t clock_info
    _ccurpmfc_cg_input_clock_select_register_t
        double   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
char    fpga_chip_temperature
                                degree C in two's complement
u_int   FirmwareTime
{
    u_char   Hour
    u_char   Minute
    u_char   Second
}
u_int   FirmwareFlavorCode
u_int   NumberAdvancedIPCoers
u_short RunLevelSectorNumber
char    FirmwareReloadFailed
char    MultiFirmwareSupport
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_MAX_REGION 32

// DMA
typedef struct {
    short    num_trans_tbl_entries;    /* number of translation table pages */
    int      avalon_page_bits;        /* number of avalon page size in bits */
    int      avalon_page_size;        /* size of each avalon page */
    int      tx_interface_base;        /* TX base address */
    int      dma_max_engines;          /* maximum dma engines */
    int      dma_max_burst_size;        /* maximum burst size in bytes */
    int      dma_max_transactions;     /* maximum DMA transactions at a time */
    int      dma_max_size;             /* maximum DMA size */
    int      dma_width_in_bytes;       /* DMA transfer width in bytes */
    int      dma_fire_command;         /* DMA fire command */
} ccurpmfc_driver_dma_info_t;

// Interrupt
typedef struct
{
    long long unsigned    count;
    long long unsigned    dma_count[CCURPMFC_DMA_MAX_ENGINES];
    long long unsigned    DIO_COS_ChannelsCount[CCURPMFC_DIO_MAX_REGISTERS];
} ccurpmfc_interrupt_counters_t;

typedef struct
{
    union {
        uint                status;                // Obsolete
        uint                InterruptsOccurredMask; // use this name
    };
    union {
        uint                mask;                // Obsolete
        uint                WakeupInterruptMask; // use this name
    };
    int                    timeout_seconds;
    int                    DmaControl;

    // 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_counters_t    counters;
} ccurpmfc_driver_int_t;

// Memory Region
typedef struct
{
    uint physical_address;
    uint size;
    uint flags;
    uint *virtual_address;
} ccurpmfc_dev_region_t;

// ADC
typedef struct {
    double adc_max_voltage_range;    /* adc max voltage range */

```

```

    int    number_of_adcs;           /* number of ADCs in this board */
    int    number_of_adc_channels;   /* number of ADC channels in this board*/
    int    number_of_adc_resolutionbits; /* number of ADC resolution bits */
    int    all_adc_channels_mask;    /* all ADC channels mask */
    int    max_adc_fifo_threshold;   /* maximum ADC Fifo Threshold */
} ccurpmfc_driver_adc_info_t;

// DAC
typedef struct {
    double dac_max_voltage_range;    /* dac max voltage range */
    int    number_of_dacs;           /* number of DACs in this board */
    int    number_of_dac_channels;   /* number of DAC channels in this board*/
    int    number_of_dac_resolutionbits; /* number of DAC resolution bits */
    int    all_dac_channels_mask;    /* all DAC channels mask */
} ccurpmfc_driver_dac_info_t;

// SDRAM
typedef struct {
    int    sdr_max_size_in_words;    /* maximum SDRAM size in words */
    _ccurpmfc_clock_generator_output_t sdr_output_clock;
                                        /* SDRAM output clock */
    double sdr_output_clock_frequency; /* SDRAM output clock frequency */
} ccurpmfc_driver_sdr_info_t;

// CLOCK
typedef struct {
    _ccurpmfc_cg_input_clock_select_register_t default_input_clock;
                                        /* Default Input Clock */
    double default_input_clock_frequency; /* Default Input Clock Frequency */
    double default_clock_tolerance_ppt;  /* Default Clock Tolerance in
                                        Parts/Trillion */
} ccurpmfc_driver_clock_info_t;

// Information
typedef struct {
    char version[12];                 /* driver version */
    char built[32];                  /* driver date built */
    char module_name[16];            /* driver name */
    int board_index;                 /* board index */
    int table_index;                 /* table index */
    char board_desc[32];             /* board description */
    int bus;                          /* bus number */
    int slot;                         /* slot number */
    int func;                         /* function number */
    int vendor_id;                   /* vendor id */
    int sub_vendor_id;               /* sub-vendor id */
    int sub_device_id;               /* sub device id */
    union {
        u_int BoardInfo;             /* BoardInfo if applicable */
        ccurpmfc_boardinfo_t BInfo;
    };
    union {
        u_int FirmwareDate;          /* Firmware Date if applicable */
        ccurpmfc_firmware_date_t FmDate;
    };
    union {
        u_int FirmwareRevision;      /* Firmware Revision if applicable */
        ccurpmfc_firmware_revision_t FmRev;
    };
    int msi_support;                 /* msi flag 1=MSI support, 0=NO MSI */
    int irqlevel;                   /* IRQ level */
    double calibration_reference_voltage; /* calibration bus reference voltage */
}

```

```

int driver_dma_size;                /* driver DMA size in bytes */

// DMA
ccurpmfc_driver_dma_info_t dma_info; /* DMA info */

// Interrupt
ccurpmfc_driver_int_t interrupt;     /* interrupt information */

int Ccurpmfc_Max_Region;             /*kernel DEVICE_COUNT_RESOURCE */

// Memory Region
ccurpmfc_dev_region_t mem_region[CCURPMFC_MAX_REGION];
                                     /* memory region */

// ADC
ccurpmfc_driver_adc_info_t adc_info; /* ADC info */

// DAC
ccurpmfc_driver_dac_info_t dac_info; /* DAC info */

// DIO
ccurpmfc_driver_dio_info_t dio_info; /* DIO info */

// SDRAM
ccurpmfc_driver_sdram_info_t sdram_info;
                                     /* SDRAM info */

// CLOCK
ccurpmfc_driver_clock_info_t clock_info;
                                     /* Clock info */
ccurpmfc_sprom_header_t      sprom_header;

// Chip Temperature
char    fpga_chip_temperature;      /* degree C in two's complement */
char    RESERVED_CHAR[3];
union {
    u_int FirmwareTime;              /* Firmware Time if applicable */
    ccurpmfc_firmware_time_t  FmTime;
};
u_int    FirmwareFlavorCode;         /* Firmware Flavor Code */
u_int    NumberAdvancedIPCores;      /* Number of Advanced IP Cores */

u_short RunLevelSectorNumber;       /* Run level sector number */
char     FirmwareReloadFailed;
char     MultiFirmwareSupport;

uint     RESERVED_INT[4093];
} ccurpmfc_driver_info_t;

```

## 2.2.148 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)
*****/

```

### 2.2.149 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)
*****/

```

### 2.2.150 ccurPMFC\_Get\_Lib\_Error()

This call provides detailed information about the last library error that was maintained by the API.

```

/*****
    _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           (error number)
        char name[CCURPMFC_LIB_ERROR_NAME_SIZE] (error name)
        char desc[CCURPMFC_LIB_ERROR_DESC_SIZE] (error description)
        int  line_number      (error line number in lib)
        char function[CCURPMFC_LIB_ERROR_FUNC_SIZE]

```

```

error)
Return:  _ccurpmfc_lib_error_number_t
        # CCURPMFC_LIB_BAD_HANDLE           (no/bad handler supplied)
        # CCURPMFC_LIB_NOT_OPEN            (device not open)
        # Last Library Error                (last library error)
*****

typedef struct _ccurpmfc_lib_error_t {
    uint    error;                          /* lib error number */
    char    name[CCURPMFC_LIB_ERROR_NAME_SIZE]; /* error name used in lib */
    char    desc[CCURPMFC_LIB_ERROR_DESC_SIZE]; /* error description */
    int     line_number;                     /* line number in library */
    char    function[CCURPMFC_LIB_ERROR_FUNC_SIZE];
                                              /* library function */
} ccurpmfc_lib_error_t;

```

**Possible library errors:**

```

typedef enum {
    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, /* Already mapped */
    CCURPMFC_LIB_IOCTL_FAILED            = -11, /* Driver 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 failed to initialize */
CCURPMFC_LIB_DAC_FIFO_UNDERFLOW        = -40 /* DAC Fifo Underflow Occurred */
CCURPMFC_LIB_DAC_FIFO_OVERFLOW         = -41 /* DAC Fifo Overflow Occurred */
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 /* Authorization failure */
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_error_number_t;

```

## 2.2.151 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;

/* IpCore Pointers */
        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_Receiiver_ptr;
        void                *IpCore_SENT_Transmitter_ptr;
        void                *IpCore_Angular_Encoder_ptr;
        void                *IpCore_Angular_Decoder_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)
*****/

typedef struct
{
    int                fp; /* file descriptor number */

    /* local memory */
    ccurpmfc_local_ctrl_data_t    *local_ptr;
    void                *munmap_local_ptr;
    int                local_mmap_size;

    /* config memory */
    ccurpmfc_config_local_data_t  *config_ptr;
    void                *munmap_config_ptr;
    int                config_mmap_size;

    /* physical memory */
    ccurpmfc_user_phys_mem_t    PhysMem[CCURPMFC_MAX_AVALON_NUM_TRANS_TBL_ENTRIES];

    ccurpmfc_driver_library_common_t  *driver_lib_ptr;
    void                *munmap_driver_lib_ptr;
    int                driver_lib_mmap_size;
} ccurpmfc_library_info_t;

```

## 2.2.152 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)
*****/

```



## 2.2.153 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_Get_Mapped_Driver_Library_Ptr()
_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)

Return:  -- structure in ccurpmfc_user.h
         _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)

*****/

typedef struct
{
    uint dma_abort_count;          /* DMA abort count */
    uint library_needs_initialization;
    uint library_lock[8];
    uint open_count;
    uint ADC_FifoDriverThreshold;
    uint DAC_FifoDriverThreshold;
    ccurpmfc_sprom_header_t sprom_header;
    uint RESERVED[1024];
} ccurpmfc_driver_library_common_t;

```

## 2.2.154 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)
*****/

```

## 2.2.155 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)
*****/

```

## 2.2.156 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. 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 mmaped() 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)

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)
*****/

```

```

typedef struct
{
    uint        user_pid;           /* user PID */
    void        *phys_mem_ptr;      /* physical memory address */
    void        *driver_virt_mem_ptr; /* physical mem: driver virtual
                                     address */

    void        *mmaped_user_mem_ptr; /* mmaped user virtual memory */
    uint        phys_mem_size;       /* physical mem: memory size - bytes*/
    uint        phys_mem_size_freed; /* physical memory: freed size */
    uint        entry_num_in_tran_table; /* Physical Address entry number in
                                     translation table */
} ccurpmfc_user_phys_mem_t;

```

### 2.2.157 ccurPMFC\_Get\_RunCount\_UserProcess()

This call returns to the user a count of the number of times the User Process has entered.

```

/*****
_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)
*****/

```

### 2.2.158 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)
*****/

```

### 2.2.159 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)
*****/

```

### 2.2.160 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)
*****/

```

### 2.2.161 ccurPMFC\_Initialize\_Board()

This call resets the board to a default initial state. This call is currently identical to the *ccurPMFC\_Reset\_Board()* call.

```

/*****
_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)
*****/

```

### 2.2.162 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_status_disabled_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_status_disabled_t activate (activate/deactivate)
        # CCURPMFC_IPCORE_COS_STATUS_IP_ENABLED
        # CCURPMFC_IPCORE_COS_STATUS_IP_DISABLED
        # CCURPMFC_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
        # CCURPMFC_DO_NOT_CHANGE

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)
*****/

```

### 2.2.163 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 decide 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)
*****/

```

## 2.2.164 ccurPMFC\_IpCore\_COS\_Decode\_Timestamp()

This is a useful call that is available to the user to decode the raw timestamp of the change-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
*****/

```

## 2.2.165 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)
*****/

```

## 2.2.166 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 in 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)
*****/

```

## 2.2.167 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)
*****/

```



## 2.2.168 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_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
        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)
*****/
```

## 2.2.169 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_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

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)
*****/

```

## 2.2.170 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. 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 the previously allocated physical memory.

```

/*****
_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
        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_MMAP_SELECT_FAILED (mmap selection failed)
        # CCURPMFC_LIB_MMAP_FAILED     (mmap failed)
        # CCURPMFC_LIB_NO_SPACE_IN_TABLE (no space in phys memory table)
*****/

```

```

typedef struct
{
    uint        user_pid;                /* user PID */
    void        *phys_mem_ptr;           /* physical memory address */
    void        *driver_virt_mem_ptr;    /* physical mem: driver virtual
                                        address */

    void        *mmaped_user_mem_ptr;    /* mmaped user virtual memory */
    uint        phys_mem_size;           /* physical mem: memory size - bytes*/
    uint        phys_mem_size_freed;     /* physical memory: freed size */
    uint        entry_num_in_tran_table; /* Physical Address entry number in
                                        translation table */
} ccurpmfc_user_phys_mem_t;

```

### 2.2.171 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)
*****/

```

### 2.2.172 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
*****/

```

### 2.2.173 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 a 0, however the user may use the *O\_NONBLOCK* option for *read(2)* calls which will change the default reading in block mode.

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 mmaped)
         # CCURPMFC_LIB_MMAP_SELECT_FAILED (mmap selection failed)
         # CCURPMFC_LIB_MMAP_FAILED  (mmap failed)
*****/

```

### 2.2.174 ccurPMFC\_Pause\_UserProcess()

This call causes a running User Process to sleep for user specified micro-seconds.

```

/*****
_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)
*****/

```

### 2.2.175 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 value.

If the desired output clock frequencies are unable to be computed due to hardware limitation, they may wish to increase the desired tolerance for the particular clock. Note that this tolerance is only applicable to computing a clock value as close to the desired frequency 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_INFB
    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)

```

\*\*\*\*\*/

## 2.2.176 ccurPMFC\_Read()

Currently, this call is not supported. 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.

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)

```

\*\*\*\*\*/

## 2.2.177 ccurPMFC\_Reload\_Firmware()

The purpose of this call is to power cycle the board with 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)

```

\*\*\*\*\*/

## 2.2.178 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)
    *****/

```

## 2.2.179 ccurPMFC\_Reset\_Board()

This call resets the board to a known initial default state. This call is currently identical to the *ccurPMFC\_Initialize\_Board()* call.

```

/*****
    _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)
    *****/

```

## 2.2.180 ccurPMFC\_Reset\_Clock()

This call performs a hardware reset of the clock. All active output clocks are stopped and set to default state.

```

/*****
    _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)
*****/

```

### 2.2.181 ccurPMFC\_Resume\_UserProcess()

Use this call to resume an already paused User Process.

```

/*****
_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)
*****/

```

### 2.2.182 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)
Output:  none
Return:  char                        *BoardFuncDesc          (board function
                                                               description)
*****/

```

### 2.2.183 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_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)
*****/

```

## 2.2.184 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_sdr*_sram_csr_t *sdr*_sram_csr)

Description: Get SDRAM Control and Status information

Input:  void          *Handle (Handle pointer)
Output: ccurpmfc_sdr*_sram_csr_t *sdr*_sram_csr (pointer to SDRAM csr)
        _ccurpmfc_sdr*_sram_read_auto_increment_t read_auto_increment;
        # CCURPMFC_SDRAM_READ_AUTO_INCREMENT_DISABLE
        # CCURPMFC_SDRAM_READ_AUTO_INCREMENT_ENABLE
        _ccurpmfc_sdr*_sram_write_auto_increment_t write_auto_increment;
        # CCURPMFC_SDRAM_WRITE_AUTO_INCREMENT_DISABLE
        # CCURPMFC_SDRAM_WRITE_AUTO_INCREMENT_ENABLE
        _ccurpmfc_sdr*_sram_read_timeout_t read_timeout;
        # CCURPMFC_SDRAM_READ_TIMEOUT_DID_NOT_OCCUR
        # CCURPMFC_SDRAM_READ_TIMEOUT_OCCURRED
        _ccurpmfc_sdr*_sram_calibration_fail_t calibration_failed;
        # CCURPMFC_SDRAM_CALIBRATION_FAIL_RESET
        # CCURPMFC_SDRAM_CALIBRATION_FAIL_SET
        _ccurpmfc_sdr*_sram_calibration_pass_t calibration_passed;
        # CCURPMFC_SDRAM_CALIBRATION_PASS_RESET
        # CCURPMFC_SDRAM_CALIBRATION_PASS_SET
        _ccurpmfc_sdr*_sram_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)
*****/

```

## 2.2.185 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)
```

\*\*\*\*\*/

## 2.2.186 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_sdracm_csr_t *sdracm_csr)
```

Description: Set SDRAM Control and Status information

```
Input:  void                *Handle    (Handle pointer)
        _ccurpmfc_sdracm_read_auto_increment_t read_auto_increment;
        # CCURPMFC_SDRAM_READ_AUTO_INCREMENT_DISABLE
        # CCURPMFC_SDRAM_READ_AUTO_INCREMENT_ENABLE
        _ccurpmfc_sdracm_write_auto_increment_t write_auto_increment;
        # CCURPMFC_SDRAM_WRITE_AUTO_INCREMENT_DISABLE
        # CCURPMFC_SDRAM_WRITE_AUTO_INCREMENT_ENABLE
        _ccurpmfc_sdracm_read_timeout_t read_timeout;
        # CCURPMFC_SDRAM_READ_TIMEOUT_DID_NOT_OCCUR
        # CCURPMFC_SDRAM_READ_TIMEOUT_OCCURRED
        _ccurpmfc_sdracm_calibration_fail_t calibration_failed;
        # CCURPMFC_SDRAM_CALIBRATION_FAIL_RESET
        # CCURPMFC_SDRAM_CALIBRATION_FAIL_SET
        _ccurpmfc_sdracm_calibration_pass_t calibration_passed;
        # CCURPMFC_SDRAM_CALIBRATION_PASS_RESET
        # CCURPMFC_SDRAM_CALIBRATION_PASS_SET
        _ccurpmfc_sdracm_initilization_done_t initialization_done;
        # CCURPMFC_SDRAM_INITIALIZATION_NOT_COMPLETE
        # CCURPMFC_SDRAM_INITIALIZATION_COMPLETE
```

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)
```

\*\*\*\*\*/

## 2.2.187 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)
        int          Offset      (word offset into SDRAM)
        u_int32_t    Size        (size of buffer in words)
        u_int32_t    *Buf        (pointer to buffer)
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)
*****/

```

## 2.2.188 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)
*****/

```

## 2.2.189 ccurPMFC\_Set\_Calibration\_CSR()

This call sets the current calibration control and status register.

```

/*****
    ccurPMFC_Set_Calibration_CSR()

```

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)
*****/

```

## 2.2.190 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_DO_NOT_CHANGE
        _ccurpmfc_intsta_dio_cos_t dio_cos_group1_int
        # CCURPMFC_INT_DIO_COS_NONE
        # CCURPMFC_INT_DIO_COS_RESET
        # CCURPMFC_DO_NOT_CHANGE
        _ccurpmfc_intsta_dio_cos_t dio_cos_group0_int
        # CCURPMFC_INT_DIO_COS_NONE
        # CCURPMFC_INT_DIO_COS_RESET
        # CCURPMFC_DO_NOT_CHANGE
        _ccurpmfc_intsta_dac_t
        # CCURPMFC_INT_DAC_FIFO_THRESHOLD_NONE
        # CCURPMFC_INT_DAC_FIFO_THRESHOLD_RESET
        # CCURPMFC_DO_NOT_CHANGE
        _ccurpmfc_intsta_adc_t
        # CCURPMFC_INT_ADC_FIFO_THRESHOLD_NONE
        # CCURPMFC_INT_ADC_FIFO_THRESHOLD_RESET
        # CCURPMFC_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)
```

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

### 2.2.191 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)

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

### 2.2.192 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 (pointer to control select)

Return: \_ccurpmfc\_lib\_error\_number\_t  
# CCURPMFC\_TBUS\_CONTROL\_OPEN  
# CCURPMFC\_TBUS\_CONTROL\_CAL\_BUS  
# 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)

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

### 2.2.193 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)

```

\*\*\*\*\*/

## 2.2.194 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_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)

```

\*\*\*\*\*/

## 2.2.195 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_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)
*****/

```

### 2.2.196 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_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)
*****/

```

### 2.2.197 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)
            # CCUREMFC_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_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)
*****/

```

## 2.2.198 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_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)
*****/

```

## 2.2.199 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.

```

/*****
_ccurpmfc_lib_error_number_t
ccurPMFC_Transfer_Data (void
                        void      *Handle,
                        void      *PciDmaMemory,
                        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)
        void                *PciDmaMemory          (pointer to virtual memory)
        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_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_IOCTL_FAILED            (driver ioctl call failed)
*****

```

## 2.2.200 ccurPMFC\_Update\_Clock\_Generator\_Divider()

Update the selected clock generator divider so that its changes take affect.

```

/*****
_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)
*****/

```

## 2.2.201 ccurPMFC\_UserProcess\_Command()

The user can control the execution of the created User Process with the help of this call.

```

/*****
_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
*****/

```

## 2.2.202 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)
*****/

typedef struct
{
    double VoltageRange; /* Maximum Volts */
    _ccurpmfc_csr_dataformat_t Format; /* CCURPMFC_OFFSET_BINARY
                                       CCURPMFC_TWOS_COMPLEMENT */
    ccurpmfc_bool BiPolar; /* CCURPMFC_TRUE/CCURPMFC_FALSE */
    int ResolutionBits; /* Number of resolution bits */
} ccurpmfc_volt_convert_t;

```

## 2.2.203 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
        int          timeout_seconds;

Output: ccurpmfc_driver_int_t *drv_int (pointer to drv_int struct)
        uint          InterruptsOccurredMask
        int          timeout_seconds
        int          DmaControl
        # CCURPMFC_DMA_CONTROL_RCON      (read constant)
        # CCURPMFC_DMA_CONTROL_WCON      (write constant)
        # CCURPMFC_DMA_CONTROL_INCREMENT (increment)
        ccurpmfc_dio_channels_t DIO_COS_ChannelsStatus
        ccurpmfc_dio_channel_t  DIO_COS_ChannelsStatusX
        ccurpmfc_dio_channels_t DIO_COS_ChannelsOverflow
        ccurpmfc_dio_channel_t  DIO_COS_ChannelsOverflowX
        ccurpmfc_interrupt_counters_t counters
        unsigned long long      count
        unsigned long long      dma_count[CCURPMFC_DMA_MAX_ENGINES]
        long long unsigned
        DIO_COS_ChannelsCount[CCURPMFC_DIO_MAX_REGISTERS]

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)
*****/

// Interrupt Counters
typedef struct
{
    long long unsigned count;
    long long unsigned dma_count[CCURPMFC_DMA_MAX_ENGINES];
    long long unsigned DIO_COS_ChannelsCount[CCURPMFC_DIO_MAX_REGISTERS];
} ccurpmfc_interrupt_counters_t;

// Interrupt
typedef struct
{
    union {
        uint          status;          // Obsolete
        uint          InterruptsOccurredMask; // use this name
    };
    union {
        uint          mask;            // Obsolete
        uint          WakeupInterruptMask; // use this name
    };
    int          timeout_seconds;
    int          DmaControl;
}

```

```

// 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_counters_t  counters;
} ccurpmfc_driver_int_t;

```

### 2.2.204 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	05252016	00550003	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\_dma

This test transfers data from physical memory to the Local register area and back. There are three modes of operation. Two are DMA and one 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. 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 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 = Simple DMA)
  -md       (Simple DMA mode)
  -mp       (Programmed I/O mode)
  -mx       (Low-Level DMA mode)
-s Size      (Transfer Size (multiple of byte width) - default = 12288)
-V VerboseNo (Verbose -- default = 0)
```

#### Example display:

```
./ccurpmfc_dma
```

```
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
DMA: Address 0x7ffff70f4000 Offset 0x0 Size=0x00200000
                                           Physical Address=0x79800000
```

#### Physical Memory Information:

```
UserPID          =5665
PhysMemPtr       =0x79800000
DriverVirtMemPtr=0xffff880079800000
MmappedUserMemPtr=0x7ffff70f4000
PhysMemSize      =0x00200000
PhysMemSizeFreed=0x00000000
EntryInTxTbl    =0
NumOfEntriesUsed=2
```

```
### Avalon Address[A0]: 0x00001000 - 0x00004000
```

```
### DMA Address[A0]: 0x00100400 - 0x00103400
```

```
### Transfer Size: 12288 (0x00003000) (Simple DMA without interrupts: DMA Engine 0) ###
```

```
1000: A2P: Total: 566.125us ( 21.71 MB/s): first=0xface0000 last=0xface0bff
```

	(micro-seconds)			(MBytes/second)		
	Min	Max	Ave	Min	Max	Ave
P2A:	513.56	519.96	515.66	23.63	23.93	23.83
A2P:	566.01	571.24	566.77	21.51	21.71	21.68

### 3.1.3 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=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 BOARD REGISTERS =====

```
LBR: @0x0000 --> 0x92900101
```

```
LBR: @0x0004 --> 0x05252016
```

```
LBR: @0x0008 --> 0x00550003
```

```
LBR: @0x000c --> 0x00000000
```

```
LBR: @0x0010 --> 0x00000000
```

```
LBR: @0x0014 --> 0x00000000
```

```
LBR: @0x0018 --> 0x00000000
```

```
LBR: @0x001c --> 0x00000000
```

```
LBR: @0x0020 --> 0x00000000
```

```
LBR: @0x0024 --> 0x00000000
```

```
LBR: @0x0028 --> 0x00000000
```

```
LBR: @0x002c --> 0x00000000
```

```
LBR: @0x0030 --> 0x00000000
```

```
.
```

```
.
```

```
.
```

```
LBR: @0xffcc --> 0x00000000
```

```
LBR: @0xffd0 --> 0x00000000
```

```
LBR: @0xffd4 --> 0x00000000
```

```
LBR: @0xffd8 --> 0x00000000
```

```
LBR: @0xffdc --> 0x00000000
```

```
LBR: @0xffe0 --> 0x00000000
```

```
LBR: @0xffe4 --> 0x00000000
```

```
LBR: @0xffe8 --> 0x00000000
```

```
LBR: @0xffec --> 0x00000000
```

```
LBR: @0xfff0 --> 0x00000000
```

```
LBR: @0xfff4 --> 0x00000000
```

```
LBR: @0xfff8 --> 0x00000000
```

```
LBR: @0xfffc --> 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 --> 0xc4910000
PCR: @0x0014 --> 0x00000000
PCR: @0x0018 --> 0xc4900000
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

```

### 3.1.4 ccurpmfc\_rdreg

This is a simple program that returns the local register value for a given offset.

```

Usage: ./ccurpmfc_rdreg [-b Board] [-C] [-o Offset] [-s Size]
-b Board   : Board number -- default board is 0
-C         : Select Config Registers instead of Local Registers
-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  05252016  00550003  00000000  *.....% ..U.....*
+LCL+     0x10  00000000  00000000  00000000  00000000  *.....*
+LCL+     0x20  00000000  00000000  00000000  00000000  *.....*
+LCL+     0x30  00000000  00000000  00000000  00000000  *.....*
```

./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
```

### 3.1.5 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  05252016  00550003  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+ 0xffff0 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              =0x05252016 @0x00000004
FirmwareRevision          =0x00550003 @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	@x00003154
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_6]	=0x7ffa3a42	@x00003158
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_7]	=0x7fce5fe4	@x0000315c
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_8]	=0x7fc7cf4b	@x00003160
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_9]	=0x7fb18546	@x00003164
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_10]	=0x7fc57a75	@x00003168
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_11]	=0x7fb5d398	@x0000316c
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_12]	=0x7fb6057f	@x00003170
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_13]	=0x7fe2e1a0	@x00003174
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_14]	=0x7fc33037	@x00003178
ADC_NegativeCalibration[CCURPMFC_ADC_CHANNEL_15]	=0x7fd3a2d9	@x0000317c
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_0]	=0x00000002	@x00003180
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_1]	=0x00000001	@x00003184
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_2]	=0x00000000	@x00003188
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_3]	=0x0000ffff	@x0000318c
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_4]	=0x00000001	@x00003190
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_5]	=0x00000001	@x00003194
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_6]	=0x0000fffe	@x00003198
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_7]	=0x00000000	@x0000319c
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_8]	=0x00000001	@x000031a0
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_9]	=0x00000000	@x000031a4
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_10]	=0x00000000	@x000031a8
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_11]	=0x0000ffff	@x000031ac
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_12]	=0x00000000	@x000031b0
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_13]	=0x00000000	@x000031b4
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_14]	=0x0000fffe	@x000031b8
ADC_OffsetCalibration[CCURPMFC_ADC_CHANNEL_15]	=0x0000ffff	@x000031bc
ADC_Data[CCURPMFC_ADC_CHANNEL_0]	=0x00007fff	@x00003200
ADC_Data[CCURPMFC_ADC_CHANNEL_1]	=0x00007fff	@x00003204
ADC_Data[CCURPMFC_ADC_CHANNEL_2]	=0x00007fff	@x00003208
ADC_Data[CCURPMFC_ADC_CHANNEL_3]	=0x00007fff	@x0000320c
ADC_Data[CCURPMFC_ADC_CHANNEL_4]	=0x00008000	@x00003210
ADC_Data[CCURPMFC_ADC_CHANNEL_5]	=0x00008000	@x00003214
ADC_Data[CCURPMFC_ADC_CHANNEL_6]	=0x00007fff	@x00003218
ADC_Data[CCURPMFC_ADC_CHANNEL_7]	=0x00007fff	@x0000321c
ADC_Data[CCURPMFC_ADC_CHANNEL_8]	=0x00007fff	@x00003220
ADC_Data[CCURPMFC_ADC_CHANNEL_9]	=0x00008000	@x00003224
ADC_Data[CCURPMFC_ADC_CHANNEL_10]	=0x00007fff	@x00003228
ADC_Data[CCURPMFC_ADC_CHANNEL_11]	=0x00008000	@x0000322c
ADC_Data[CCURPMFC_ADC_CHANNEL_12]	=0x00008000	@x00003230
ADC_Data[CCURPMFC_ADC_CHANNEL_13]	=0x00008000	@x00003234
ADC_Data[CCURPMFC_ADC_CHANNEL_14]	=0x00008000	@x00003238
ADC_Data[CCURPMFC_ADC_CHANNEL_15]	=0x00007fff	@x0000323c
ADC_FifoData	=0xbaadbeef	@x00003300
DAC_Enable	=0x00000000	@x00004000
DAC_UpdateSourceSelect	=0x00000000	@x00004004
DAC_FifoChannelSelect	=0x00000000	@x00004008
DAC_ControlStatus[CCURPMFC_DAC_0]	=0x00000000	@x00004010
DAC_ControlStatus[CCURPMFC_DAC_1]	=0x00000000	@x00004014
DAC_ControlStatus[CCURPMFC_DAC_2]	=0x00000000	@x00004018
DAC_ControlStatus[CCURPMFC_DAC_3]	=0x00000000	@x0000401c
DAC_FifoCSR	=0x00000000	@x00004030
DAC_FifoThreshold	=0x00000000	@x00004034
DAC_FifoWriteCount	=0x00000000	@x00004038
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_0]	=0x00000000	@x00004100
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_1]	=0x00000000	@x00004104
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_2]	=0x00000000	@x00004108
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_3]	=0x00000000	@x0000410c
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_4]	=0x00000000	@x00004110
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_5]	=0x00000000	@x00004114
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_6]	=0x00000000	@x00004118
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_7]	=0x00000000	@x0000411c
DAC_GainCalibration[CCURPMFC_DAC_CHANNEL_8]	=0x00000000	@x00004120

```

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

```

===== 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

```

### 3.1.6 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 0x7ffff7fd0000 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)->

```

### 3.1.7 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)->
```

### **3.1.8 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]  
-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
```

### 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*

```

### 3.1.9 Flash/ccurpmfc\_flash

This program is used to burn new firmware. 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 -r[OutFile] -R -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
              : Force Write Flash: Do not abort Flash burn for header label
              : mismatch
-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
-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'
-w [InFile]    : Read input FPGA file and Flash the board
-X            : Use Full File. Do not truncate for firmware write

```

==== Notes ====

Board '-b' option is mandatory

Firmware reload '-R' or '-B' not allowed with '-i', '-L', '-r' or '-w' options

Firmware read flash '-r' not allowed with '-B', '-i', '-L', '-B' or '-w' options

Inquiry '-i' can be used '-w' options

```

e.g. ./ccurpmfc_flash -b0                (Query chip and on-board Flash)
     ./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 -X   (InFile ==> On-board FPGA -
                                           use entire file)
     ./ccurpmfc_flash -b0 -w InFile -L   (InFile ==> On-board FPGA -
                                           only license updated)
     ./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 -B         (Reload Firmware - i.e. power-
                                           cycle the card) - Base Level
     ./ccurpmfc_flash -b0 -R 200        (Reload Firmware - i.e. power-
                                           cycle the card) - sector 200

```

### 3.1.10 Flash/ccurpmfc\_label

This utility is only supplied for those customers that are creating their own firmware and need to install in on a RedHawk system. In its simplest form, the customer will request a License 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.

```
./ccurpmfc_label -d[OutputDirectory] -c[ChipName] -F -i[InputFile]
                  -L[LicenseFile] -m[MemberCode] -o[OutputFile]
                  -S[RunLevelSectorAddress]
-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)
-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
```

==== 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>/<BoardFunction>_<ChipName>_<MemberCode>_<RunLevel>.cust
```

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)
```

### 3.1.11 Flash/ccurpmfc\_dump\_license

This utility is allows the customer to dump the license information from a firmware (\*.cust) file.

```
./ccurpmfc_dump_license <Firmware file>
```

This utility only dumps the license information from the \*.cust file  
and not the \*.lic license file

e.g ./ccurpmfc\_dump\_license COS\_IpCoreCOS\_EPCQ256A5S100.cust

## 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)
-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 ===")
  @ 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)
  -m1D                 (Library: (FIFO) DMA mode)
  -mlp                 (Library: (Channel Registers) PIO
                        mode)
  -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)

```

**Example display:**

```
./ccurpmfc_adc -A -C0@150000.0/1@1234.0 -se/+
```

```

local_ptr=0x7ffff7fd7000
Physical Memory Information:
  UserPID           =22341
  PhysMemPtr        =0x352d9000
  DriverVirtMemPtr =0xffff8800352d9000
  MmappedUserMemPtr=0x7ffff7fcc000
  PhysMemSize       =0x00001000
  PhysMemSizeFreed =0x00000000
  EntryInTxTbl     =0
  NumOfEntriesUsed =1
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
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 #####

```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

    [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
=====

```

```

<----- (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]
-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])

```

```

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')

```

#### Example display:

```
./ccurpmfc_adc_calibrate -A
```

```

Device Name      : /dev/ccurpmfc0
Board Serial No: 98765 (0x000181cd)
Auto Calibration started...done. (2.067 seconds)

```

```
=====> Dump to 'stdout'
```

```
#Date           : Wed Jul 12 13:15:01 2017
```

```

#Chan  Negative          Offset          Positive
#====  =====          =====          =====
ch00:  0.99858815083280205727  0.0006103515625000000  0.99863700661808252335

```



```

ch01: 0.99849565094336867332 0.00030517578125000000 0.99847780773416161537
ch02: 0.99830451887100934982 0.00000000000000000000 0.99833975685760378838
ch03: 0.99862672341987490654 -0.00030517578125000000 0.99858157476410269737
ch04: 0.99798926850780844688 0.00030517578125000000 0.99801187450066208839
ch05: 0.99849147163331508636 0.00030517578125000000 0.99852089816704392433
ch06: 0.99974241713061928749 -0.00061035156250000000 0.99974604090675711632
ch07: 0.99841002095490694046 0.00000000000000000000 0.99846544535830616951
ch08: 0.99820307642221450806 0.00030517578125000000 0.99826521240174770355
ch09: 0.99752102326601743698 0.00000000000000000000 0.99755099555477499962
ch10: 0.99814050458371639252 0.00000000000000000000 0.99812307395040988922
ch11: 0.99766269559040665627 -0.00030517578125000000 0.99767517531290650368
ch12: 0.99766628351062536240 0.00000000000000000000 0.99769096216186881065
ch13: 0.99902953673154115677 0.00000000000000000000 0.99908065097406506538
ch14: 0.99805873911827802658 -0.00061035156250000000 0.99813077831640839577
ch15: 0.99856105307117104530 -0.00030517578125000000 0.99859240418300032616

```

### 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)
-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 -- default "=== None ===")
  @                     (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)
  -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.)

```

**Example display:**

```
./ccurpmfc_adc_fifo
```

```

local_ptr=0x7ffff7fd7000
      Number of Samples =49152
      Transfer Mode     =Library DMA Mode
      Physical Memory Information:
        UserPID         =26293
        PhysMemPtr      =0x3680000
        DriverVirtMemPtr=0xffff880003680000
        MmappedUserMemPtr=0x7ffff7f44000
        PhysMemSize     =0x00080000
        PhysMemSizeFreed=0x00000000
        EntryInTxTbl   =0
        NumOfEntriesUsed=1
      NumOfChannels=16, FirstChannel=0, LastChannel=15, NumAdc0Chans=8
      NumAdc1Chans=8

```

```
(** Enter <CONTROL-C> to Terminate **)
```

```

6800: usec=11097.26 (min=11095.58/max=11102.85/ave=11097.46) 17.72 MBytes/Sec -
EmptyCnt=92905 (71%)

```

```

=====
                        Date: Mon Oct 25 14:15:44 2017
      Expected Input Volts: === Not Specified ===
        Scan Counter: ADC0=31629312 ADC1=10543104
      Approx. Sample/Second: ADC0=299997 ADC1=99999
        NumberOfChans: ADC0=8 ADC1=8
=====

```

```

<----- (volts) ----->
Chan  Min      Max      Ave      DetectedCnt  TolerExeededCnt
=====
00    -0.0015   0.0003  -0.0002   31629312     -
01    -0.0012   0.0009  -0.0000   31629312     -
02    -0.0012   0.0003  -0.0003   31629312     -
03    -0.0012   0.0003  -0.0002   31629312     -
04    -0.0006   0.0009   0.0002   31629312     -
05    -0.0009   0.0009   0.0001   31629312     -
06    -0.0012   0.0003  -0.0002   31629312     -
07    -0.0009   0.0006  -0.0000   31629312     -
08     9.9072   9.9124   9.9099   10543104     -
09     9.9075   9.9127   9.9100   10543104     -
10     9.9072   9.9124   9.9098   10543104     -
11     9.9075   9.9127   9.9100   10543104     -
12     9.9072   9.9124   9.9097   10543104     -
13     9.9075   9.9124   9.9099   10543104     -
14     9.9075   9.9127   9.9100   10543104     -
15     9.9072   9.9127   9.9100   10543104     -
=====

```

### 3.2.4 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]
-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)

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
      (Reset all clocks and then set Clock 0 to 100000 SPS and
      Clock 4 to 12345 SPS)
```

#### Example display:

```
./ccurpmfc_clock -R -C0@100000 -C4@12345

Board Number [-b]: 0
Delay [-d]: 10 milli-seconds
Loop Count [-l]: ***Forever***
Scan Count: 693

_____ 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: 1234.000 Samples/Second/Channel
User output clock frequency 2: *** Not Set ***
User output clock frequency 3: *** Not Set ***
User output clock frequency 4: *** Not Set ***
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
```

### 3.2.5 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)
  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)
  -v 2.5/7.5,12.7 (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 )
```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

        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)

```

### Example display:

```
./ccurpmfc_dac -A -os -s7 -Vb5 -v4.5 -E4.5
```

```

local_ptr=0x7ffff7fd7000
Physical Memory Information:
  UserPID          =27364
  PhysMemPtr       =0x1c2000
  DriverVirtMemPtr=0xffff88000001c2000
  MmappedUserMemPtr=0x7ffff7fcc000
  PhysMemSize      =0x00001000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl    =0
  NumOfEntriesUsed=1
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
Tolerance Exceeded Count : 0 (=== Passed ===)

```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

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)
Tolerance Exceed Count: 0
Scan Counter: 827984

```

```

=====
<----- (volts) ----->
Chan  Min      Max      Ave      TolerExceededCnt
=====
07   -0.0037   0.0020  -0.0008      -
=====

```

./ccurpmfc\_dac -C0 -Z

```

local_ptr=0x7ffff7fd7000
Physical Memory Information:
UserPID      =27381
PhysMemPtr   =0x86439000
DriverVirtMemPtr=0xffff880086439000
MmappedUserMemPtr=0x7ffff7fcc000
PhysMemSize  =0x00001000
PhysMemSizeFreed=0x00000000
EntryInTxTbl =0
NumOfEntriesUsed=1

```

```

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)

```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

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  
 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
=====

```

```

<----- (volts) ----->
Chan Min Max Ave TolerExeededCnt

```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```
=====
00      4.4969      4.5021      4.4997      -
=====
```

### 3.2.6 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 initiating calibration.

```
Usage: ./ccurpmfc_dac_calibrate [-A] [-b board] [-c ChanMask] [-f DataFormat]
      [-i inCalFile] [-o outCalFile] [-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])
-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')
```

#### Example display:

```
./ccurpmfc_dac_calibrate -A -oOutputCal
```

```
Device Name      : /dev/ccurpmfc0
Board Serial No: 98765 (0x000181cd)
Auto Calibration started...done. (2.527 seconds)
```

```
==> Dump of 'OutputCal' file
#Date           : Wed Jul 12 15:12:06 2017
```

#Chan	Gain	Offset
=====	=====	=====
ch00:	0.0022888183593750	0.0000000000000000
ch01:	0.0022888183593750	0.0000000000000000
ch02:	0.0013732910156250	0.0000000000000000
ch03:	0.0012207031250000	0.0000000000000000
ch04:	0.0010681152343750	0.0000000000000000
ch05:	0.0012207031250000	0.0000000000000000
ch06:	0.0022888183593750	0.0000000000000000
ch07:	0.0022888183593750	0.0000000000000000
ch08:	0.0013732910156250	0.0000000000000000
ch09:	0.0010681152343750	0.0000000000000000
ch10:	0.0018310546875000	0.0000000000000000
ch11:	0.0018310546875000	0.0000000000000000
ch12:	0.0018310546875000	0.0000000000000000
ch13:	0.0015258789062500	0.0000000000000000
ch14:	0.0024414062500000	0.0000000000000000
ch15:	0.0022888183593750	0.0000000000000000

```
==> Board calibration data written to 'OutputCal' file
```



### 3.2.7 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)
  -v 2.5/7.5,12.7 (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)
  -V b5/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)
-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, dacl&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
  MmappedUserMemPtr=0x7ffff7fbc000
  PhysMemSize    =0x00008000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl  =0
  NumOfEntriesUsed=1
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

### 3.2.8 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     : 0xFFFFFFFF (All Output)
DIO Enable        : 0x00000001 (Enable)
```

```

Input Snapshot          : 0x00000001 (Snapshot)
Scan Count              : 183

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
[4]  skip skip skip skip skip skip skip skip skip
[5]  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 ===

### 3.2.9 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:

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

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
      DMA Counts (dma0..0):           0          0
      DIO COS Counts (group2..0):      6520825    6520825    6520825
      Overflow DIO COS Counts (group2..0): 0          0          0
```

	[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.	0.	0.	0.	0.

	[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.	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. 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. 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 ###

```

### 3.2.10 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)
  -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
  MmappedUserMemPtr=0x7ffff7fdc000
  PhysMemSize  =0x00001000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl =0
  NumOfEntriesUsed=1
-----
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]: 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 05252016 00550003 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

```

### 3.2.11 lib/ccurpmfc\_dma

This test transfers data from physical memory to the Local register area and back. There are two modes of operation. One is DMA and the other 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. 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 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)
  -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)

```

#### Example display:

```

./ccurpmfc_dma

Device Name: /dev/ccurpmfc0
local_ptr=0x7ffff7fe7000

```



```

Physical Memory Information:
  UserPID          =18950
  PhysMemPtr       =0x5ea00000
  DriverVirtMemPtr=0xffff88005ea00000
  MmappedUserMemPtr=0x7ffff70f4000
  PhysMemSize      =0x00200000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl     =0
  NumOfEntriesUsed=2
### 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

```

### 3.2.12 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

local_ptr=0x7ffff7fe7000
Physical Memory Information:
  UserPID          =18954
  PhysMemPtr       =0x8566b000
  DriverVirtMemPtr=0xffff88008566b000
  MmappedUserMemPtr=0x7ffff7fdb000
  PhysMemSize      =0x00001000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl     =1
  NumOfEntriesUsed=1
### Configuring ADC ###
- Activate ADC
- Configure ADC
- 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 ==== (length=64)
+DMP+      0  0000ffff  00000001  0000ffff  0000ffff  *.....*
+DMP+     0x10  00000000  00000000  00000000  00000000  *.....*
+DMP+     0x20  00000000  0000ffff  0000ffff  00000000  *.....*
+DMP+     0x30  0000ffff  0000ffff  0000ffff  00000000  *.....*
### 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
- 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=0xaaaaaaaaab
CCURPMFC_DIO_CHAN_32_63=0x0000aaab
### Writing DIO Channels 48..95 ###
CCURPMFC_DIO_CHAN_32_63=0xbabe0000
CCURPMFC_DIO_CHAN_64_95=0xfeedface

```

### 3.2.13 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 restrictions)

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

```

### 3.2.14 lib/ccurpmfc\_identify

This test is useful in identifying a particular card by displaying its LED.

```

Usage: ./ccurpmfc_identify -[bsx]
    -b <board>          (board #, default = 0)
    -s <seconds>        (seconds to sleep, default = 10)
    -s 0                (Identify Board: DISABLE)
    -s <negative value> (Identify Board: ENABLE forever)
    -x                  (silent)

```

Example display:

```

./ccurpmfc_identify

Device Name      : /dev/ccurpmfc0
Board ID        : 9290
Board Type       : 01
Board Function   : 01
Board Serial No: 98765 (0x000181cd)

Identify ENABLED on board 0 (LED should start flashing)
Sleeping for 10 seconds...done
Identify DISABLED on board 0 (LED should stop flashing)

```

### 3.2.15 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) FmFlvCod IPCores
                                Temp:C/F SerialNo RLS# Func
0 56 Y 07:00:00 1542:1542 9290.01.01:0100 0055.0003(05/25/16 00:00:00) 00000000 0
                                36/ 96.8 012345 100 MultFunc

```

`./ccurpmfc_info -l`

```

##### Board 0 #####
Version: 22.7_2
Build: Dec 29 2017, 11:53:22
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))
Sub-Device ID: 0x0100
Firmware Date/Time: 0x10142016 0x00000000 (10/14/2016 00:00:00)
Firmware Revision: 0x00030002 (0003.0002)
Firmware Flavor Code: 0x00000000 (0) ( )
Number of Advanced IP Cores: 0x00000000 (0)
Board Serial Number: 0x000181cd (98765)
Board SPROM Revision: 0x0000 (0)
FPGA Chip Temperature: 0x25 (37 degree C, 98.6 degree F)
Run Level Sector Number: 0x64 (100)
Multi-Firmware Support: 0x1 (Yes)
MSI Support: Enabled
IRQ Level: 57
Calibration Reference: 9.91 Volts

```

```

##### Board 1 #####
Version: 22.7_2
Build: Dec 29 2017, 11:53:22
Module: ccurpmfc
Board Index: 0 (PCIe-CCUR_FPGA_PMFC)
Bus: 0x05
Slot: 0x00
Func: 0x00
Vendor ID: 0x1542
Sub-Vendor ID: 0x1542
Board Info: 0x92900020 (id=9290, type=0x00, func=0x20 (IpCoreCOS))
Sub-Device ID: 0x0100
Firmware Date/Time: 0x09292016 0x00093204 (09/29/2016 09:32:04)
Firmware Revision: 0x00050002 (0005.0002)
Firmware Flavor Code: 0x47453031 (1195716657) (GE01)
Number of Advanced IP Cores: 0x00000001 (1)
Board Serial Number: 0x000a33b7 (668599)
Board SPROM Revision: 0x0000 (0)
FPGA Chip Temperature: 0x26 (38 degree C, 100.4 degree F)
Run Level Sector Number: 0x64 (100)
Multi-Firmware Support: 0x1 (Yes)
MSI Support: Enabled
IRQ Level: 57
Calibration Reference: 9.91 Volts

```

`./ccurpmfc_info -l -v`

```

##### Board 0 #####
      Version: 22.7_2
      Build: Dec 29 2017, 11:53:22
      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))
      Sub-Device ID: 0x0100
      Firmware Date/Time: 0x10142016 0x00000000 (10/14/2016 00:00:00)
      Firmware Revision: 0x00030002 (0003.0002)
      Firmware Flavor Code: 0x00000000 (0) ()
Number of Advanced IP Cores: 0x00000000 (0)
      Board Serial Number: 0x000181cd (98765)
      Board SPROM Revision: 0x0000 (0)
      FPGA Chip Temperature: 0x25 (37 degree C, 98.6 degree F)
      Run Level Sector Number: 0x64 (100)
      Multi-Firmware Support: 0x1 (Yes)
      MSI Support: Enabled
      IRQ Level: 57
      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
      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=0xc7b10000 Size=32768 (0x8000)  
Region 2: Addr=0xc7b00000 Size=65536 (0x10000)

##### Board 1 #####

Version: 22.7\_2  
Build: Dec 29 2017, 11:53:22

Module: ccurpmfc  
Board Index: 0 (PCIe-CCUR\_FPGA\_PMFC)  
Bus: 0x05  
Slot: 0x00  
Func: 0x00  
Vendor ID: 0x1542  
Sub-Vendor ID: 0x1542  
Board Info: 0x92900020 (id=9290, type=0x00, func=0x20 (IpCoreCOS))  
Sub-Device ID: 0x0100  
Firmware Date/Time: 0x09292016 0x00093204 (09/29/2016 09:32:04)  
Firmware Revision: 0x00050002 (0005.0002)  
Firmware Flavor Code: 0x47453031 (1195716657) (GE01)  
Number of Advanced IP Cores: 0x00000001 (1)  
Board Serial Number: 0x000a33b7 (668599)  
Board SPROM Revision: 0x0000 (0)  
FPGA Chip Temperature: 0x26 (38 degree C, 100.4 degree F)  
Run Level Sector Number: 0x64 (100)  
Multi-Firmware Support: 0x1 (Yes)  
MSI Support: Enabled  
IRQ Level: 57  
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
    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]: 0x0000000c ([12]: CCURPMFC_IPCODE_CHANGE_OF_STATE -
Change of State)
        IpCore Revision [00]: 0x00030001 (0003.0001)
        IpCore Offset [00]: 0x00013400
        IpCore Information [00]: 0x00000001
        IpCore Mapped Pointer [00]: 0x7ffff7fea400

    ---Memory Regions Information---
        Region 0: Addr=0xc4920000 Size=32768 (0x8000)
        Region 2: Addr=0xc4900000 Size=131072 (0x20000)

```

### 3.2.16 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
local_ptr=0x7ffff7fe7000

```

```

Size of Avalon RAM = 32768 (0x00008000)
Physical Memory Information:
  UserPID      =19913
  PhysMemPtr   =0x799c8000
  DriverVirtMemPtr=0xffff8800799c8000
  MmappedUserMemPtr=0x7ffff7fd5000
  PhysMemSize  =0x00008000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl =0
  NumOfEntriesUsed=1

1: Memory -> Avalon RAM (DMA0) (Size=0x8000): 100 (1358.47 us, 24.12 MBytes/Sec)
2: Memory -> Avalon RAM (DMA1) (Size=0x8000): 100 (1358.11 us, 24.13 MBytes/Sec)
3: Memory -> Avalon RAM (PIO) (Size=0x8000): 100 (688.09 us, 47.62 MBytes/Sec)
4: Avalon RAM -> Memory (DMA0) (Size=0x8000): 100 (1509.97 us, 21.70 MBytes/Sec)
5: Avalon RAM -> Memory (DMA1) (Size=0x8000): 100 (1509.92 us, 21.70 MBytes/Sec)
6: Avalon RAM -> Memory (PIO) (Size=0x8000): 100 (6192.22 us, 5.29 MBytes/Sec)
7: Memory -> Avalon ADC Calibration (DMA0) (Size=0x40): 10000 (9.39 us, 6.82 MBytes/Sec)
8: Memory -> Avalon ADC Calibration (DMA1) (Size=0x40): 10000 (9.40 us, 6.81 MBytes/Sec)
9: Memory -> Avalon ADC Calibration (PIO) (Size=0x40): 10000 (0.18 us, 355.51 MBytes/Sec)
10: Avalon ADC Calibration -> Memory (DMA0) (Size=0x40): 10000 (9.98 us, 6.41 MBytes/Sec)
11: Avalon ADC Calibration -> Memory (DMA1) (Size=0x40): 10000 (9.99 us, 6.41 MBytes/Sec)
12: Avalon ADC Calibration -> Memory (PIO) (Size=0x40): 10000 (13.61 us, 4.70 MBytes/Sec)
13: Memory -> Avalon SDRAM(FIFO) (DMA0) (Size=0x8000): 100 (1358.64 us, 24.12 MBytes/Sec)
14: Memory -> Avalon SDRAM(FIFO) (DMA1) (Size=0x8000): 100 (1358.43 us, 24.12 MBytes/Sec)
15: Memory -> Avalon SDRAM(FIFO) (PIO) (Size=0x8000): 100 (1388.21 us, 23.60 MBytes/Sec)
16: Avalon SDRAM(FIFO) -> Memory (DMA0) (Size=0x8000): 100 (2005.77 us, 16.34 MBytes/Sec)
17: Avalon SDRAM(FIFO) -> Memory (DMA1) (Size=0x8000): 100 (2005.88 us, 16.34 MBytes/Sec)
18: Avalon SDRAM(FIFO) -> Memory (PIO) (Size=0x8000): 100 (14340.24 us, 2.29 MBytes/Sec)
**** Test Passed ****

```

### 3.2.17 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
03 = Clear Library Error
05 = Display CONFIG Registers
07 = Get All Boards Driver Information
09 = Get Board Information
11 = Get Driver Information
13 = Get Mapped Config Pointer
15 = Get Mapped Local Pointer
17 = Get Test Bus Control
19 = Initialize Board
21 = Munmap Physical Memory
23 = Reset Board
25 = Set Test Bus Control
27 = ### ADC MENU ###
29 = ### CLOCK GENERATOR MENU ###
31 = ### DIO MENU ###
33 = ### IP CORE MENU ###
35 = ### SPROM MENU ###
02 = Clear Driver Error
04 = Display BOARD Registers
06 = Dump Physical Memory List
08 = Get Board CSR
10 = Get Driver Error
12 = Get Library Error
14 = Get Mapped Driver/Library Pointer
16 = Get Physical Memory
18 = Get Value
20 = MMap Physical Memory
22 = Reload Firmware
24 = Set Board CSR
26 = Set Value
28 = ### CALIBRATION MENU ###
30 = ### DAC MENU ###
32 = ### INTERRUPT MENU ###
34 = ### SDRAM 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 Driver Read Operation	04 = ADC Get CSR
05 = ADC Get Driver Read Mode	06 = ADC Get FIFO Channel Select
07 = ADC Get FIFO Information	08 = ADC Get FIFO Threshold
09 = ADC Read Channels	10 = ADC Reset FIFO
11 = ADC Set CSR	12 = ADC Set Driver Read Mode
13 = ADC Set FIFO Channel Select	14 = 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 Driver Write Operation	04 = DAC Get CSR
05 = DAC Get Driver Write Mode	06 = DAC Get FIFO Channel Select
07 = DAC Get FIFO Information	08 = DAC Get FIFO Threshold
09 = DAC Get FIFO Write Count	10 = DAC Get Update Select
11 = DAC ReadBack Channels	12 = DAC Read Channels
13 = DAC Reset FIFO	14 = DAC Set CSR
15 = DAC Set Driver Write Mode	16 = DAC Set FIFO Channel Select
17 = DAC Set FIFO Threshold	18 = DAC Set FIFO Write Count
19 = DAC Set Update Select	20 = 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 Get Channels Polarity	04 = DIO Get COS Channels Edge Sense
05 = DIO Get COS Channels Enable	06 = DIO Get COS Channels Mode
07 = DIO Get COS Channels Overflow	08 = DIO Get COS Channels Status
09 = DIO Get Input Channels Filter	10 = DIO Get Input Snapshot
11 = DIO Get Mode	12 = DIO Get Output Sync
13 = DIO Get Ports Direction	14 = DIO Information

```

15 = DIO Read Custom Channels          16 = DIO Read Input Channels
17 = DIO Read Output Channels         18 = DIO Set Channels Polarity
19 = DIO Set COS Channels Edge Sense  20 = DIO Set COS Channels Enable
21 = DIO Set COS Channels Mode        22 = DIO Set Input Channels Filter
23 = DIO Set Input Snapshot           24 = DIO Set Mode
25 = DIO Set Output Sync              26 = DIO Set Ports Direction
27 = DIO Set Ports Direction to Inputs 28 = DIO Set Ports Direction to Outputs
29 = DIO Write Output Channels        30 = DIO Write Output Channels High
31 = 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)->
-----

```

### 3.2.18 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 -- AAAAAAAAA/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

```

### 3.2.19 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*