

Software Interface

CCRTNGFC (WC-CP-FIO2)

PCIe Next Generation FPGA I/O Card

<i>Driver</i>	cctrngfc (WC-CP-FIO2)	
<i>OS</i>	RedHawk (CentOS/Rocky or Ubuntu based)	
<i>Vendor</i>	Concurrent Real-Time	
<i>Hardware</i>	PCIe Programmable Multi-Function Card (CP-FPGA-4 & 5)	
<i>Author</i>	Darius Dubash	
<i>Date</i>	March 15 th , 2024	Rev 2024.1



This page intentionally left blank

Table of Contents

1. INTRODUCTION	8
1.1 Related Documents	8
2. SOFTWARE SUPPORT	8
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
2.2 Application Program Interface (API) Access	12
2.2.1 ccrtNGFC_Add_Irq().....	12
2.2.2 ccrtNGFC_BoardExpirationTimeRemaining().....	12
2.2.3 ccrtNGFC_Clear_Driver_Error().....	13
2.2.4 ccrtNGFC_Clear_Interrupt_Status().....	13
2.2.5 ccrtNGFC_Clear_Lib_Error().....	14
2.2.6 ccrtNGFC_Clock_Generator_Soft_Reset()	14
2.2.7 ccrtNGFC_Clock_Get_Generator_CSR()	14
2.2.8 ccrtNGFC_Clock_Get_Generator_Info()	15
2.2.9 ccrtNGFC_Clock_Get_Generator_Input_Clock_Enable()	18
2.2.10 ccrtNGFC_Clock_Get_Generator_Input_Clock_Select()	18
2.2.11 ccrtNGFC_Clock_Get_Generator_Input_Clock_Status()	19
2.2.12 ccrtNGFC_Clock_Get_Generator_M_Divider()	20
2.2.13 ccrtNGFC_Clock_Get_Generator_N_Divider()	20
2.2.14 ccrtNGFC_Clock_Get_Generator_Output_Config()	21
2.2.15 ccrtNGFC_Clock_Get_Generator_Output_Format()	21
2.2.16 ccrtNGFC_Clock_Get_Generator_Output_Mode().....	22
2.2.17 ccrtNGFC_Clock_Get_Generator_Output_Mux()	23
2.2.18 ccrtNGFC_Clock_Get_Generator_P_Divider().....	24
2.2.19 ccrtNGFC_Clock_Get_Generator_P_Divider_Enable().....	24
2.2.20 ccrtNGFC_Clock_Get_Generator_R_Divider()	25
2.2.21 ccrtNGFC_Clock_Get_Generator_Revision().....	25
2.2.22 ccrtNGFC_Clock_Get_Generator_Value()	26
2.2.23 ccrtNGFC_Clock_Get_Generator_Voltage_Select().....	26
2.2.24 ccrtNGFC_Clock_Get_Generator_Zero_Delay().....	27
2.2.25 ccrtNGFC_Clock_PLL_CSR().....	27
2.2.26 ccrtNGFC_Clock_ReturnOutputFrequency().....	28
2.2.27 ccrtNGFC_Clock_Set_Generator_CSR()	28
2.2.28 ccrtNGFC_Clock_Set_Generator_Input_Clock_Enable().....	29
2.2.29 ccrtNGFC_Clock_Set_Generator_Input_Clock_Select()	30
2.2.30 ccrtNGFC_Clock_Set_Generator_M_Divider()	30
2.2.31 ccrtNGFC_Clock_Set_Generator_N_Divider().....	31
2.2.32 ccrtNGFC_Clock_Set_Generator_Output_Config().....	31
2.2.33 ccrtNGFC_Clock_Set_Generator_Output_Format()	32
2.2.34 ccrtNGFC_Clock_Set_Generator_Output_Mode()	33
2.2.35 ccrtNGFC_Clock_Set_Generator_Output_Mux()	34
2.2.36 ccrtNGFC_Clock_Set_Generator_P_Divider()	35
2.2.37 ccrtNGFC_Clock_Set_Generator_P_Divider_Enable()	35
2.2.38 ccrtNGFC_Clock_Set_Generator_R_Divider().....	36
2.2.39 ccrtNGFC_Clock_Set_Generator_Value()	36
2.2.40 ccrtNGFC_Clock_Set_Generator_Voltage_Select()	37
2.2.41 ccrtNGFC_Clock_Set_Generator_Zero_Delay().....	37
2.2.42 ccrtNGFC_Close().....	38

2.2.43	ccrtNGFC_Compute_All_Output_Clocks()	38
2.2.44	ccrtNGFC_Convert_Physmem2avmm_Address()	39
2.2.45	ccrtNGFC_Create_UserLDioCosInterruptHandler()	40
2.2.46	ccrtNGFC_Create_UserProcess()	41
2.2.47	ccrtNGFC_DataToVolts()	42
2.2.48	ccrtNGFC_DataToVoltsLxxFormat()	43
2.2.49	ccrtNGFC_DaughterCard_EEPROM_Read()	43
2.2.50	ccrtNGFC_DaughterCard_EEPROM_Write()	44
2.2.51	ccrtNGFC_Destroy_AllUserProcess()	44
2.2.52	ccrtNGFC_Destroy_UserLDioCosInterruptHandler()	45
2.2.53	ccrtNGFC_Destroy_UserProcess()	45
2.2.54	ccrtNGFC_DIO_Get_Channels_Terminator()	45
2.2.55	ccrtNGFC_DIO_Get_Ports_Direction()	47
2.2.56	ccrtNGFC_DIO_Set_Channels_Terminator()	49
2.2.57	ccrtNGFC_DIO_Set_Channels_Terminator_To_Off()	51
2.2.58	ccrtNGFC_DIO_Set_Channels_Terminator_To_On()	52
2.2.59	ccrtNGFC_DIO_Set_Ports_Direction()	53
2.2.60	ccrtNGFC_DIO_Set_Ports_Direction_To_Input()	55
2.2.61	ccrtNGFC_DIO_Set_Ports_Direction_To_Output()	57
2.2.62	ccrtNGFC_Disable_Pci_Interrupts()	58
2.2.63	ccrtNGFC_Enable_Pci_Interrupts()	59
2.2.64	ccrtNGFC_Fast_Memcpy()	59
2.2.65	ccrtNGFC_Fast_Memcpy_Unlocked()	60
2.2.66	ccrtNGFC_Fast_Memcpy_Unlocked_FIFO()	60
2.2.67	ccrtNGFC_Fraction_To_Hex()	61
2.2.68	ccrtNGFC_Get_All_Boards_Driver_Info()	61
2.2.69	ccrtNGFC_Get_Board_CSR()	63
2.2.70	ccrtNGFC_Get_Board_Info()	63
2.2.71	ccrtNGFC_Get_Driver_Error()	64
2.2.72	ccrtNGFC_Get_Driver_Info()	65
2.2.73	ccrtNGFC_Get_Interrupt_Status()	67
2.2.74	ccrtNGFC_Get_Interrupt_Timeout_Seconds()	67
2.2.75	ccrtNGFC_Get_Lib_Error()	68
2.2.76	ccrtNGFC_Get_Lib_Error_Description()	70
2.2.77	ccrtNGFC_Get_Library_Info()	70
2.2.78	ccrtNGFC_Get_Mapped_Config_Ptr()	71
2.2.79	ccrtNGFC_Get_Mapped_Driver_Library_Ptr()	72
2.2.80	ccrtNGFC_Get_Mapped_Local_Ptr()	72
2.2.81	ccrtNGFC_Get_Open_File_Descriptor()	73
2.2.82	ccrtNGFC_Get_Physical_Memory()	73
2.2.83	ccrtNGFC_Get_PowerModule_Info()	74
2.2.84	ccrtNGFC_Get_RunCount_UserProcess()	74
2.2.85	ccrtNGFC_Get_Value()	75
2.2.86	ccrtNGFC_Hex_To_Fraction()	75
2.2.87	ccrtNGFC_Identify_Board()	75
2.2.88	ccrtNGFC_Initialize_Board()	76
2.2.89	ccrtNGFC_IpCore_Get_Info()	76
2.2.90	ccrtNGFC_IpCore_Get_Mapped_Ptr()	77
2.2.91	ccrtNGFC_LDIO_Activate()	78
2.2.92	ccrtNGFC_LDIO_Get_Channels_Polarity()	79
2.2.93	ccrtNGFC_LDIO_Get_COS_Channels_Edge_Sense()	81
2.2.94	ccrtNGFC_LDIO_Get_COS_Channels_Enable()	83
2.2.95	ccrtNGFC_LDIO_Get_COS_Channels_Mode()	85
2.2.96	ccrtNGFC_LDIO_Get_COS_Channels_Overflow()	87
2.2.97	ccrtNGFC_LDIO_Get_COS_Channels_Status()	89
2.2.98	ccrtNGFC_LDIO_Get_Input_Channels_Filter()	91

2.2.99	ccrtNGFC_LDIO_Get_Input_Snapshot()	93
2.2.100	ccrtNGFC_LDIO_Get_Output_Sync()	93
2.2.101	ccrtNGFC_LDIO_Read_Input_Channel_Registers()	94
2.2.102	ccrtNGFC_LDIO_Read_Output_Channel_Registers()	96
2.2.103	ccrtNGFC_LDIO_Set_Channels_Polarity()	98
2.2.104	ccrtNGFC_LDIO_Set_COS_Channels_Edge_Sense()	100
2.2.105	ccrtNGFC_LDIO_Set_COS_Channels_Enable()	102
2.2.106	ccrtNGFC_LDIO_Set_COS_Channels_Mode()	104
2.2.107	ccrtNGFC_LDIO_Set_Input_Channels_Filter()	106
2.2.108	ccrtNGFC_LDIO_Set_Input_Snapshot()	108
2.2.109	ccrtNGFC_LDIO_Set_Output_Sync()	109
2.2.110	ccrtNGFC_LDIO_Write_Output_Channel_Registers()	110
2.2.111	ccrtNGFC_LDIO_Write_Output_Channel_Registers_High()	112
2.2.112	ccrtNGFC_LDIO_Write_Output_Channel_Registers_Low()	114
2.2.113	ccrtNGFC_LIO_On_Off()	116
2.2.114	ccrtNGFC_LIO_Get_Ports_Direction()	117
2.2.115	ccrtNGFC_LIO_Read_Output_Loopbacked_Channels()	118
2.2.116	ccrtNGFC_LIO_Set_Ports_Direction()	120
2.2.117	ccrtNGFC_LIO_Set_Ports_Direction_To_Input()	122
2.2.118	ccrtNGFC_LIO_Set_Ports_Direction_To_Output()	123
2.2.119	ccrtNGFC_MMap_Physical_Memory()	124
2.2.120	ccrtNGFC_MsgDma_Clone() (<i>US Patent No.: US 11,281,584 B1</i>)	125
2.2.121	ccrtNGFC_MsgDma_Configure_Descriptor()	127
2.2.122	ccrtNGFC_MsgDma_Configure_Single()	129
2.2.123	ccrtNGFC_MsgDma_Fire()	130
2.2.124	ccrtNGFC_MsgDma_Fire_Fifo()	131
2.2.125	ccrtNGFC_MsgDma_Fire_Single()	132
2.2.126	ccrtNGFC_MsgDma_Free_Descriptor()	133
2.2.127	ccrtNGFC_MsgDma_Get_Descriptor()	134
2.2.128	ccrtNGFC_MsgDma_Get_Dispatcher_CSR()	135
2.2.129	ccrtNGFC_MsgDma_Get_Info()	136
2.2.130	ccrtNGFC_MsgDma_Get_Prefetcher_CSR()	137
2.2.131	ccrtNGFC_MsgDma_Release()	138
2.2.132	ccrtNGFC_MsgDma_Seize()	139
2.2.133	ccrtNGFC_MsgDma_Setup()	140
2.2.134	ccrtNGFC_Munmap_Physical_Memory()	141
2.2.135	ccrtNGFC_NanoDelay()	142
2.2.136	ccrtNGFC_Open()	142
2.2.137	ccrtNGFC_Pause_UserProcess()	143
2.2.138	ccrtNGFC_PowerModule_Control()	143
2.2.139	ccrtNGFC_Program_All_Output_Clocks()	143
2.2.140	ccrtNGFC_Read()	145
2.2.141	ccrtNGFC_Reload_Firmware()	146
2.2.142	ccrtNGFC_Remove_Irq()	146
2.2.143	ccrtNGFC_Reset_Board()	146
2.2.144	ccrtNGFC_Reset_Clock()	147
2.2.145	ccrtNGFC_Resume_UserProcess()	147
2.2.146	ccrtNGFC_Return_Board_Info_Description()	147
2.2.147	ccrtNGFC_Set_Board_CSR()	148
2.2.148	ccrtNGFC_Set_Interrupt_Timeout_Seconds()	148
2.2.149	ccrtNGFC_Set_Value()	149
2.2.150	ccrtNGFC_Transfer_Data()	149
2.2.151	ccrtNGFC_Update_Clock_Generator_Divider()	151
2.2.152	ccrtNGFC_UserProcess_Command()	151
2.2.153	ccrtNGFC_VoltsToData()	152
2.2.154	ccrtNGFC_Wait_For_Interrupt()	152

3. TEST PROGRAMS.....	154
3.1 Direct Driver Access Example Tests	154
3.1.1 ccrtnngfc_disp.....	154
3.1.2 ccrtnngfc_dump	155
3.1.3 ccrtnngfc_rdreg.....	158
3.1.4 ccrtnngfc_reg.....	159
3.1.5 ccrtnngfc_regedit	169
3.1.6 ccrtnngfc_tst	169
3.1.7 ccrtnngfc_wreg	170
3.1.8 Flash/ccrtnngfc_flash.....	171
3.1.9 Flash/ccrtnngfc_label.....	172
3.1.10 Flash/ccrtnngfc_dump_license	173
3.2 Application Program Interface (API) Access Example Tests	174
3.2.1 lib/ccrtnngfc_acquire_physmem.....	174
3.2.2 lib/ccrtnngfc_adc	174
3.2.3 lib/ccrtnngfc_adc_calibrate.....	177
3.2.4 lib/ccrtnngfc_adc_fifo.....	178
3.2.5 lib/ccrtnngfc_adc_sps	182
3.2.6 lib/ccrtnngfc_check_bus	183
3.2.7 lib/ccrtnngfc_clock	184
3.2.8 lib/ccrtnngfc_dac	185
3.2.9 lib/ccrtnngfc_dac_calibrate.....	190
3.2.10 lib/ccrtnngfc_dac_fifo.....	191
3.2.11 lib/ccrtnngfc_dac_setchan	195
3.2.12 lib/ccrtnngfc_daughtercard_info.....	197
3.2.13 lib/ccrtnngfc_dio.....	198
3.2.14 lib/ccrtnngfc_disp	199
3.2.15 lib/ccrtnngfc_dma	201
3.2.16 lib/ccrtnngfc_example	202
3.2.17 lib/ccrtnngfc_expires	204
3.2.18 lib/ccrtnngfc_identify.....	205
3.2.19 lib/ccrtnngfc_info	206
3.2.20 lib/ccrtnngfc_ldio_intr	208
3.2.21 lib/ccrtnngfc_lio.....	212
3.2.22 lib/ccrtnngfc_msgdma	214
3.2.23 lib/ccrtnngfc_msgdma_clone.....	216
3.2.24 lib/ccrtnngfc_msgdma_info	222
3.2.25 lib/ccrtnngfc_msgdma_multi_clone.....	225
3.2.26 lib/ccrtnngfc_sensors	228
3.2.27 lib/ccrtnngfc_smp_affinity.....	229
3.2.28 lib/ccrtnngfc_transfer.....	229
3.2.29 lib/ccrtnngfc_tst_lib	230

This page intentionally left blank

1. Introduction

This document provides the software interface to the *ccrtngfc* driver which communicates with the Concurrent Real-Time PCI Express Next Generation FPGA I/O Card (NGFC). Low-level programming is contained in the *Concurrent Real-Time PCI Express Next Generation FPGA I/O Cards (NGFC) Design Specification (No. 0610111)* which is a CCRT internal document that is not supplied to the customer.

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

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

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

1.1 Related Documents

- PCIe Next Generation FPGA Driver Installation on RedHawk Release Notes by Concurrent Real-Time.
- PCIe Next Generation FPGA Driver Technical Guide by Concurrent Real-Time.
- PCIe Next Generation FPGA Card I/O (NGFC) Design Specification (No. 0610111) by Concurrent Real-Time (*internal document*).

2. Software Support

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

2.1 Direct Driver Access

2.1.1 *open(2)* system call

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

```
int fp;  
fp = open("/dev/ccrtngfc0", 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/ccrtngfc<num>"* where *num* is a digit 0..39 which represents the board number that is to be accessed. Basically, the driver only allows one application to open a board at a time. The reason for this is that the application can have full access to the card, even at the board and API level. If another application were to communicate with the same card concurrently, the results would be unpredictable unless proper synchronization between applications is performed external to the driver API.

This driver allows multiple applications to open the same board by specifying an additional *oflag O_APPEND*. It is then the responsibility of the user to ensure that the various applications communicating with the same cards are properly synchronized. Various tests supplied in this package has the *O_APPEND* flags enabled, however, it is

strongly recommended that only one application be run with a single card at a time, unless the user is well aware of how the applications are going to interact with each other and accept any unpredictable results.

Each Next Generation FPGA card can have installed in it one to two daughter cards that are capable of having their own unique capabilities. They can also be opened via the *open(2)* system call, however, there is a lot of dependencies and interactions between the main and daughter cards, therefore, that information will not be discussed. In this case, when attempting to access daughter cards, you must open the main board and the daughter cards via the supplied API calls.

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_CCRTNGFC_ADD_IRQ  
IOCTL_CCRTNGFC_DISABLE_PCI_INTERRUPTS  
IOCTL_CCRTNGFC_ENABLE_PCI_INTERRUPTS  
IOCTL_CCRTNGFC_GET_DRIVER_ERROR  
IOCTL_CCRTNGFC_GET_DRIVER_INFO  
IOCTL_CCRTNGFC_GET_PHYSICAL_MEMORY  
IOCTL_CCRTNGFC_INIT_BOARD  
IOCTL_CCRTNGFC_INTERRUPT_TIMEOUT_SECONDS  
IOCTL_CCRTNGFC_MMAP_SELECT  
IOCTL_CCRTNGFC_NO_COMMAND  
IOCTL_CCRTNGFC_PCI_CONFIG_REGISTERS  
IOCTL_CCRTNGFC_REMOVE_IRQ  
IOCTL_CCRTNGFC_RESET_BOARD  
IOCTL_CCRTNGFC_WAIT_FOR_INTERRUPT  
IOCTL_CCRTNGFC_RELOAD_FIRMWARE  
IOCTL_CCRTNGFC_GET_ALL_BOARDS_DRIVER_INFO  
IOCTL_CCRTNGFC_LDIO_WAKEUP_COS_INTERRUPT  
IOCTL_CCRTNGFC_LDIO_WAIT_FOR_COS_INTERRUPT  
IOCTL_CCRTNGFC_WAIT_FOR_MSGDMA_INTERRUPT
```

IOCTL_CCRTNGFC_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_CCRTNGFC_REMOVE_IRQ* call earlier to remove the interrupt handler.

IOCTL_CCRTNGFC_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_CCRNGFC_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_CCRNGFC_GET_DRIVER_ERROR: The argument supplied to this *ioctl* is a pointer to the *ccrngfc_user_error_t* structure. Information on the structure is located in the *ccrngfc_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 *CCRNGFC_SUCCESS*.

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

IOCTL_CCRNGFC_GET_PHYSICAL_MEMORY: The argument supplied to this *ioctl* is a pointer to the *ccrngfc_user_phys_mem_t* structure. Information on the structure is located in the *ccrngfc_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_CCRNGFC_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_CCRNGFC_RESET_BOARD* call.

IOCTL_CCRNGFC_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_CCRNGFC_MMAP_SELECT: The argument to this *ioctl* is a pointer to the *ccrngfc_mmap_select_t* structure. Information on the structure is located in the *ccrngfc_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 (*CCRNGFC_SELECT_LOCAL_MMAP*), the configuration register space (*CCRNGFC_SELECT_CONFIG_MMAP*) the physical memory (*CCRNGFC_SELECT_PHYS_MEM_MMAP*) that is created by the *mmap(2)* system call and the driver/library mapping (*CCRNGFC_SELECT_DRIVER_LIBRARY_MMAP*).

IOCTL_CCRNGFC_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_CCRNGFC_PCI_CONFIG_REGISTERS: The argument supplied to this *ioctl* is a pointer to the *ccrngfc_pci_config_reg_addr_mapping_t* structure whose definition is located in the *ccrngfc_user.h* include file.

IOCTL_CCRNGFC_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_CCRNGFC_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_CCRNGFC_INIT_BOARD* call.

IOCTL_CCRNGFC_WAIT_FOR_INTERRUPT: The argument to this *ioctl* is a pointer to the *ccrngfc_driver_int_t* structure. Information on the structure is located in the *ccrngfc_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_CCRTNGFC_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_CCRTNGFC_GET_ALL_BOARDS_DRIVER_INFO: The argument to this *ioctl* is a pointer to *crrtngfc_all_boards_driver_info*. It provides the ability to supply all driver information for all the *crrtngfc* cards in the system to the user.

IOCTL_CCRTNGFC_LDIO_WAKEUP_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_CCRTNGFC_LDIO_WAIT_FOR_COS_INTERRUPT: This *ioctl* waits for a LIO/DIO change-of-state interrupt.

IOCTL_CCRTNGFC_LDIO_WAIT_FOR_COS_INTERRUPT: The argument to this *ioctl* is a pointer to the *crrtngfc_driver_ldio_cos_int_t* structure. Information on the structure is located in the *crrtngfc_user.h* include file. The user can wait for a LIO/DIO complete interrupt with the *WakeupInterruptMas* mask. This call blocks indefinitely until a LIO/DIO change-of-state interrupt occurs. If a change-of-state interrupt occurs, this call returns useful LIO/DIO related statistics to the user. To cancel a pending wait, users can use the *IOCTL_CCRTNGFC_LDIO_WAKEUP_COS_INTERRUPT ioctl()* call.

IOCTL_CCRTNGFC_WAIT_FOR_MSGDMA_INTERRUPT: This *ioctl* waits for a MsgDma interrupt. The argument to this *ioctl* is a pointer to the *crrtngfc_driver_int_t* structure. Information on the structure is located in the *crrtngfc_user.h* include file.

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_CCRTNGFC_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;
crrtngfc_local_ctrl_data_t *local_ptr;
crrtngfc_mmap_select_t      mmap_select;
unsigned long mmap_local_size;

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

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

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

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

2.2 Application Program Interface (API) Access

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

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

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

The following are a list of calls that are available.

2.2.1 `ccrtNGFC_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 ccrtNGFC_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 ccrtNGFC_Remove_Irq(), then this call adds it back.  
  
Input:   void *Handle           (Handle pointer)  
Output:  none  
Return:  _ccrtngfc_lib_error_number_t  
         # CCRTNGFC_LIB_NO_ERROR           (successful)  
         # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)  
         # CCRTNGFC_LIB_NOT_OPEN          (library not open)  
         # CCRTNGFC_LIB_IOCTL_FAILED      (driver ioctl call failed)  
*****/
```

2.2.2 `ccrtNGFC_BoardExpirationTimeRemaining()`

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

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtNGFC_BoardExpirationTimeRemaining(void *Handle,  
                                         time_t *SecondsToExpire,  
                                         ccrtngfc_date_string_t *GmtDateTimeString,  
                                         ccrtngfc_date_string_t *LocalDateTimeString,  
                                         _ccrtngfc_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)  
         ccrtngfc_date_string_t *GmtDateTimeString (GMT date/time  
                                                    string)  
         char date[CCRTNGFC_DATE_TIME_STRING_SIZE]  
         ccrtngfc_date_string_t *LocalDateTimeString (Local date/time  
                                                    string)
```

```

        char date[CCRTNGFC_DATE_TIME_STRING_SIZE]
        _ccrtngfc_firmware_state *FirmwareState      (Firmware State)
        # CCRTNGFC_FIRMWARE_STATE_UNRESTRICTED
        # CCRTNGFC_FIRMWARE_STATE_RESTRICTED
        # CCRTNGFC_FIRMWARE_STATE_EXPIRED
Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR                    (successful)
        # CCRTNGFC_LIB_BAD_HANDLE                  (no/bad handler supplied)
        # CCRTNGFC_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:

- CCRTNGFC_FIRMWARE_STATE_UNRESTRICTED. This firmware has no restrictions.
- CCRTNGFC_FIRMWARE_STATE_RESTRICTED. This firmware has restrictions. It is possible that and expiration date restriction is not present.
- CCRTNGFC_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.

2.2.3 ccrtNGFC_Clear_Driver_Error()

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

```

/*****
_ccrtngfc_lib_error_number_t ccrtNGFC_Clear_Driver_Error(void *Handle)

Description: Clear any previously generated driver related error.

Input:  void *Handle          (Handle pointer)
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_IOCTL_FAILED      (driver ioctl call failed)
*****/

```

2.2.4 ccrtNGFC_Clear_Interrupt_Status()

This call clears the interrupt status.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clear_Interrupt_Status (void          *Handle,
                                ccrtnngfc_interrupt_t *intr)

Description: Clear Interrupt Status

Input:  void          *Handle      (handle pointer)
        ccrtnngfc_interrupt_t *intr (pointer to interrupt status)
        _ccrtngfc_intsta_ldio_cos_t ldio_cos_module_int[CCRTNGFC_LDIO_MAX_MODULES];
        # CCRTNGFC_INT_LDIO_COS_NONE
        # CCRTNGFC_INT_LDIO_COS_RESET
        # CCRTNGFC_INT_LDIO_COS_DO_NOT_CHANGE

Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
        # CCRTNGFC_LIB_NO_LOCAL_REGION  (local region error)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
*****/

```

2.2.5 ccrtNGFC_Clear_Lib_Error()

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

```

/*****
_ccrtngfc_lib_error_number_t ccrtNGFC_Clear_Lib_Error(void *Handle)

Description: Clear any previously generated library related error.

Input:  void *Handle      (Handle pointer)
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
*****/

```

2.2.6 ccrtNGFC_Clock_Generator_Soft_Reset()

Perform a soft clock reset on all the output clocks.

```

/*****
_ccrtngfc_lib_error_number_t ccrtNGFC_Clock_Generator_Soft_Reset(void *Handle)

Description: Perform Soft Reset to Clock Generator

Input:  void *Handle      (Handle pointer)
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.7 ccrtNGFC_Clock_Get_Generator_CSR()

Return the clock generator control and status register.

```

/*****

```

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_CSR (void *Handle,
                                   ccrtnngfc_clkgen_csr_t *CgCsr)

```

Description: Get Generator Control and Status information

```

Input:  void *Handle (Handle pointer)
Output: ccrtnngfc_clkgen_csr_t *CgCsr (pointer to clock
                                         generator csr)

```

```

_ccrtngfc_clkgen_interface_t interface
# CCRTNGFC_CLOCK_GENERATOR_INTERFACE_IDLE
# CCRTNGFC_CLOCK_GENERATOR_INTERFACE_BUSY
_ccrtngfc_clkgen_output_t output
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_DISABLE
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_ENABLE
_ccrtngfc_clkgen_state_t state
# CCRTNGFC_CLOCK_GENERATOR_ACTIVE
# CCRTNGFC_CLOCK_GENERATOR_RESET

```

```

Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler
                             supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not
                                 present)

```

*****/

2.2.8 ccrtNGFC_Clock_Get_Generator_Info()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_Info (void *Handle,
                                   _ccrtngfc_clock_generator_output_t WhichOutput,
                                   ccrtnngfc_clock_generator_info_t *CgInfo)

```

Description: Get Clock Generator Information

```

Input:  void *Handle (Handle pointer)
        _ccrtngfc_clock_generator_output_t WhichOutput (select output)

```

```

# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_9

```

```

Output: ccrtnngfc_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
        _ccrtngfc_cg_zero_delay_t ZeroDelay

```

```

# CCRTNGFC_CG_ZERO_DELAY_MODE
# CCRTNGFC_CG_NORMAL_MODE
_ccrtngfc_cg_stat_ctrl_voltselect_t           Voltage_select
# CCRTNGFC_CG_VOLTAGE_SELECT_1_8V
# CCRTNGFC_CG_VOLTAGE_SELECT_3_3V
_ccrtngfc_cg_input_xaxb_extclk_sel_t         Input_xaxb_selection
# CCRTNGFC_CG_INPUT_XAXB_USE_CRYSTAL
# CCRTNGFC_CG_INPUT_XAXB_USE_EXTCLK_SOURCE
_ccrtngfc_cg_xaxb_power_down_t               Input_xaxb_power
# CCRTNGFC_CG_XAXB_POWER_DOWN
# CCRTNGFC_CG_XAXB_DO_NOT_POWER_DOWN
ccrtngfc_clkgen_csr_t                         Clkcsr
  _ccrtngfc_clkgen_interface_t               interface
    # CCRTNGFC_CLOCK_GENERATOR_INTERFACE_IDLE
    # CCRTNGFC_CLOCK_GENERATOR_INTERFACE_BUSY
  _ccrtngfc_clkgen_output_t                   output
    # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_DISABLE
    # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_ENABLE
  _ccrtngfc_clkgen_state_t                     state
    # CCRTNGFC_CLOCK_GENERATOR_ACTIVE
    # CCRTNGFC_CLOCK_GENERATOR_RESET
ccrtngfc_clkgen_output_config_t               Config
  _ccrtngfc_cg_outcfg_force_rdiv2_t           force_rdiv2
    # CCRTNGFC_CG_OUTPUT_CONFIG_DONT_FORCE_RDIV2
    # CCRTNGFC_CG_OUTPUT_CONFIG_FORCE_RDIV2
  _ccrtngfc_cg_outcfg_enable_t                 enable
    # CCRTNGFC_CG_OUTPUT_CONFIG_DISABLE
    # CCRTNGFC_CG_OUTPUT_CONFIG_ENABLE
  _ccrtngfc_cg_outcfg_shutdown_t              shutdown
    # CCRTNGFC_CG_OUTPUT_CONFIG_POWER_UP
    # CCRTNGFC_CG_OUTPUT_CONFIG_SHUTDOWN
ccrtngfc_clkgen_output_format_t               Format
  _ccrtngfc_cg_outfmt_cmos_drive_t            cmos_drive
    # CCRTNGFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_LVDS
    # CCRTNGFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_CMOS
  _ccrtngfc_cg_outfmt_disable_state_t         disable_state
    # CCRTNGFC_CG_OUTPUT_FORMAT_DISABLE_LOW
    # CCRTNGFC_CG_OUTPUT_FORMAT_DISABLE_HIGH
  _ccrtngfc_cg_outfmt_sync_t                   sync
    # CCRTNGFC_CG_OUTPUT_FORMAT_SYNC_DISABLE
    # CCRTNGFC_CG_OUTPUT_FORMAT_SYNC_ENABLE
  _ccrtngfc_cg_outfmt_format_t                 format
    # CCRTNGFC_CG_OUTPUT_FORMAT_FORMAT_LVDS
    # CCRTNGFC_CG_OUTPUT_FORMAT_FORMAT_CMOS
ccrtngfc_clkgen_output_mode_t                 Mode
  _ccrtngfc_cg_outmode_amplitude_t            amplitude
    # CCRTNGFC_CG_OUTPUT_AMPLITUDE_CMOS
    # CCRTNGFC_CG_OUTPUT_AMPLITUDE_LVDS
  _ccrtngfc_cg_outmode_common_t               common
    # CCRTNGFC_CG_OUTPUT_COMMON_CMOS
    # CCRTNGFC_CG_OUTPUT_COMMON_LVDS
    # CCRTNGFC_CG_OUTPUT_COMMON_LVPECL
ccrtngfc_clkgen_output_mux_t                  Mux
  _ccrtngfc_cg_outmux_inversion_t             inversion
    # CCRTNGFC_CG_OUTPUT_MUX_COMPLEMENTARY
    # CCRTNGFC_CG_OUTPUT_MUX_IN_PHASE
    # CCRTNGFC_CG_OUTPUT_MUX_INVERTED
    # CCRTNGFC_CG_OUTPUT_MUX_OUT_OF_PHASE
  _ccrtngfc_cg_outmux_ndiv_select_t           ndiv_mux
    # CCRTNGFC_CG_OUTPUT_MUX_NDIV_0
    # CCRTNGFC_CG_OUTPUT_MUX_NDIV_1

```



```

# CCRTNGFC_CG_OUTPUT_MUX_NDIV_2
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_3
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_4
ccrtngfc_clkgen_input_clock_enable_t      Input_clock_enable
  _ccrtngfc_cg_input_clock_enable_t      input_0_clock
# CCRTNGFC_CG_INPUT_CLOCK_DISABLE
# CCRTNGFC_CG_INPUT_CLOCK_ENABLE
  _ccrtngfc_cg_input_clock_enable_t      input_1_clock
# CCRTNGFC_CG_INPUT_CLOCK_DISABLE
# CCRTNGFC_CG_INPUT_CLOCK_ENABLE
  _ccrtngfc_cg_input_clock_enable_t      input_2_clock
# CCRTNGFC_CG_INPUT_CLOCK_DISABLE
# CCRTNGFC_CG_INPUT_CLOCK_ENABLE
  _ccrtngfc_cg_input_clock_enable_t      input_fb_clock
ccrtngfc_clkgen_input_clock_select_t      Input_clock_select
  _ccrtngfc_cg_input_clock_select_control_t control
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_PIN_CONTROL
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_REG_CONTROL
  _ccrtngfc_cg_input_clock_select_register_t select
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN0
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN1
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN2
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_INXAXB
ccrtngfc_pdiv_all_info_t                  Pdiv_info
  __u64                                    Pfb_divider
ccrtngfc_pdiv_info_t                      P0
  __u64                                    Divider
  _ccrtngfc_cg_pdiv_enable_t              Enable
# CCRTNGFC_CG_PDIV_DISABLE
# CCRTNGFC_CG_PDIV_ENABLE
  _ccrtngfc_cg_pdiv_input_state_t         State
# CCRTNGFC_CG_PDIV_INPUT_UNUSED
# CCRTNGFC_CG_PDIV_INPUT_DISABLED
# CCRTNGFC_CG_PDIV_INPUT_SELECTED
ccrtngfc_pdiv_info_t                      P1
  __u64                                    Divider
  _ccrtngfc_cg_pdiv_enable_t              Enable
# CCRTNGFC_CG_PDIV_DISABLE
# CCRTNGFC_CG_PDIV_ENABLE
  _ccrtngfc_cg_pdiv_input_state_t         State
# CCRTNGFC_CG_PDIV_INPUT_UNUSED
# CCRTNGFC_CG_PDIV_INPUT_DISABLED
# CCRTNGFC_CG_PDIV_INPUT_SELECTED
ccrtngfc_pdiv_info_t                      P2
  __u64                                    Divider
  _ccrtngfc_cg_pdiv_enable_t              Enable
# CCRTNGFC_CG_PDIV_DISABLE
# CCRTNGFC_CG_PDIV_ENABLE
  _ccrtngfc_cg_pdiv_input_state_t         State
# CCRTNGFC_CG_PDIV_INPUT_UNUSED
# CCRTNGFC_CG_PDIV_INPUT_DISABLED
# CCRTNGFC_CG_PDIV_INPUT_SELECTED
ccrtngfc_pdiv_info_t                      Pxaxb
  __u64                                    Divider
  _ccrtngfc_cg_pdiv_enable_t              Enable
# CCRTNGFC_CG_PDIV_DISABLE
# CCRTNGFC_CG_PDIV_ENABLE
  _ccrtngfc_cg_pdiv_input_state_t         State
# CCRTNGFC_CG_PDIV_INPUT_UNUSED

```

```

        # CCRTNGFC_CG_PDIV_INPUT_DISABLED
        # CCRTNGFC_CG_PDIV_INPUT_SELECTED
int          Which_Pdiv_Selected
int          P_Divider
long double  OutputClockFrequency;
# <valid positive output clock frequency>
# CCRTNGFC_CLOCK_ERROR_INVALID_P_DIVIDER
# CCRTNGFC_CLOCK_ERROR_VCO_CLOCK_NOT_IN_RANGE
# CCRTNGFC_CLOCK_ERROR_N_DIVIDER_NOT_IN_RANGE
# CCRTNGFC_CLOCK_ERROR_P_DIVIDER_NOT_IN_RANGE
# CCRTNGFC_CLOCK_ERROR_R_DIVIDER_NOT_IN_RANGE
# CCRTNGFC_CLOCK_ERROR_INVALID_CLOCK_FREQUENCY
Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION  (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.9 ccrtNGFC_Clock_Get_Generator_Input_Clock_Enable()

This call returns the status of all the input clocks.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_Input_Clock_Enable (void          *Handle,
                                                ccrtngfc_clkgen_input_clock_enable_t *InputClockEnable)

```

Description: Return the Clock Generator Input Clock Enable

```

Input:  void          *Handle          (Handle pointer)
Output: ccrtngfc_clkgen_input_clock_enable_t *InputClockEnable (pointer to
                                                                    input clock enable)

```

```

        _ccrtngfc_cg_input_clock_enable_t  input_0_clock
        # CCRTNGFC_CG_INPUT_CLOCK_DISABLE
        # CCRTNGFC_CG_INPUT_CLOCK_ENABLE
        _ccrtngfc_cg_input_clock_enable_t  input_1_clock
        # CCRTNGFC_CG_INPUT_CLOCK_DISABLE
        # CCRTNGFC_CG_INPUT_CLOCK_ENABLE
        _ccrtngfc_cg_input_clock_enable_t  input_2_clock
        # CCRTNGFC_CG_INPUT_CLOCK_DISABLE
        # CCRTNGFC_CG_INPUT_CLOCK_ENABLE
        _ccrtngfc_cg_input_clock_enable_t  input_fb_clock
        # CCRTNGFC_CG_INPUT_CLOCK_DISABLE
        # CCRTNGFC_CG_INPUT_CLOCK_ENABLE
Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION  (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.10 ccrtNGFC_Clock_Get_Generator_Input_Clock_Select()

This call returns the input clock selection.

```

/*****

```

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_Input_Clock_Select (void          *Handle,
                                                ccrtnngfc_clkgen_input_clock_select_t *ClkSel)

```

Description: Get Input Clock Selection

```

Input:  void          *Handle (Handle pointer)
Output: ccrtnngfc_clkgen_input_clock_select_t *ClkSel (pointer to
                                                input clock selection)

```

```

_ccrtngfc_cg_input_clock_select_control_t control;
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_PIN_CONTROL
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_REG_CONTROL
_ccrtngfc_cg_input_clock_select_register_t select;
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN0
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN1
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN2
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_INXAXB

```

```

Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR          (successful)
# CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN        (device not open)
# CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region error)
# CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)

```

*****/

2.2.11 ccrtNGFC_Clock_Get_Generator_Input_Clock_Status()

The call returns the input clock status.

/*****

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_Input_Clock_Status (void          *Handle,
                                                ccrtnngfc_clkgen_input_clock_status_t *ClkStatus)

```

Description: Get Input Clock Status

```

Input:  void          *Handle (Handle pointer)
Output: ccrtnngfc_clkgen_input_clock_status_t *ClkStatus (pointer to input
                                                         clock status)

```

```

_ccrtngfc_cg_calibration_status_t calstat
# CCRTNGFC_CG_STATUS_DEVICE_IS_NOT_CALIBRATING
# CCRTNGFC_CG_STATUS_DEVICE_IS_CALIBRATING
_ccrtngfc_cg_lol_pll_locked_t PLL_locked
# CCRTNGFC_CG_STATUS_LOL_PLL_LOCKED
# CCRTNGFC_CG_STATUS_LOL_PLL_NOT_LOCKED
_ccrtngfc_cg_smbus_timeout_error_t SMBUS_timeout
# CCRTNGFC_CG_STATUS_LOL_SMBUS_NOT_TIMEDOUT
# CCRTNGFC_CG_STATUS_LOL_SMBUS_TIMEDOUT
_ccrtngfc_cg_los_signal_present_t input_signal
# CCRTNGFC_CG_STATUS_LOS_SIGNAL_PRESENT
# CCRTNGFC_CG_STATUS_LOS_SIGNAL_NOT_PRESENT
_ccrtngfc_cg_los_alarm_t input_0_clock
# CCRTNGFC_CG_LOS_INPUT_CLOCK_PRESENT
# CCRTNGFC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
_ccrtngfc_cg_los_alarm_t input_1_clock
# CCRTNGFC_CG_LOS_INPUT_CLOCK_PRESENT
# CCRTNGFC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
_ccrtngfc_cg_los_alarm_t input_2_clock
# CCRTNGFC_CG_LOS_INPUT_CLOCK_PRESENT
# CCRTNGFC_CG_LOS_INPUT_CLOCK_NOT_PRESENT

```

```

        _ccrtngfc_cg_los_alarm_t                input_fb_clock
        # CCRTNGFC.CG_LOS_INPUT_CLOCK_PRESENT
        # CCRTNGFC.CG_LOS_INPUT_CLOCK_NOT_PRESENT
        _ccrtngfc_cg_losaxb_signal_present_t    input_xaxb_clock
        # CCRTNGFC.CG_LOS_INPUT_CLOCK_PRESENT
        # CCRTNGFC.CG_LOS_INPUT_CLOCK_NOT_PRESENT
Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC.LIB_NO_ERROR                (successful)
        # CCRTNGFC.LIB_BAD_HANDLE             (no/bad handler supplied)
        # CCRTNGFC.LIB_NOT_OPEN               (device not open)
        # CCRTNGFC.LIB_INVALID_ARG           (invalid argument)
        # CCRTNGFC.LIB_NO_LOCAL_REGION        (local region not present)
        # CCRTNGFC.LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
*****/

```

2.2.12 ccrtNGFC_Clock_Get_Generator_M_Divider()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_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: _ccrtngfc_lib_error_number_t
        # CCRTNGFC.LIB_NO_ERROR                (successful)
        # CCRTNGFC.LIB_BAD_HANDLE             (no/bad handler supplied)
        # CCRTNGFC.LIB_NOT_OPEN               (device not open)
        # CCRTNGFC.LIB_INVALID_ARG           (invalid argument)
        # CCRTNGFC.LIB_NO_LOCAL_REGION        (local region not present)
        # CCRTNGFC.LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
*****/

```

2.2.13 ccrtNGFC_Clock_Get_Generator_N_Divider()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_N_Divider (void      *Handle,
                                         _ccrtngfc_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)
        _ccrtngfc_clock_generator_divider_t WhichDivider (select divider)
        # CCRTNGFC.CLOCK_GENERATOR_DIVIDER_N0
        # CCRTNGFC.CLOCK_GENERATOR_DIVIDER_N1
        # CCRTNGFC.CLOCK_GENERATOR_DIVIDER_N2
        # CCRTNGFC.CLOCK_GENERATOR_DIVIDER_N3
        # CCRTNGFC.CLOCK_GENERATOR_DIVIDER_N4

```

```

Output:  __u64                *Numerator   (pointer to Numerator)
         __u32                *Denominator  (pointer to Denominator)
         long double          *Value         (pointer to Value)
Return:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR           (successful)
         # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
         # CCRTNGFC_LIB_NOT_OPEN          (device not open)
         # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
         # CCRTNGFC_LIB_NO_LOCAL_REGION   (local region not present)
         # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****

```

2.2.14 ccrtNGFC_Clock_Get_Generator_Output_Config()

Return the clock generator output configuration for the selected output.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_Output_Config (void          *Handle,
                                             _ccrtngfc_clock_generator_output_t WhichOutput,
                                             ccrtngfc_clkgen_output_config_t *OutCfg)

```

Description: Return Clock Generator Output Configuration

```

Input:  void                *Handle   (Handle pointer)
        _ccrtngfc_clock_generator_output_t WhichOutput (select output)
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_9
Output: ccrtngfc_clkgen_output_config_t *OutCfg (pointer to output config)
        _ccrtngfc_cg_outcfg_force_rdiv2_t force_rdiv2
        # CCRTNGFC_CG_OUTPUT_CONFIG_DONT_FORCE_RDIV2
        # CCRTNGFC_CG_OUTPUT_CONFIG_FORCE_RDIV2
        _ccrtngfc_cg_outcfg_enable_t enable
        # CCRTNGFC_CG_OUTPUT_CONFIG_DISABLE
        # CCRTNGFC_CG_OUTPUT_CONFIG_ENABLE
        _ccrtngfc_cg_outcfg_shutdown_t shutdown
        # CCRTNGFC_CG_OUTPUT_CONFIG_POWER_UP
        # CCRTNGFC_CG_OUTPUT_CONFIG_SHUTDOWN
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****

```

2.2.15 ccrtNGFC_Clock_Get_Generator_Output_Format()

Return the clock generator output format for the selected output.

```

/*****
_ccrtngfc_lib_error_number_t

```

```

ccrtNGFC_Clock_Get_Generator_Output_Format (void          *Handle,
                                             _ccrtngfc_clock_generator_output_t WhichOutput,
                                             ccrtngfc_clkgen_output_format_t  *OutFmt)

```

Description: Return Clock Generator Output Format

```

Input:  void          *Handle      (Handle pointer)
        _ccrtngfc_clock_generator_output_t WhichOutput (select output)
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_9
Output: ccrtngfc_clkgen_output_format_t  *OutFmt (pointer to output format)
        _ccrtngfc_cg_outfmt_cmos_drive_t      cmos_drive
        # CCRTNGFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_LVDS
        # CCRTNGFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_CMOS
        _ccrtngfc_cg_outfmt_disable_state_t    disable_state
        # CCRTNGFC_CG_OUTPUT_FORMAT_DISABLE_LOW
        # CCRTNGFC_CG_OUTPUT_FORMAT_DISABLE_HIGH
        _ccrtngfc_cg_outfmt_sync_t            sync
        # CCRTNGFC_CG_OUTPUT_FORMAT_SYNC_DISABLE
        # CCRTNGFC_CG_OUTPUT_FORMAT_SYNC_ENABLE
        _ccrtngfc_cg_outfmt_format_t          format
        # CCRTNGFC_CG_OUTPUT_FORMAT_FORMAT_LVDS
        # CCRTNGFC_CG_OUTPUT_FORMAT_FORMAT_CMOS
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR                (successful)
        # CCRTNGFC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN                (device not open)
        # CCRTNGFC_LIB_INVALID_ARG             (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION         (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
*****

```

2.2.16 ccrtNGFC_Clock_Get_Generator_Output_Mode()

Return the clock generator output mode for the selected output.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_Output_Mode (void          *Handle,
                                             _ccrtngfc_clock_generator_output_t WhichOutput,
                                             ccrtngfc_clkgen_output_mode_t  *OutMode)

```

Description: Return Clock Generator Output Mode

```

Input:  void          *Handle      (Handle pointer)
        _ccrtngfc_clock_generator_output_t WhichOutput (select output)
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6

```

```

# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_9
Output: ccrtnngfc_clkgen_output_mode_t      *OutMode (pointer to output
                                             amplitude/common mode)
        _ccrtnngfc_cg_outmode_amplitude_t  amplitude
        # CCRTNGFC_CG_OUTPUT_AMPLITUDE_CMOS
        # CCRTNGFC_CG_OUTPUT_AMPLITUDE_LVDS
        _ccrtnngfc_cg_outmode_common_t     common
        # CCRTNGFC_CG_OUTPUT_COMMON_CMOS
        # CCRTNGFC_CG_OUTPUT_COMMON_LVDS
        # CCRTNGFC_CG_OUTPUT_COMMON_LVPECL
Return: _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR             (successful)
        # CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN            (device not open)
        # CCRTNGFC_LIB_INVALID_ARG         (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION     (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****

```

2.2.17 ccrtnngfc_Clock_Get_Generator_Output_Mux()

Return the clock generator output mux for the selected output.

```

/*****
_ccrtnngfc_lib_error_number_t
ccrtnngfc_Clock_Get_Generator_Output_Mux (void          *Handle,
                                           _ccrtnngfc_clock_generator_output_t WhichOutput,
                                           ccrtnngfc_clkgen_output_mux_t      *OutMux)

```

Description: Return Clock Generator Output Mux

```

Input: void          *Handle      (Handle pointer)
        _ccrtnngfc_clock_generator_output_t WhichOutput (select output)
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_9
Output: ccrtnngfc_clkgen_output_mux_t      *OutMux (pointer to output
                                             inversion/N-divider mux)
        _ccrtnngfc_cg_outmux_inversion_t   inversion
        # CCRTNGFC_CG_OUTPUT_MUX_COMPLEMENTARY
        # CCRTNGFC_CG_OUTPUT_MUX_IN_PHASE
        # CCRTNGFC_CG_OUTPUT_MUX_INVERTED
        # CCRTNGFC_CG_OUTPUT_MUX_OUT_OF_PHASE
        _ccrtnngfc_cg_outmux_ndiv_select_t ndiv_mux
        # CCRTNGFC_CG_OUTPUT_MUX_NDIV_0
        # CCRTNGFC_CG_OUTPUT_MUX_NDIV_1
        # CCRTNGFC_CG_OUTPUT_MUX_NDIV_2
        # CCRTNGFC_CG_OUTPUT_MUX_NDIV_3
        # CCRTNGFC_CG_OUTPUT_MUX_NDIV_4
Return: _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR             (successful)
        # CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)

```

```

# CCRTNGFC_LIB_NOT_OPEN          (device not open)
# CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION   (local region not present)
# CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.18 ccrtNGFC_Clock_Get_Generator_P_Divider()

Return the clock generator P-Divider.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_P_Divider (void          *Handle,
                                         _ccrtngfc_clock_generator_divider_t WhichDivider,
                                         __u64          *Divider)

```

Description: Return Clock Generator P-Divider

```

Input:  void          *Handle      (Handle pointer)
        _ccrtngfc_clock_generator_divider_t WhichDivider (select divider)
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P0
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P1
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P2
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_PFB
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_PXAXB
Output: __u64          *Divider    (pointer to
                                   Divider)

```

```

Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)

```

```

*****/

```

2.2.19 ccrtNGFC_Clock_Get_Generator_P_Divider_Enable()

Return the clock generator P-Divider Enable state.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_P_Divider_Enable (void          *Handle,
                                                _ccrtngfc_clock_generator_divider_t WhichDivider,
                                                _ccrtngfc_cg_pdiv_enable_t      *Pdiv_Enable)

```

Description: Return Clock Generator P-Divider Enable

```

Input:  void          *Handle      (Handle pointer)
        _ccrtngfc_clock_generator_divider_t WhichDivider (select divider)
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P0
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P1
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P2
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_PXAXB
Output: _ccrtngfc_cg_pdiv_enable_t      *Pdiv_Enable (pointer to enable
                                                       flag)

```

```

        # CCRTNGFC_CG_PDIV_DISABLE
        # CCRTNGFC_CG_PDIV_ENABLE
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)

```



```

# CCRTNGFC_LIB_NOT_OPEN          (device not open)
# CCRTNGFC_LIB_INVALID_ARG      (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION  (local region not present)
# CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.20 ccrtNGFC_Clock_Get_Generator_R_Divider()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_R_Divider (void          *Handle,
                                         _ccrtngfc_clock_generator_divider_t WhichDivider,
                                         __u32          *Divider)

Description: Return Clock Generator R-Divider

Input:  void          *Handle      (Handle pointer)
        _ccrtngfc_clock_generator_divider_t WhichDivider (select divider)
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R0
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R1
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R2
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R3
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R4
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R5
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R6
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R7
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R8
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R9

Output: __u32          *Divider    (pointer to Divider)

Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.21 ccrtNGFC_Clock_Get_Generator_Revision()

Return the clock generator revision information.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_Revision (void          *Handle,
                                         ccrtngfc_clock_revision_t *Revision)

Description: Return Clock Generator Revision

Input:  void          *Handle      (Handle pointer)
Output: ccrtngfc_clock_revision_t *Revision (pointer to Divider)
        _ccrtngfc_cg_die_revision_t DieRevision
        # CCRTNGFC_CG_SILICON_REVISION_A0
        # CCRTNGFC_CG_SILICON_REVISION_A1
        _ccrtngfc_convert_base_part_number_t BasePartNumber;
        u_short BPN
        u_char NChar[2]
        _ccrtngfc_cg_clock_speed_grade_t ClockSpeedGrade;
        # CCRTNGFC_CG_CLOCK_SPEED_GRADE_A

```

```

        # CCRTNGFC_CG_CLOCK_SPEED_GRADE_B
        # CCRTNGFC_CG_CLOCK_SPEED_GRADE_C
        # CCRTNGFC_CG_CLOCK_SPEED_GRADE_D
        _ccrtngfc_cg_clock_revision_t      ClockRevision;
        # CCRTNGFC_CG_CLOCK_REVISION_A
        # CCRTNGFC_CG_CLOCK_REVISION_B
        # CCRTNGFC_CG_CLOCK_REVISION_C
        # CCRTNGFC_CG_CLOCK_REVISION_D
Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE         (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN           (device not open)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION     (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.22 ccrtNGFC_Clock_Get_Generator_Value()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_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: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE         (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN           (device not open)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION     (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.23 ccrtNGFC_Clock_Get_Generator_Voltage_Select()

Return the clock generator Voltage Selection.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_Voltage_Select (void          *Handle,
                                              _ccrtngfc_cg_stat_ctrl_voltsel_t *VoltSel)

```

Description: Return the Clock Generator Voltage Selection

```

Input:  void          *Handle      (Handle pointer)
Output: _ccrtngfc_cg_stat_ctrl_voltsel_t *VoltSel (pointer to voltage select)
        # CCRTNGFC_CG_VOLTAGE_SELECT_1_8V
        # CCRTNGFC_CG_VOLTAGE_SELECT_3_3V
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE         (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN           (device not open)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)

```

```

# CCRTNGFC_LIB_NO_LOCAL_REGION      (local region not present)
# CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE  (Clock is not active)
*****/

```

2.2.24 ccrtNGFC_Clock_Get_Generator_Zero_Delay()

Return the clock generator Zero Delay status.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Get_Generator_Zero_Delay (void          *Handle,
                                          _ccrtngfc_cg_zero_delay_t *ZeroDelay)

```

Description: Return the Clock Generator Zero Delay setting.

```

Input:  void          *Handle      (Handle pointer)
Output: _ccrtngfc_cg_zero_delay_t *ZeroDelay (pointer to zero delay)
        # CCRTNGFC.CG_ZERO_DELAY_MODE
        # CCRTNGFC.CG_NORMAL_MODE
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN     (device not open)
        # CCRTNGFC_LIB_INVALID_ARG  (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.25 ccrtNGFC_Clock_PLL_CSR()

This call is provided to select either the clock generator or the clock oscillator as the source for the FPGA. By default, the FPGA firmware loads by using the clock oscillator and switches to the clock generator the moment any clocks are programmed by the user. When any clock is programmed, the software programs a reserved clock 5 (the generator source) to 100MHz. Once successfully programmed, the firmware automatically switches to this clock for all card access.

If for whatever reason, the customer wishes to use the clock oscillator instead of the clock generator, then they can use this call to accomplish it.

In addition to this clock source selection, this API also supplies useful information on the current status of the clock sources.

Note!!! This PLL Synchronization feature is not supported by the initial firmware release, it is supported by future releases.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_PLL_CSR (void          *Handle,
                        _ccrtngfc_pll_clock_source_t PLL_ClockSourceSelect,
                        ccrtngfc_clock_pll_status_t *PLL_ClockStatus)

```

Description: Clock PLL Control and Status

```

Input:  void          *Handle      (Handle pointer)
        _ccrtngfc_pll_clock_source_t PLL_ClockSourceSelect
        # CCRTNGFC.PLL_CLOCK_GENERATOR_SOURCE
        # CCRTNGFC.PLL_CLOCK_OSCILLATOR_SOURCE
        # CCRTNGFC.PLL_CLOCK_STATE_DO_NOT_CHANGE

```

```

Output:  ccrtngfc_clock_pll_status_t  *PLL_ClockStatus
         ccrtngfc_bool_t              PLL_firmware_support
         # CCRTNGFC_TRUE
         # CCRTNGFC_FALSE
         _ccrtngfc_pll_clock_source_t  clock_select
         # CCRTNGFC_PLL_CLOCK_GENERATOR_SOURCE
         # CCRTNGFC_PLL_CLOCK_OSCILLATOR_SOURCE
         _ccrtngfc_pll_unlock_error_t  PLL_unlock_error
         # CCRTNGFC_PLL_NEVER_UNLOCKED
         # CCRTNGFC_PLL_UNLOCKED_AT_SOME_POINT
         _ccrtngfc_pll_locked_status_t  PLL_locked_status
         # CCRTNGFC_PLL_NOT_LOCKED_ON_FREQUENCY
         # CCRTNGFC_PLL_LOCKED_ON_FREQUENCY
         _ccrtngfc_pll_clock_status_t  clock_status_0
         # CCRTNGFC_PLL_CLOCK_BAD_OR_MISSING
         # CCRTNGFC_PLL_CLOCK_PRESENT
         _ccrtngfc_pll_clock_status_t  clock_status_1
         # CCRTNGFC_PLL_CLOCK_BAD_OR_MISSING
         # CCRTNGFC_PLL_CLOCK_PRESENT
         _ccrtngfc_pll_active_clock_t  active_clock
         # CCRTNGFC_PLL_CLOCK_GENERATOR_ACTIVE
         # CCRTNGFC_PLL_CLOCK_OSCILLATOR_ACTIVE
         NULL
Return:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR              (successful)
         # CCRTNGFC_LIB_BAD_HANDLE            (no/bad handler supplied)
         # CCRTNGFC_LIB_NOT_OPEN              (device not open)
         # CCRTNGFC_LIB_INVALID_ARG           (invalid argument)
         # CCRTNGFC_LIB_NO_LOCAL_REGION        (local region not present)
         # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
         # CCRTNGFC_LIB_PLL_SYNC_NOT_SUPPORTED (PLL Sync not supported)
*****/

```

2.2.26 ccrtNGFC_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
ccrtNGFC_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
*****/

```

2.2.27 ccrtNGFC_Clock_Set_Generator_CSR()

This call sets the clock generator control and status register.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_CSR (void          *Handle,
                                   ccrtnngfc_clkgen_csr_t *CgCsr)

Description: Set Clock Generator Control and Status information

Input:  void          *Handle      (Handle pointer)
        ccrtnngfc_clkgen_csr_t *CgCsr (pointer to clock generator csr)
        _ccrtngfc_clkgen_output_t      output
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_DISABLE
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_ENABLE
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_DO_NOT_CHANGE
        _ccrtngfc_clkgen_state_t      state
        # CCRTNGFC_CLOCK_GENERATOR_ACTIVE
        # CCRTNGFC_CLOCK_GENERATOR_RESET
        # CCRTNGFC_CLOCK_GENERATOR_STATE_DO_NOT_CHANGE

Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE   (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN     (device not open)
        # CCRTNGFC_LIB_INVALID_ARG  (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
*****/

```

2.2.28 ccrtNGFC_Clock_Set_Generator_Input_Clock_Enable()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_Input_Clock_Enable (void          *Handle,
                                                  ccrtnngfc_clkgen_input_clock_enable_t *InputClockEnable)

Description: Set Clock Generator Input Clock Enable

Input:  void          *Handle      (Handle pointer)
        ccrtnngfc_clkgen_input_clock_enable_t *InputClockEnable (pointer to input clock enable)
        _ccrtngfc_cg_input_clock_enable_t      input_0_clock
        # CCRTNGFC_CG_INPUT_CLOCK_DISABLE
        # CCRTNGFC_CG_INPUT_CLOCK_ENABLE
        # CCRTNGFC_CG_INPUT_CLOCK_DO_NOT_CHANGE
        _ccrtngfc_cg_input_clock_enable_t      input_1_clock
        # CCRTNGFC_CG_INPUT_CLOCK_DISABLE
        # CCRTNGFC_CG_INPUT_CLOCK_ENABLE
        # CCRTNGFC_CG_INPUT_CLOCK_DO_NOT_CHANGE
        _ccrtngfc_cg_input_clock_enable_t      input_2_clock
        # CCRTNGFC_CG_INPUT_CLOCK_DISABLE
        # CCRTNGFC_CG_INPUT_CLOCK_ENABLE
        # CCRTNGFC_CG_INPUT_CLOCK_DO_NOT_CHANGE
        _ccrtngfc_cg_input_clock_enable_t      input_fb_clock
        # CCRTNGFC_CG_INPUT_CLOCK_DISABLE
        # CCRTNGFC_CG_INPUT_CLOCK_ENABLE
        # CCRTNGFC_CG_INPUT_CLOCK_DO_NOT_CHANGE

Output: none
Return: _ccrtngfc_lib_error_number_t

```

```

# CCRTNGFC_LIB_NO_ERROR                (successful)
# CCRTNGFC_LIB_BAD_HANDLE              (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN                (device not open)
# CCRTNGFC_LIB_INVALID_ARG            (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION        (local region not present)
# CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
*****/

```

2.2.29 ccrtNGFC_Clock_Set_Generator_Input_Clock_Select()

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

```

/*****

```

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_Input_Clock_Select (void      *Handle,
                                                ccrtngfc_clkgen_input_clock_select_t *ClkSel)

```

Description: Set Clock Generator Input Clock Selection

Input: void *Handle (Handle pointer)
ccrtngfc_clkgen_input_clock_select_t *ClkSel (pointer to input clock select)

```

_ccrtngfc_cg_input_clock_select_control_t control;
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_PIN_CONTROL
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_REG_CONTROL
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_CONTROL_DO_NOT_CHANGE
_ccrtngfc_cg_input_clock_select_register_t select;
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN0
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN1
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN2
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_INXAXB
# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN_DO_NOT_CHANGE

```

Output: none

```

Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR                (successful)
# CCRTNGFC_LIB_BAD_HANDLE              (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN                (device not open)
# CCRTNGFC_LIB_INVALID_ARG            (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION        (local region not present)
# CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE    (Clock is not active)
*****/

```

2.2.30 ccrtNGFC_Clock_Set_Generator_M_Divider()

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

```

/*****

```

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_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:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR      (successful)
         # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
         # CCRTNGFC_LIB_NOT_OPEN      (device not open)
         # CCRTNGFC_LIB_INVALID_ARG   (invalid argument)
         # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
         # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.31 ccrtNGFC_Clock_Set_Generator_N_Divider()

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

```

/*****
ccrtNGFC_Clock_Set_Generator_N_Divider()
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_N_Divider (void          *Handle,
    _ccrtngfc_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)
        _ccrtngfc_clock_generator_divider_t  WhichDivider  (select divider)
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_N0
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_N1
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_N2
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_N3
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_N4
        __u64          Numerator    (Numerator)
        __u32          Denominator  (Denominator)
        int            Update       (True=Update)
Output: none
Return:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR      (successful)
         # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
         # CCRTNGFC_LIB_NOT_OPEN      (device not open)
         # CCRTNGFC_LIB_INVALID_ARG   (invalid argument)
         # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
         # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.32 ccrtNGFC_Clock_Set_Generator_Output_Config()

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

```

/*****
    _ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_Output_Config (void          *Handle,
    _ccrtngfc_clock_generator_output_t  WhichOutput,

```

ccrtngfc_clkgen_output_config_t *OutCfg)

Description: Set Clock Generator Output Configuration

```

Input:  void *Handle (Handle pointer)
        _ccrtngfc_clock_generator_output_t WhichOutput (select output)
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_9
        ccrtngfc_clkgen_output_config_t *OutCfg (pointer to output config)
        _ccrtngfc_cg_outcfg_force_rdiv2_t force_rdiv2
        # CCRTNGFC_CG_OUTPUT_CONFIG_DONT_FORCE_RDIV2
        # CCRTNGFC_CG_OUTPUT_CONFIG_FORCE_RDIV2
        # CCRTNGFC_CG_OUTPUT_CONFIG_FORCE_DO_NOT_CHANGE
        _ccrtngfc_cg_outcfg_enable_t enable
        # CCRTNGFC_CG_OUTPUT_CONFIG_DISABLE
        # CCRTNGFC_CG_OUTPUT_CONFIG_ENABLE
        # CCRTNGFC_CG_OUTPUT_CONFIG_ENABLE_DO_NOT_CHANGE
        _ccrtngfc_cg_outcfg_shutdown_t shutdown
        # CCRTNGFC_CG_OUTPUT_CONFIG_POWER_UP
        # CCRTNGFC_CG_OUTPUT_CONFIG_SHUTDOWN
        # CCRTNGFC_CG_OUTPUT_CONFIG_SHUTDOWN_DO_NOT_CHANGE

Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR (successful)
        # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN (device not open)
        # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****

```

2.2.33 ccrtNGFC_Clock_Set_Generator_Output_Format()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_Output_Format (void *Handle,
        _ccrtngfc_clock_generator_output_t WhichOutput,
        ccrtngfc_clkgen_output_format_t *OutFmt)

```

Description: Set Clock Generator Output Format

```

Input:  void *Handle (Handle pointer)
        _ccrtngfc_clock_generator_output_t WhichOutput (select output)
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5

```



```

# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_9
ccrtngfc_clkgen_output_format_t      *OutFmt      (pointer to
                                          output format)

_ccrtngfc_cg_outfmt_cmos_drive_t      cmos_drive
# CCRTNGFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_LVDS
# CCRTNGFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_CMOS
# CCRTNGFC_CG_OUTPUT_FORMAT_CMOS_DRIVE_DO_NOT_CHANGE
_ccrtngfc_cg_outfmt_disable_state_t    disable_state
# CCRTNGFC_CG_OUTPUT_FORMAT_DISABLE_LOW
# CCRTNGFC_CG_OUTPUT_FORMAT_DISABLE_HIGH
# CCRTNGFC_CG_OUTPUT_FORMAT_DISABLE_DO_NOT_CHANGE
_ccrtngfc_cg_outfmt_sync_t             sync
# CCRTNGFC_CG_OUTPUT_FORMAT_SYNC_DISABLE
# CCRTNGFC_CG_OUTPUT_FORMAT_SYNC_ENABLE
# CCRTNGFC_CG_OUTPUT_FORMAT_SYNC_DO_NOT_CHANGE
_ccrtngfc_cg_outfmt_format_t           format
# CCRTNGFC_CG_OUTPUT_FORMAT_FORMAT_LVDS
# CCRTNGFC_CG_OUTPUT_FORMAT_FORMAT_CMOS
# CCRTNGFC_CG_OUTPUT_FORMAT_FORMAT_DO_NOT_CHANGE
Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR                 (successful)
# CCRTNGFC_LIB_BAD_HANDLE               (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN                 (device not open)
# CCRTNGFC_LIB_INVALID_ARG             (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION         (local region not present)
# CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
*****/

```

2.2.34 ccrtNGFC_Clock_Set_Generator_Output_Mode()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_Output_Mode (void      *Handle,
                                          _ccrtngfc_clock_generator_output_t WhichOutput,
                                          ccrtngfc_clkgen_output_mode_t *OutMode)

```

Description: Set Clock Generator Output Mode

```

Input: void      *Handle      (Handle pointer)
       _ccrtngfc_clock_generator_output_t WhichOutput (select output)
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_9
ccrtngfc_clkgen_output_mode_t      *OutMode      (pointer to
                                          output mode)
_ccrtngfc_cg_outmode_amplitude_t    amplitude

```

```

# CCRTNGFC_CG_OUTPUT_AMPLITUDE_CMOS
# CCRTNGFC_CG_OUTPUT_AMPLITUDE_LVDS
# CCRTNGFC_CG_OUTPUT_AMPLITUDE_DO_NOT_CHANGE
_ccrtngfc_cg_outmode_common_t          common
# CCRTNGFC_CG_OUTPUT_COMMON_CMOS
# CCRTNGFC_CG_OUTPUT_COMMON_LVDS
# CCRTNGFC_CG_OUTPUT_COMMON_LVPECL
# CCRTNGFC_CG_OUTPUT_COMMON_DO_NOT_CHANGE

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR                (successful)
# CCRTNGFC_LIB_BAD_HANDLE               (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN                 (device not open)
# CCRTNGFC_LIB_INVALID_ARG              (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION           (local region not present)
# CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE      (Clock is not active)
*****/

```

2.2.35 ccrtNGFC_Clock_Set_Generator_Output_Mux()

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

/******

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_Output_Mux (void          *Handle,
                                          _ccrtngfc_clock_generator_output_t WhichOutput,
                                          ccrtngfc_clkgen_output_mux_t *OutMux)

```

Description: Set Clock Generator Output Mux

```

Input: void          *Handle (Handle pointer)
       _ccrtngfc_clock_generator_output_t WhichOutput (select output)
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8
# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_9
ccrtngfc_clkgen_output_mux_t *OutMux (pointer to output
                                      inversion/N-divider mux)

_ccrtngfc_cg_outmux_inversion_t inversion
# CCRTNGFC_CG_OUTPUT_MUX_COMPLEMENTARY
# CCRTNGFC_CG_OUTPUT_MUX_IN_PHASE
# CCRTNGFC_CG_OUTPUT_MUX_INVERTED
# CCRTNGFC_CG_OUTPUT_MUX_OUT_OF_PHASE
# CCRTNGFC_CG_OUTPUT_MUX_INVERSION_DO_NOT_CHANGE
_ccrtngfc_cg_outmux_ndiv_select_t ndiv_mux
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_0
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_1
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_2
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_3
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_4
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_DO_NOT_CHANGE

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR                (successful)

```

```

# CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN            (device not open)
# CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION    (local region not present)
# CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.36 ccrtNGFC_Clock_Set_Generator_P_Divider()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_P_Divider (void          *Handle,
                                         _ccrtngfc_clock_generator_divider_t WhichDivider,
                                         __u64          Divider,
                                         int            Update)

Description: Set Clock Generator R-Divider
Input:   void          *Handle      (Handle pointer)
         _ccrtngfc_clock_generator_divider_t WhichDivider (select divider)
         # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P0
         # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P1
         # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P2
         # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_PFB
         # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_PXAXB
         __u64          Divider      (Divider)
         int            Update       (True=Update)
Output:  none
Return:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR      (successful)
         # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
         # CCRTNGFC_LIB_NOT_OPEN      (device not open)
         # CCRTNGFC_LIB_INVALID_ARG   (invalid argument)
         # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
         # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.37 ccrtNGFC_Clock_Set_Generator_P_Divider_Enable()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_P_Divider_Enable (void          *Handle,
                                                _ccrtngfc_clock_generator_divider_t WhichDivider,
                                                _ccrtngfc_cg_pdiv_enable_t      Pdiv_Enable)

Description: Set Clock Generator P-Divider Enable

Input:   void          *Handle      (Handle pointer)
         _ccrtngfc_clock_generator_divider_t WhichDivider (select divider)
         # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P0
         # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P1
         # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P2
         # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_PXAXB

```

```

        _ccrtngfc_cg_pdiv_enable_t          Pdiv_Enable      (enable flag)
        # CCRTNGFC_CG_PDIV_DISABLE
        # CCRTNGFC_CG_PDIV_ENABLE
Output:  none
Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR            (successful)
        # CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN            (device not open)
        # CCRTNGFC_LIB_INVALID_ARG         (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION     (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.38 ccrtNGFC_Clock_Set_Generator_R_Divider()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_R_Divider (void          *Handle,
                                         _ccrtngfc_clock_generator_divider_t WhichDivider,
                                         __u32          Divider)

```

Description: Set Clock Generator R-Divider

```

Input:  void          *Handle      (Handle pointer)
        _ccrtngfc_clock_generator_divider_t WhichDivider (select divider)
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R0
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R1
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R2
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R3
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R4
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R5
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R6
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R7
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R8
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_R9
        __u32          Divider      (Divider)
Output:  none
Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR            (successful)
        # CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN            (device not open)
        # CCRTNGFC_LIB_INVALID_ARG         (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION     (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.39 ccrtNGFC_Clock_Set_Generator_Value()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_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: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN      (device not open)
        # CCRTNGFC_LIB_INVALID_ARG   (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
```

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

2.2.40 ccrtNGFC_Clock_Set_Generator_Voltage_Select()

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

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

```
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_Voltage_Select (void          *Handle,
                                              _ccrtngfc_cg_stat_ctrl_voltSel_t VoltSel)
```

Description: Set Clock Generator voltage selection

```
Input:  void          *Handle      (Handle pointer)
        _ccrtngfc_cg_stat_ctrl_voltSel_t VoltSel      (voltage selection)
        # CCRTNGFC_CG_VOLTAGE_SELECT_1_8V
        # CCRTNGFC_CG_VOLTAGE_SELECT_3_3V
```

Output: none

```
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN      (device not open)
        # CCRTNGFC_LIB_INVALID_ARG   (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
```

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

2.2.41 ccrtNGFC_Clock_Set_Generator_Zero_Delay()

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

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

```
_ccrtngfc_lib_error_number_t
ccrtNGFC_Clock_Set_Generator_Zero_Delay (void          *Handle,
                                          _ccrtngfc_cg_zero_delay_t ZeroDelay)
```

Description: Set Clock Generator Zero Delay selection

```
Input:  void          *Handle      (Handle pointer)
        _ccrtngfc_cg_zero_delay_t ZeroDelay      (zero delay selection)
```

```

        # CCRTNGFC_CG_ZERO_DELAY_MODE
        # CCRTNGFC_CG_NORMAL_MODE
Output:  none
Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.42 ccrtNGFC_Close()

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

```

/*****
_ccrtngfc_lib_error_number_t ccrtNGFC_Close(void *Handle)
Description: Close a previously opened device.

Input:  void *Handle           (Handle pointer)
Output:  none
Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
*****/

```

2.2.43 ccrtNGFC_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.

```

/*****

ccrtNGFC_Compute_All_Output_Clocks()

Description: Compute All Output Clocks

Input:  void *Handle           (Handle pointer)
        double InputClockFrequency (Input clock frequency)
        ccrtngfc_compute_all_output_clocks_t *AllClocks (Pointer to all output clocks info)
        ccrtngfc_compute_single_output_clock_t *Clock
        long double DesiredFrequency
        double DesiredTolerancePPT
Output:  ccrtngfc_compute_all_output_clocks_t *AllClocks
        (Pointer to returned output clocks info)
        __u32 NumberOfNdividers
        ccrtngfc_compute_single_output_clock_t *Clock
        _ccrtngfc_clock_generator_output_t OutputClock
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7

```

```

        # CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTNGFC_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
__ccrtngfc_cg_outmux_ndiv_select_t  Ndiv_ToUse
        # CCRTNGFC_CG_OUTPUT_MUX_NDIV_0
        # CCRTNGFC_CG_OUTPUT_MUX_NDIV_1
        # CCRTNGFC_CG_OUTPUT_MUX_NDIV_2
        # CCRTNGFC_CG_OUTPUT_MUX_NDIV_3
        # CCRTNGFC_CG_OUTPUT_MUX_NDIV_4
__u32      Rdiv_value
__u32      Rdivider
__u32      Pdivider
Return:    _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler
                                         supplied)
        # CCRTNGFC_LIB_NOT_OPEN         (library not open)
        # CCRTNGFC_LIB_NO_LOCAL_REGION  (local region error)
        # CCRTNGFC_LIB_IO_ERROR         (device not ready)
        # CCRTNGFC_LIB_N_DIVIDERS_EXCEEDED (number of N-Dividers
                                         exceeded)
        # CCRTNGFC_LIB_CANNOT_COMPUTE_OUTPUT_FREQ (cannot compute
                                         output freq)
        # CCRTNGFC_LIB_INVALID_ARG      (invalid argument)
*****/

```

2.2.44 ccrtNGFC_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.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_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: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN         (library not open)
        # CCRTNGFC_LIB_INVALID_ARG      (invalid argument)
        # CCRTNGFC_LIB_AVALON_TRANSLATION_TABLE (avalon translation table
                                                error)
        # CCRTNGFC_LIB_ADDRESS_RANGE_ERROR (address range error)

```

*****/

2.2.45 ccrtNGFC_Create_UserLDioCosInterruptHandler()

This call provides the ability for a user to get notification when a LIO/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 LDIO 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 LIO/DIO COS interrupts. When a LIO/DIO COS interrupt occurs, the driver will wake up this thread which in turn execute the user supplied *interrupt callback* function. Various LIO/DIO COS statistics will be returned to the user as an argument *driver_ldio_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 *ccrtNGFC_Destroy_UserDioCosInterruptHandler()* call prior to creating a new one.

```
/*
_ccrtngfc_lib_error_number_t
ccrtNGFC_Create_UserLDioCosInterruptHandler(void      *Handle,
                                             void      Callback(),
                                             u_int     LdioCosWakeupInterruptMask
                                             int      SchedulePolicy,
                                             int      SchedulePriority)
```

Description: Create a User DIO COS Interrupt Handler

```
Input:  void      *Handle      (Handle pointer)
        void      Callback()   (user callback function)
        u_int     LDioCosWakeupInterruptMask (wakeup interrupt mask)
        # CCRTNGFC_LDIO_MODULE_0_INTMASK
        # CCRTNGFC_LDIO_MODULE_1_INTMASK
        # CCRTNGFC_LDIO_MODULE_2_INTMASK
        # CCRTNGFC_LDIO_MODULE_3_INTMASK
        int      SchedulePolicy
        # SCHED_FIFO
        # SCHED_RR
        # SCHED_OTHER
        int      SchedulePriority
        # 1..99 for SCHED_FIFO or SCHED_RR
        # 0     for SCHED_OTHER
```

```
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN      (device not open)
        # CCRTNGFC_LIB_INTHDLR_CREATE_FAILURE (failed to create
                                                interrupt handler)
        # CCRTNGFC_LIB_INTHDLR_ALREADY_RUNNING (interrupt hdlr already
                                                running)
        # CCRTNGFC_LIB_IOCTL_FAILED   (ioctl failed)
        # CCRTNGFC_LIB_INVALID_ARG    (invalid argument)
```

*****/

```
// User interrupt callback()
```

```
void LDioCosUserCallback(void      *Handle,
                           ccrtngfc_driver_ldio_cos_int_t *driver_ldio_cos_int)
```



```

{
    // User supplied code for handling interrupt
}

// Interrupt Counters
typedef struct
{
    long long unsigned   LDIO_COS_WakeupUserCounter;
    long long unsigned   LDIO_COS_ChannelsCount[CCRTNGFC_LDIO_MAX_MODULES];
    long long unsigned   LDIO_COS_ChannelsOverflowCount[CCRTNGFC_LDIO_MAX_MODULES];
    int RESERVED[128];
} ccrtnngfc_interrupt_ldio_cos_counters_t;

typedef u_int32_t   ccrtnngfc_ldio_modules_t[CCRTNGFC_LDIO_MAX_MODULES];

// LDIO COS Interrupt
typedef struct
{
    uint           InterruptsOccurredMask;
    uint           WakeupInterruptMask;

    // LDIO Current information
    ccrtnngfc_ldio_modules_t LDIO_COS_ChannelsStatus;
    ccrtnngfc_ldio_modules_t LDIO_COS_ChannelsOverflow;

    // LDIO Queued 1-deep information
    ccrtnngfc_ldio_modules_t LDIO_COS_Queued1ChannelsStatus;

    // LDIO Queued 2-deep information
    ccrtnngfc_ldio_modules_t LDIO_COS_Queued2ChannelsStatus;

    ccrtnngfc_interrupt_ldio_cos_counters_t   counters;
    int RESERVED[220];
} ccrtnngfc_driver_ldio_cos_int_t;

```

2.2.46 ccrtnngfc_Create_UserProcess()

Typically reads from the card 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. (*This is an experimental API for debugging and testing*).

```

/*****
_ccrtnngfc_lib_error_number_t
ccrtnngfc_Create_UserProcess(void           *Handle,
                                _ccrtnngfc_UserFunction_t *UFunc,
                                _ccrtnngfc_UserFunction_t **UFuncHandle)

```

Description: Create a User Process for user defined processing

Input:	void	*Handle	(Handle pointer)
	_ccrtnngfc_UserFunction_t	*UFunc	(pointer to user information structure)
Output:	_ccrtnngfc_UserFunction_t	**UFuncHandle	(pointer to user function struct handle)
Return:	_ccrtnngfc_lib_error_number_t	# CCRTNGFC_LIB_NO_ERROR	(successful)

```

# CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN           (device not open)
# CCRTNGFC_LIB_NO_RESOURCE        (cannot allocate memory)
# CCRTNGFC_LIB_INTERNAL_ERROR     (pthread attr failed)
# CCRTNGFC_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 ccrtnngfc_uf_action_t Action;
    volatile ccrtnngfc_uf_state_t State;
    volatile int CommandPending;
    volatile void *Next_UserFunction;
    volatile unsigned int long long RunCount;
    volatile int Pause;
} _ccrtnngfc_UserFunction_t;

```

2.2.47 ccrtnngfc_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 *CCRTNGFC_TWOS_COMPLEMENT* or *CCRTNGFC_OFFSET_BINARY*. The data supplied in *us_data* must not be greater than the hardware resolution bits *CCRTNGFC_ADC_RESOLUTION_BITS* supported by the board. Data greater than this will be masked out.

```

/*****
double ccrtnngfc_DataToVolts(int us_data, ccrtnngfc_volt_convert_t *conv)

Description: Convert Data to volts

Input:  int          us_data          (data to convert)
        ccrtnngfc_volt_convert_t *conv (pointer to
                                         conversion struct)
        double       VoltageRange     (maximum voltage
                                         range)
        _ccrtnngfc_csr_dataformat_t Format (format)
        # CCRTNGFC_OFFSET_BINARY
        # CCRTNGFC_TWOS_COMPLEMENT
        ccrtnngfc_bool BiPolar       (bi-polar)

```

```

        # CCRTNGFC_TRUE
        # CCRTNGFC_FALSE
    int ResolutionBits (Number of
                        resolution bits)
    Output: none
    Return: double volts (returned volts)
    *****/

```

2.2.48 ccrtNGFC_DataToVoltsLxxFormat()

This routine takes the various raw data maintained by the power module and converts to a floating point format for user display. The *LxxFormat* value to be selected is specific to the data being collected.

```

/*****
double
ccrtNGFC_DataToVoltsLxxFormat (int us_data,
                               _ccrtngfc_floating_point_format_t LxxFormat)

Description: Convert Data to volts in L11 or L16 format

Input:  uint us_data (data to convert)
        _ccrtngfc_floating_point_format_t LxxFormat (L11 or L16 format)
        # CCRTNGFC_FLOATING_POINT_FORMAT_L11
        # CCRTNGFC_FLOATING_POINT_FORMAT_L16

Output: none
Return: double volts (returned volts)
*****/

```

2.2.49 ccrtNGFC_DaughterCard_EEPROM_Read()

The purpose of this call is to read the EEPROM contents of the daughter card installed on the board. The user needs to select the daughter card, the address within the EEPROM and the size to read. Normally, this call is not to be used by the user as the EEPROM data has been setup by Concurrent Real-Time.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_DaughterCard_EEPROM_Read(void *Handle,
                                   ccrtnngfc_dc_eeprom_io_t *eeprom_read)

Description: Daughter Card EEPROM Read

Input:  void *Handle (handle pointer)
        ccrtnngfc_dc_eeprom_io_t *eeprom_read
        - _ccrtngfc_dc_eeprom_daughter_card_select_t DaughterCardSelect
          # CCRTNGFC_DCE_DAUGHTER_CARD_0
          # CCRTNGFC_DCE_DAUGHTER_CARD_1
        - uint Address
        - uint Size
        - uint FirmwareId

Output: ccrtnngfc_dc_eeprom_io_t *eeprom_read
        - char *DataPtr

Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR (successful)
        # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN (library not open)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTNGFC_LIB_CANNOT_OPEN_FILE (cannot open calib. file)
        # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
        # CCRTNGFC_LIB_DC_EEPROM_FAILURE (Daughter Card EEPROM Failure)
        # CCRTNGFC_LIB_DC_EEPROM_BUSY (Daughter Card EEPROM Busy)

```

```

# CCRTNGFC_LIB_NO_DAUGHTER_CARD_PRESENT
(Daughter Card Not Present)
*****/

```

2.2.50 ccrtNGFC_DaughterCard_EEPROM_Write()

The purpose of this call is to write to the EEPROM of the daughter card installed on the board. The user needs to select the daughter card, the address within the EEPROM, the size to read and the data to be written. Normally, this call is not to be used by the user as the EEPROM data has been setup by Concurrent Real-Time. If its contents is changed, it is possible that the driver may no longer be able to access the card.

Normally, the routine does a read of the data at the EEPROM offset and only writes to the the EEPROM if the value to be written is different from that of what is already there. If for whatever reason, the user wishes to force a write of the location (bad EEPROM?) then the ForceWriteToEEPROM argument allows the user to do that.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_DaughterCard_EEPROM_Write(void          *Handle,
                                     ccrtngfc_dc_eeprom_io_t *eeprom_write,
                                     int                ForceWriteToEEPROM)

```

Description: Daughter Card EEPROM Write

```

Input:  void          *Handle (handle pointer)
        ccrtngfc_dc_eeprom_io_t *eeprom_write
        - _ccrtngfc_dc_eeprom_daughter_card_select_t DaughterCardSelect
          # CCRTNGFC_DCE_DAUGHTER_CARD_0
          # CCRTNGFC_DCE_DAUGHTER_CARD_1
        - uint Address
        - uint Size
        - char *DataPtr
        int                ForceWriteToEEPROM)

```

Output: none

```

Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (library not open)
        # CCRTNGFC_LIB_NO_LOCAL_REGION  (local region not present)
        # CCRTNGFC_LIB_CANNOT_OPEN_FILE (cannot open calib. file)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_DC_EEPROM_FAILURE (Daughter Card EEPROM Failure)
        # CCRTNGFC_LIB_DC_EEPROM_BUSY  (Daughter Card EEPROM Busy)
        # CCRTNGFC_LIB_NO_DAUGHTER_CARD_PRESENT
                                          (Daughter Card Not Present)

```

```

*****/

```

2.2.51 ccrtNGFC_Destroy_AllUserProcess()

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

```

/*****
_ccrtngfc_lib_error_number_t ccrtNGFC_Destroy_AllUserProcess(void *Handle)

```

Description: Destroy all created user processes

```

Input:  void          *Handle (Handle pointer)

```

Output: none

```

Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)

```

*****/

2.2.52 ccrtNGFC_Destroy_UserLDioCosInterruptHandler()

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

/******

```
_ccrtngfc_lib_error_number_t
ccrtNGFC_Destroy_UserLDioCosInterruptHandler(void *Handle)
```

Description: Destroy a previously created User LDIO COS Interrupt Handler

```
Input:  void                *Handle                (Handle pointer)
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR                (successful)
        # CCRTNGFC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN                (device not open)
        # CCRTNGFC_LIB_IOCTL_FAILED            (ioctl failed)
        # CCRTNGFC_LIB_IO_ERROR                (failed to terminate
                                                handler)
```

*****/

2.2.53 ccrtNGFC_Destroy_UserProcess()

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

/******

```
_ccrtngfc_lib_error_number_t ccrtNGFC_Destroy_UserProcess(void *Handle,
                                                           _ccrtngfc_UserFunction_t **UFuncHandle)
```

Description: Destroy an already created user process

```
Input:  void                *Handle                (Handle pointer)
        _ccrtngfc_UserFunction_t **UFuncHandle      (pointer to user handle)
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR                (successful)
        # CCRTNGFC_LIB_BAD_HANDLE              (no/bad handler supplied)
```

*****/

2.2.54 ccrtNGFC_DIO_Get_Channels_Terminator()

This call returns to the user the Digital I/O channel terminators. The ChannelSelectMask is used to select up to 32 channels.

The module selection for the DIO located on the mother-board is CCRTNGFC_MAIN_DIO_MODULE_0. The CCRTNGFC_MAIN_LIO_MODULE_1 is for the LIO module that is located on the mother-board and is therefore invalid for this DIO API. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the CCRTNGFC_DC_DIO_MODULE_2 and/or CCRTNGFC_DC_DIO_MODULE_3 option depending on where the daughter-card is installed.

/******

```
_ccrtngfc_lib_error_number_t
ccrtNGFC_DIO_Get_Channels_Terminator(void                *Handle,
                                     ccrtngfc_ldio_modules_t DIO_ChannelsTerminator,
                                     ccrtngfc_ldio_modules_t ChannelSelectMask)
```

Description: Get DIO Channels Terminator Mask

```

Input:  void                *Handle                (handle pointer)
        ccrtnngfc_ldio_modules_t  ChannelSelectMask  (channel selection mask)
        # NULL                (select all channels)
        # u_int32_t  ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
          # CCRTNGFC_LDIO_CHANNEL_MASK_0
          # CCRTNGFC_LDIO_CHANNEL_MASK_1
          # CCRTNGFC_LDIO_CHANNEL_MASK_2
          # CCRTNGFC_LDIO_CHANNEL_MASK_3
          # CCRTNGFC_LDIO_CHANNEL_MASK_4
          # CCRTNGFC_LDIO_CHANNEL_MASK_5
          # CCRTNGFC_LDIO_CHANNEL_MASK_6
          # CCRTNGFC_LDIO_CHANNEL_MASK_7
          # CCRTNGFC_LDIO_CHANNEL_MASK_8
          # CCRTNGFC_LDIO_CHANNEL_MASK_9
          # CCRTNGFC_LDIO_CHANNEL_MASK_10
          # CCRTNGFC_LDIO_CHANNEL_MASK_11
          # CCRTNGFC_LDIO_CHANNEL_MASK_12
          # CCRTNGFC_LDIO_CHANNEL_MASK_13
          # CCRTNGFC_LDIO_CHANNEL_MASK_14
          # CCRTNGFC_LDIO_CHANNEL_MASK_15
          # CCRTNGFC_LDIO_CHANNEL_MASK_16
          # CCRTNGFC_LDIO_CHANNEL_MASK_17
          # CCRTNGFC_LDIO_CHANNEL_MASK_18
          # CCRTNGFC_LDIO_CHANNEL_MASK_19
          # CCRTNGFC_LDIO_CHANNEL_MASK_20
          # CCRTNGFC_LDIO_CHANNEL_MASK_21
          # CCRTNGFC_LDIO_CHANNEL_MASK_22
          # CCRTNGFC_LDIO_CHANNEL_MASK_23
          # CCRTNGFC_LDIO_CHANNEL_MASK_24
          # CCRTNGFC_LDIO_CHANNEL_MASK_25
          # CCRTNGFC_LDIO_CHANNEL_MASK_26
          # CCRTNGFC_LDIO_CHANNEL_MASK_27
          # CCRTNGFC_LDIO_CHANNEL_MASK_28
          # CCRTNGFC_LDIO_CHANNEL_MASK_29
          # CCRTNGFC_LDIO_CHANNEL_MASK_30
          # CCRTNGFC_LDIO_CHANNEL_MASK_31
          # CCRTNGFC_LDIO_ALL_CHANNELS_MASK
        CCRTNGFC_LDIO_MAX_MODULES can be one of:
          # CCRTNGFC_MAIN_DIO_MODULE_0    // On Main Board - DIO
          # CCRTNGFC_MAIN_LIO_MODULE_1    // On Main Board - LIO
          # CCRTNGFC_DC_DIO_MODULE_2      // optional DIO Daughter Card
          # CCRTNGFC_DC_DIO_MODULE_3      // optional DIO Daughter Card
Output: ccrtnngfc_ldio_modules_t        DIO_ChannelsTerminator (DIO Channels Terminator)
        - CCRTNGFC_DIO_TERMINATOR_OFF   = (0)
        - CCRTNGFC_DIO_TERMINATOR_ON    = (1)
        # u_int32_t  ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
          # CCRTNGFC_LDIO_CHANNEL_MASK_0
          # CCRTNGFC_LDIO_CHANNEL_MASK_1
          # CCRTNGFC_LDIO_CHANNEL_MASK_2
          # CCRTNGFC_LDIO_CHANNEL_MASK_3
          # CCRTNGFC_LDIO_CHANNEL_MASK_4
          # CCRTNGFC_LDIO_CHANNEL_MASK_5
          # CCRTNGFC_LDIO_CHANNEL_MASK_6
          # CCRTNGFC_LDIO_CHANNEL_MASK_7
          # CCRTNGFC_LDIO_CHANNEL_MASK_8
          # CCRTNGFC_LDIO_CHANNEL_MASK_9
          # CCRTNGFC_LDIO_CHANNEL_MASK_10
          # CCRTNGFC_LDIO_CHANNEL_MASK_11
          # CCRTNGFC_LDIO_CHANNEL_MASK_12
          # CCRTNGFC_LDIO_CHANNEL_MASK_13

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card

Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.55 **ccrtNGFC_DIO_Get_Ports_Direction()**

This call returns to the user the direction of the Digital I/O channels.

The module selection for the DIO located on the mother-board is `CCRTNGFC_MAIN_DIO_MODULE_0`. The `CCRTNGFC_MAIN_LIO_MODULE_1` is for the LIO module that is located on the mother-board and is therefore invalid for this DIO API. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the `CCRTNGFC_DC_DIO_MODULE_2` and/or `CCRTNGFC_DC_DIO_MODULE_3` option depending on where the daughter-card is installed.

The DIO that is located on the mother-board has each of the 32 channels controlled by 32 independent ports. Hence, each channels direction can be controlled independent of each other. The optional DIO daughter card has groups of 4 channels controlled by a single port. Hence the direction of the channels must be controlled in groups of 4 adjacent 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.

```

/*****

```

```

ccrtNGFC_DIO_Get_Ports_Direction()

```

Description: Get DIO Ports Direction Mask

```

Input: void *Handle (handle pointer)
       ccrtngfc_dio_ports_t PortSelectionMask (port selection mask)

```

```

# NULL (select all ports)
# _ccrtngfc_dio_port_mask_t ccrtngfc_dio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_DIO_PORT_MASK_P0
# CCRTNGFC_DIO_PORT_MASK_P1
# CCRTNGFC_DIO_PORT_MASK_P2
# CCRTNGFC_DIO_PORT_MASK_P3
# CCRTNGFC_DIO_PORT_MASK_P4
# CCRTNGFC_DIO_PORT_MASK_P5
# CCRTNGFC_DIO_PORT_MASK_P6
# CCRTNGFC_DIO_PORT_MASK_P7
# CCRTNGFC_DIO_PORT_MASK_P8 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P9 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P10 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P11 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P12 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P13 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P14 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P15 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P16 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P17 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P18 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P19 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P20 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P21 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P22 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P23 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P24 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P25 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P26 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P27 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P28 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P29 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P30 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P31 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_ALL_PORTS_MASK - valid for Main DIO only
# CCRTNGFC_DC_DIO_ALL_PORTS_MASK - valid for Daughter Card DIO only
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
Output: ccrtngfc_dio_ports_t DIO_PortDirection (port direction)
- CCRTNGFC_LDIO_DIRECTION_INPUT = (0)
- CCRTNGFC_LDIO_DIRECTION_OUTPUT = (1)
# _ccrtngfc_dio_port_mask_t ccrtngfc_dio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_DIO_PORT_MASK_P0
# CCRTNGFC_DIO_PORT_MASK_P1
# CCRTNGFC_DIO_PORT_MASK_P2
# CCRTNGFC_DIO_PORT_MASK_P3
# CCRTNGFC_DIO_PORT_MASK_P4
# CCRTNGFC_DIO_PORT_MASK_P5
# CCRTNGFC_DIO_PORT_MASK_P6
# CCRTNGFC_DIO_PORT_MASK_P7
# CCRTNGFC_DIO_PORT_MASK_P8 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P9 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P10 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P11 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P12 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P13 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P14 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P15 - not valid for Daughter Card DIO

```



```

# CCRTNGFC_DIO_PORT_MASK_P16      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P17      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P18      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P19      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P20      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P21      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P22      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P23      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P24      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P25      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P26      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P27      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P28      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P29      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P30      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P31      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_ALL_PORTS_MASK     - valid for Main DIO only
# CCRTNGFC_DC_DIO_ALL_PORTS_MASK  - valid for Daughter Card DIO only
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0      // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1      // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2        // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3        // optional DIO Daughter Card
Return:  _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR            (successful)
# CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN            (device not open)
# CCRTNGFC_LIB_INVALID_ARG         (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION     (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE  (LDIO is not active)
*****/

```

2.2.56 ccrtNGFC_DIO_Set_Channels_Terminator()

This call sets the Digital I/O channel terminators to ON or OFF for the selected channels.

The module selection for the DIO located on the mother-board is CCRTNGFC_MAIN_DIO_MODULE_0. The CCRTNGFC_MAIN_LIO_MODULE_1 is for the LIO module that is located on the mother-board and is therefore invalid for this DIO API. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the CCRTNGFC_DC_DIO_MODULE_2 and/or CCRTNGFC_DC_DIO_MODULE_3 option depending on where the daughter-card is installed.

When issuing this call, the users must initialize *all* available DIO modules properly, otherwise the call will fail. If an DIO module is to be skipped in this call, the *ChannelSelectMask* for a DIO module must be set to zero (*i.e. no channels are selected*).

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_DIO_Set_Channels_Terminator(void          *Handle,
                                       ccrtngfc_ldio_modules_t  DIO_ChannelsTerminator,
                                       ccrtngfc_ldio_modules_t  ChannelSelectMask)

```

Description: Set DIO Channels Terminator Mask

```

Input:  void          *Handle          (handle pointer)
        ccrtngfc_ldio_modules_t      DIO_ChannelsTerminator (DIO Channels Terminator)
        - CCRTNGFC_DIO_TERMINATOR_OFF = (0)
        - CCRTNGFC_DIO_TERMINATOR_ON  = (1)
        # u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
        # CCRTNGFC_LDIO_CHANNEL_MASK_0

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
ccrtngfc_ldio_modules_t ChannelSelectMask (channel selection mask)
# NULL (select all channels)
# u_int32_t ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.57 ccrtNGFC_DIO_Set_Channels_Terminator_To_Off()

This call sets the Digital I/O channel terminators to OFF for the selected channels.

The module selection for the DIO located on the mother-board is CCRTNGFC_MAIN_DIO_MODULE_0. The CCRTNGFC_MAIN_LIO_MODULE_1 is for the LIO module that is located on the mother-board and is therefore invalid for this DIO API. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the CCRTNGFC_DC_DIO_MODULE_2 and/or CCRTNGFC_DC_DIO_MODULE_3 option depending on where the daughter-card is installed.

In order to skip a DIO module, simply set *DIO_ChannelsOffTerminator* to zero for that module.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtngfc_DIO_Set_Channels_Terminator_To_Off(void *Handle,
                                             ccrtngfc_ldio_modules_t DIO_ChannelsOffTerminator)

```

Description: Set DIO Channels Terminator Off Mask

```

Input: void *Handle (handle pointer)
       ccrtngfc_ldio_modules_t DIO_ChannelsOffTerminator (DIO Channels Terminator)
       # u_int32_t ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
       # CCRTNGFC_LDIO_CHANNEL_MASK_0
       # CCRTNGFC_LDIO_CHANNEL_MASK_1
       # CCRTNGFC_LDIO_CHANNEL_MASK_2
       # CCRTNGFC_LDIO_CHANNEL_MASK_3
       # CCRTNGFC_LDIO_CHANNEL_MASK_4
       # CCRTNGFC_LDIO_CHANNEL_MASK_5
       # CCRTNGFC_LDIO_CHANNEL_MASK_6
       # CCRTNGFC_LDIO_CHANNEL_MASK_7
       # CCRTNGFC_LDIO_CHANNEL_MASK_8
       # CCRTNGFC_LDIO_CHANNEL_MASK_9
       # CCRTNGFC_LDIO_CHANNEL_MASK_10
       # CCRTNGFC_LDIO_CHANNEL_MASK_11

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.58 ccrtNGFC_DIO_Set_Channels_Terminator_To_On()

This call sets the Digital I/O channel terminators to ON for the selected channels.

The module selection for the DIO located on the mother-board is CCRTNGFC_MAIN_DIO_MODULE_0. The CCRTNGFC_MAIN_LIO_MODULE_1 is for the LIO module that is located on the mother-board and is therefore invalid for this DIO API. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the CCRTNGFC_DC_DIO_MODULE_2 and/or CCRTNGFC_DC_DIO_MODULE_3 option depending on where the daughter-card is installed.

In order to skip a DIO module, simply set *DIO_ChannelsOnTerminator* to zero for that module.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_DIO_Set_Channels_Terminator_To_On(void *Handle,
                                             ccrtngfc_ldio_modules_t DIO_ChannelsOnTerminator)

```

Description: Set DIO Channels Terminator On Mask

```

Input: void *Handle (handle pointer)
       ccrtngfc_ldio_modules_t DIO_ChannelsOnTerminator (DIO Channels Terminator)
       # u_int32_t ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
       # CCRTNGFC_LDIO_CHANNEL_MASK_0
       # CCRTNGFC_LDIO_CHANNEL_MASK_1
       # CCRTNGFC_LDIO_CHANNEL_MASK_2

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.59 ccrtNGFC_DIO_Set_Ports_Direction()

This call sets the direction of the Digital I/O channels.

The module selection for the DIO located on the mother-board is `CCRTNGFC_MAIN_DIO_MODULE_0`. The `CCRTNGFC_MAIN_LIO_MODULE_1` is for the LIO module that is located on the mother-board and is therefore invalid for this DIO API. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the `CCRTNGFC_DC_DIO_MODULE_2` and/or `CCRTNGFC_DC_DIO_MODULE_3` option depending on where the daughter-card is installed.

The DIO that is located on the mother-board has each of the 32 channels controlled by 32 independent ports. Hence, each channels direction can be controlled independent of each other. The optional DIO daughter card has groups of 4 channels controlled by a single port. Hence the direction of the channels must be controlled in groups of 4 adjacent 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.

When issuing this call, the users must initialize *all* available DIO modules properly, otherwise the call will fail. If an DIO module is to be skipped in this call, the *PortSelectionMask* for a DIO module must be set to zero (*i.e. no ports are selected*).

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_DIO_Set_Ports_Direction(void          *Handle,
                                     ccrtnngfc_dio_ports_t    DIO_PortDirection,
                                     ccrtnngfc_dio_ports_t    PortSelectionMask)

```

Description: Set DIO Ports Direction Mask

```

Input:  void          *Handle          (handle pointer)
        ccrtnngfc_dio_ports_t    DIO_PortDirection    (port direction)
        - CCRTNGFC_LDIO_DIRECTION_INPUT    = (0)
        - CCRTNGFC_LDIO_DIRECTION_OUTPUT   = (1)
# _ccrtngfc_dio_port_mask_t ccrtnngfc_dio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_DIO_PORT_MASK_P0
# CCRTNGFC_DIO_PORT_MASK_P1
# CCRTNGFC_DIO_PORT_MASK_P2
# CCRTNGFC_DIO_PORT_MASK_P3
# CCRTNGFC_DIO_PORT_MASK_P4
# CCRTNGFC_DIO_PORT_MASK_P5
# CCRTNGFC_DIO_PORT_MASK_P6
# CCRTNGFC_DIO_PORT_MASK_P7
# CCRTNGFC_DIO_PORT_MASK_P8      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P9      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P10     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P11     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P12     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P13     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P14     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P15     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P16     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P17     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P18     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P19     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P20     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P21     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P22     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P23     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P24     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P25     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P26     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P27     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P28     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P29     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P30     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P31     - not valid for Daughter Card DIO
# CCRTNGFC_DIO_ALL_PORTS_MASK    - valid for Main DIO only
# CCRTNGFC_DC_DIO_ALL_PORTS_MASK - valid for Daughter Card DIO only
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0     // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1     // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2       // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3       // optional DIO Daughter Card
ccrtnngfc_dio_ports_t          PortSelectionMask    (port selection mask)

```

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.

```

# NULL (select all ports)
# _ccrtngfc_dio_port_mask_t ccrtngfc_dio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_DIO_PORT_MASK_P0
# CCRTNGFC_DIO_PORT_MASK_P1
# CCRTNGFC_DIO_PORT_MASK_P2
# CCRTNGFC_DIO_PORT_MASK_P3
# CCRTNGFC_DIO_PORT_MASK_P4
# CCRTNGFC_DIO_PORT_MASK_P5
# CCRTNGFC_DIO_PORT_MASK_P6
# CCRTNGFC_DIO_PORT_MASK_P7
# CCRTNGFC_DIO_PORT_MASK_P8 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P9 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P10 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P11 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P12 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P13 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P14 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P15 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P16 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P17 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P18 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P19 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P20 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P21 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P22 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P23 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P24 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P25 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P26 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P27 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P28 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P29 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P30 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P31 - not valid for Daughter Card DIO
# CCRTNGFC_DIO_ALL_PORTS_MASK - valid for Main DIO only
# CCRTNGFC_DC_DIO_ALL_PORTS_MASK - valid for Daughter Card DIO only
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.60 ccrtNGFC_DIO_Set_Ports_Direction_To_Input()

This call sets the direction of the Digital I/O channels to Inputs.

The module selection for the DIO located on the mother-board is CCRTNGFC_MAIN_DIO_MODULE_0. The CCRTNGFC_MAIN_LIO_MODULE_1 is for the LIO module that is located on the mother-board and is therefore invalid for this DIO API. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the CCRTNGFC_DC_DIO_MODULE_2 and/or CCRTNGFC_DC_DIO_MODULE_3 option depending on where the daughter-card is installed.

The DIO that is located on the mother-board has each of the 32 channels controlled by 32 independent ports. Hence, each channels direction can be controlled independent of each other. The optional DIO daughter card has groups of 4 channels controlled by a single port. Hence the direction of the channels must be controlled in groups of 4 adjacent 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.

In order to skip a DIO module, simply set *DIO_InputPortDirection* to zero for that module.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_DIO_Set_Ports_Direction_To_Input(void          *Handle,
                                           ccrtngfc_dio_ports_t  DIO_InputPortDirection)

```

Description: Set DIO Port Direction to Input Mask

```

Input:  void          *Handle          (handle pointer)
        ccrtngfc_dio_ports_t          DIO_InputPortDirection (input port direction)
        # _ccrtngfc_dio_port_mask_t ccrtngfc_dio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
          # CCRTNGFC_DIO_PORT_MASK_P0
          # CCRTNGFC_DIO_PORT_MASK_P1
          # CCRTNGFC_DIO_PORT_MASK_P2
          # CCRTNGFC_DIO_PORT_MASK_P3
          # CCRTNGFC_DIO_PORT_MASK_P4
          # CCRTNGFC_DIO_PORT_MASK_P5
          # CCRTNGFC_DIO_PORT_MASK_P6
          # CCRTNGFC_DIO_PORT_MASK_P7
          # CCRTNGFC_DIO_PORT_MASK_P8          - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P9          - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P10         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P11         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P12         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P13         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P14         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P15         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P16         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P17         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P18         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P19         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P20         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P21         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P22         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P23         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P24         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P25         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P26         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P27         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P28         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P29         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P30         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P31         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_ALL_PORTS_MASK        - valid for Main DIO only
          # CCRTNGFC_DC_DIO_ALL_PORTS_MASK     - valid for Daughter Card DIO only
        CCRTNGFC_LDIO_MAX_MODULES can be one of:
          # CCRTNGFC_MAIN_DIO_MODULE_0        // On Main Board - DIO
          # CCRTNGFC_MAIN_LIO_MODULE_1        // On Main Board - LIO
          # CCRTNGFC_DC_DIO_MODULE_2         // optional DIO Daughter Card
          # CCRTNGFC_DC_DIO_MODULE_3         // optional DIO Daughter Card

```

Output: none


```

Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.61 ccrtNGFC_DIO_Set_Ports_Direction_To_Output()

This call sets the direction of the Digital I/O channels to Outputs.

The module selection for the DIO located on the mother-board is CCRTNGFC_MAIN_DIO_MODULE_0. The CCRTNGFC_MAIN_LIO_MODULE_1 is for the LIO module that is located on the mother-board and is therefore invalid for this DIO API. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the CCRTNGFC_DC_DIO_MODULE_2 and/or CCRTNGFC_DC_DIO_MODULE_3 option depending on where the daughter-card is installed.

The DIO that is located on the mother-board has each of the 32 channels controlled by 32 independent ports. Hence, each channels direction can be controlled independent of each other. The optional DIO daughter card has groups of 4 channels controlled by a single port. Hence the direction of the channels must be controlled in groups of 4 adjacent 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).

In order to skip a DIO module, simply set *DIO_OutputPortDirection* to zero for that module.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_DIO_Set_Ports_Direction_To_Output(void          *Handle,
                                             ccrtngfc_dio_ports_t  DIO_OutputPortDirection)

```

Description: Set DIO Port Direction to Output Mask

```

Input:  void          *Handle          (handle pointer)
        ccrtngfc_dio_ports_t  DIO_OutputPortDirection (output port direction)
        # _ccrtngfc_dio_port_mask_t ccrtngfc_dio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
          # CCRTNGFC_DIO_PORT_MASK_P0
          # CCRTNGFC_DIO_PORT_MASK_P1
          # CCRTNGFC_DIO_PORT_MASK_P2
          # CCRTNGFC_DIO_PORT_MASK_P3
          # CCRTNGFC_DIO_PORT_MASK_P4
          # CCRTNGFC_DIO_PORT_MASK_P5
          # CCRTNGFC_DIO_PORT_MASK_P6
          # CCRTNGFC_DIO_PORT_MASK_P7
          # CCRTNGFC_DIO_PORT_MASK_P8          - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P9          - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P10         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P11         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P12         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P13         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P14         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P15         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P16         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P17         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P18         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P19         - not valid for Daughter Card DIO
          # CCRTNGFC_DIO_PORT_MASK_P20         - not valid for Daughter Card DIO

```

```

# CCRTNGFC_DIO_PORT_MASK_P21      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P22      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P23      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P24      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P25      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P26      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P27      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P28      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P29      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P30      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_PORT_MASK_P31      - not valid for Daughter Card DIO
# CCRTNGFC_DIO_ALL_PORTS_MASK     - valid for Main DIO only
# CCRTNGFC_DC_DIO_ALL_PORTS_MASK  - valid for Daughter Card DIO only
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0      // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1      // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2        // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3        // optional DIO Daughter Card

Output:  none
Return:  _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR            (successful)
# CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN            (device not open)
# CCRTNGFC_LIB_INVALID_ARG         (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION     (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE  (LDIO is not active)
*****/

```

2.2.62 ccrtNGFC_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.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Disable_Pci_Interrupts(void *Handle,
                                   _ccrtngfc_all_interrupts_mask interrupt_mask)

```

Description: Disable interrupts being generated by the board.

```

Input:  void *Handle (Handle pointer)
        _ccrtngfc_all_interrupts_mask interrupt_mask (interrupt mask)
# CCRTNGFC_MSGDMA0_INTMASK
# CCRTNGFC_MSGDMA1_INTMASK
# CCRTNGFC_MSGDMA2_INTMASK
# CCRTNGFC_MSGDMA3_INTMASK
# CCRTNGFC_MSGDMA4_INTMASK
# CCRTNGFC_MSGDMA5_INTMASK
# CCRTNGFC_LDIO_MODULE_0_INTMASK
# CCRTNGFC_LDIO_MODULE_1_INTMASK
# CCRTNGFC_LDIO_MODULE_2_INTMASK
# CCRTNGFC_LDIO_MODULE_3_INTMASK
# CCRTNGFC_ADC0_FIFO_INTMASK
# CCRTNGFC_DAC0_FIFO_INTMASK
# CCRTNGFC_ADC1_FIFO_INTMASK
# CCRTNGFC_DAC1_FIFO_INTMASK
# CCRTNGFC_ALL_MSGDMA_INTMASK
# CCRTNGFC_ALL_ANALOG_INTMASK
# CCRTNGFC_ALL_LDIO_INTMASK
# CCRTNGFC_ALL_INTMASK

```

Output: none

```

Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_IOCTL_FAILED      (driver ioctl call failed)
*****

```

2.2.63 ccrtNGFC_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.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Enable_Pci_Interrupts (void          *Handle,
                               _ccrtngfc_all_interrupts_mask interrupt_mask)

```

Description: Enable interrupts being generated by the board.

```

Input:  void          *Handle          (Handle pointer)
        _ccrtngfc_all_interrupts_mask interrupt_mask (interrupt mask)
        # CCRTNGFC_MSGDMA0_INTMASK
        # CCRTNGFC_MSGDMA1_INTMASK
        # CCRTNGFC_MSGDMA2_INTMASK
        # CCRTNGFC_MSGDMA3_INTMASK
        # CCRTNGFC_MSGDMA4_INTMASK
        # CCRTNGFC_MSGDMA5_INTMASK
        # CCRTNGFC_LDIO_MODULE_0_INTMASK
        # CCRTNGFC_LDIO_MODULE_1_INTMASK
        # CCRTNGFC_LDIO_MODULE_2_INTMASK
        # CCRTNGFC_LDIO_MODULE_3_INTMASK
        # CCRTNGFC_ADC0_FIFO_INTMASK
        # CCRTNGFC_DAC0_FIFO_INTMASK
        # CCRTNGFC_ADC1_FIFO_INTMASK
        # CCRTNGFC_DAC1_FIFO_INTMASK
        # CCRTNGFC_ALL_MSGDMA_INTMASK
        # CCRTNGFC_ALL_ANALOG_INTMASK
        # CCRTNGFC_ALL_LDIO_INTMASK
        # CCRTNGFC_ALL_INTMASK

```

Output: none

```

Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_IOCTL_FAILED      (driver ioctl call failed)
*****

```

2.2.64 ccrtNGFC_Fast_Memcpy()

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

```

/*****
ccrtNGFC_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: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN      (device not open)
*****/

```

2.2.65 ccrtNGFC_Fast_Memcpy_Unlocked()

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

```

/*****
void
ccrtNGFC_Fast_Memcpy_Unlocked(volatile void *Destination,
                              volatile void *Source,
                              int           SizeInBytes
                              int           DoubleWordSupport)

```

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

```

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

```

2.2.66 ccrtNGFC_Fast_Memcpy_Unlocked_FIFO()

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

```

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

```

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

```

Input:  volatile void *Source          (pointer to source buffer)
        int           SizeInWords     (transfer size in words)
        int           PioControl      (PIO Control)
        # CCRTNGFC_PIO_CONTROL_RCON  (read constant)
        # CCRTNGFC_PIO_CONTROL_WCON  (write constant)
        # CCRTNGFC_PIO_CONTROL_INCREMENT (read/write increment)
        int           DoubleWordSupport (double word support flag)

```

```

        # CCRTNGFC_FALSE           (h/w double word transfers not supported)
        # CCRTNGFC_TRUE            (h/w double word transfers supported)
Output: volatile void *Destination (pointer to destination buffer)
Return: none
*****/

```

2.2.67 ccrtNGFC_Fraction_To_Hex()

This converts a fractional decimal to a hexadecimal value.

```

/*****
int
ccrtNGFC_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)
*****/

```

2.2.68 ccrtNGFC_Get_All_Boards_Driver_Info()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Get_All_Boards_Driver_Info (void *Handle,
                                     ccrtngfc_all_boards_driver_info *all_boards_info)

Description: Get device information from driver for all boards.

Input:  void *Handle (Handle pointer)
Output: ccrtngfc_driver_info_t *all_boards_info (info struct pointer)
        char          version[12]
        char          built[32]
        char          module_name[16]
        int           board_index
        int           table_index
        char          board_desc[32]
        int           bus
        int           slot
        int           func
        int           vendor_id
        int           sub_vendor_id
        int           sub_device_id
        union {
            u_int      BoardInfo
            ccrtngfc_boardinfo_t BInfo
        }
        union {
            u_int      FirmwareDate
            ccrtngfc_firmware_date_t FmDate
        }
        union {
            u_int      FirmwareRevision
            ccrtngfc_firmware_revision_t FmRev
        }
        int           msi_support

```

```

int                                     irqlevel
double                                 calibration_reference_voltage

// MsgDMA
ccrtngfc_driver_msgdma_info_t          msgdma_info

// Interrupt
ccrtngfc_driver_int_t                  interrupt

// LDIO COS Interrupt
ccrtngfc_driver_ldio_cos_int_t         ldio_cos_interrupt

int                                     Ccrtngfc_Max_Region

// Memory Region
ccrtngfc_dev_region_t                  mem_region[CCRTNGFC_MAX_REGION]

// ADC
ccrtngfc_driver_adc_info_t             adc_info

// DAC
ccrtngfc_driver_dac_info_t             dac_info

// LDIO
int                                     ldio_max_modules
ccrtngfc_driver_ldio_info_t            ldio_info[CCRTNGFC_LDIO_MAX_MODULES]

// CLOCK
ccrtngfc_driver_clock_info_t           clock_info
u_int                                   board_serial_number;

// Chip Temperature
ccrtngfc_chip_temperature_t            fpga_chip_temperature

ccrtngfc_chip_voltages_t               fpga_chip_volts

char                                    double_word_support
char                                    FpgawbSupport

union {
    u_int                                FirmwareTime
    ccrtngfc_firmware_time_t            FmTime
}
union {
    u_int                                FirmwareFlavorCode
    ccrtngfc_firmware_option_code_t     FmOptionCode
}
u_int                                   NumberAdvancedIPCores

u_short                                 RunLevelSectorNumber
char                                    FirmwareReloadFailed
char                                    MultiFirmwareSupport

union {
    _ccrtngfc_ipcore_t                  IpCore[CCRTNGFC_MAX_IO_CORES_ORIG]
    _ccrtngfc_ipcore_t                  IpCore_Orig[CCRTNGFC_MAX_IO_CORES_ORIG]
}

union {
    u_int                                Dummy_time_t[2]
    time_t                                DriverLoadCurrentTime
}

```

```

    }

    u_int32_t                FirmwareBoardSerialNumber

    u_int32_t                MaxMsgDmaDescriptors
    ccrtngfc_msgdma_specific_t  MsgDmaSpecific[CCRTNGFC_MAX_MSGDMA_ENGINES]

    u_int32_t                FpgawbRevision

    _ccrtngfc_ipcore_t      IpCore[CCRTNGFC_TOTAL_NUMBER_OF_IP_CORES]

    int                     CloningSupport

    u_short                 MaximumLinkWidth
    u_short                 NegotiatedLinkWidth

Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN         (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_IOCTL_FAILED    (driver ioctl call failed)
*****/

```

2.2.69 ccrtNGFC_Get_Board_CSR()

This call returns information from the board status register.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Get_Board_CSR (void                *Handle,
                      ccrtngfc_board_csr_t *bcsr)

Description: Get Board Control and Status information

Input:  void                *Handle (Handle pointer)
Output: ccrtngfc_board_csr_t *bcsr (pointer to board csr)
        _ccrtngfc_bcsr_identify_board_t identify_board
        # CCRTNGFC_BCSR_IDENTIFY_BOARD_DISABLE
        # CCRTNGFC_BCSR_IDENTIFY_BOARD_ENABLE

Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN         (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
*****/

```

2.2.70 ccrtNGFC_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.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Get_Board_Info (void                *Handle,
                      ccrtngfc_board_info_t *binfo)

Description: Get Board Information

Input:  void                *Handle (Handle pointer)
Output: ccrtngfc_board_info_t *binfo (pointer to board info)
        int                 vendor_id

```

```

        int                                sub_vendor_id
        int                                sub_device_id
        ccrtnngfc_boardinfo_t              BInfo
            u_char Function
            u_char Type
            u_short Id
        ccrtnngfc_firmware_date_t          FmDate
            u_short Year
            u_char Day
            u_char Month
        ccrtnngfc_firmware_revision_t      FmRev
            u_short Minor
            u_short Major
Return:  _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR            (successful)
        # CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN           (device not open)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION    (local region not present)
*****

```

2.2.71 ccrtnngfc_Get_Driver_Error()

This call returns the last error generated by the driver.

```

/*****
_ccrtnngfc_lib_error_number_t
ccrtnngfc_Get_Driver_Error (void                *Handle,
                           ccrtnngfc_user_error_t *ret_err)

Description: Get the last error generated by the driver.

Input:  void                *Handle            (Handle pointer)
Output: ccrtnngfc_user_error_t *ret_err        (error struct pointer)
        uint error;          (error number)
        char name[CCRTNGFC_ERROR_NAME_SIZE]    (error name used in driver)
        char desc[CCRTNGFC_ERROR_DESC_SIZE]    (error description)
        uint error_line_number (error line number)

Return:  _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR            (successful)
        # CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN           (device not open)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
        # CCRTNGFC_LIB_IOCTL_FAILED       (driver ioctl call failed)
*****

```

```

#define CCRTNGFC_ERROR_NAME_SIZE      64
#define CCRTNGFC_ERROR_DESC_SIZE     128
#define CCRTNGFC_DRIVER_ERROR_FUNC_SIZE 200

```

```

typedef struct _ccrtnngfc_user_error_t
{
    uint error;                /* error number */
    char name[CCRTNGFC_ERROR_NAME_SIZE]; /* error name used in driver */
    char desc[CCRTNGFC_ERROR_DESC_SIZE]; /* error description */
    uint error_line_number;    /* error line number */
    char error_function[CCRTNGFC_DRIVER_ERROR_FUNC_SIZE];
    int RESERVED[100];
} ccrtnngfc_user_error_t;

```

enum


```

{
    CCRTNGFC_SUCCESS = 0,
    CCRTNGFC_INVALID_PARAMETER,
    CCRTNGFC_OPERATION_CANCELLED,
    CCRTNGFC_RESOURCE_ALLOCATION_ERROR,
    CCRTNGFC_INVALID_REQUEST,
    CCRTNGFC_FAULT_ERROR,
    CCRTNGFC_BUSY,
    CCRTNGFC_ADDRESS_IN_USE,
    CCRTNGFC_USER_INTERRUPT_TIMEOUT,
    CCRTNGFC_DATA_UNDERFLOW,
    CCRTNGFC_DATA_OVERFLOW, :
    CCRTNGFC_IO_FAILURE,
    CCRTNGFC_OPERATION_NOT_SUPPORTED,
    CCRTNGFC_ADC_FIFO_THRESHOLD_TIMEOUT,
    CCRTNGFC_DAC_FIFO_THRESHOLD_TIMEOUT,
    CCRTNGFC_INTERRUPT_HANDLER_NOT_ENABLED,
    CCRTNGFC_FIRMWARE_RELOAD_FAILED,
    CCRTNGFC_DEVICE_AUTHORIZATION_FAILED,
    CCRTNGFC_DAUGHTER_CARD_NOT_PRESENT,
    CCRTNGFC_DAUGHTER_CARD_BUSY,
    CCRTNGFC_BOARD_NOT_OPENED_FIRST,
};

```

2.2.72 ccrtNGFC_Get_Driver_Info()

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

```

/*****

```

```

    _ccrtngfc_lib_error_number_t
    ccrtNGFC_Get_Driver_Info (void          *Handle,
                             ccrtngfc_driver_info_t *info)

```

Description: Get device information from driver.

Input:	void	*Handle	(Handle pointer)
Output:	ccrtngfc_driver_info_t	*all_boards_info	(info struct pointer)
	char		version[12]
	char		built[32]
	char		module_name[16]
	int		board_index
	int		table_index
	char		board_desc[32]
	int		bus
	int		slot
	int		func
	int		vendor_id
	int		sub_vendor_id
	int		sub_device_id
	union {		
	u_int		BoardInfo
	ccrtngfc_boardinfo_t		BInfo
	}		
	union {		
	u_int		FirmwareDate
	ccrtngfc_firmware_date_t		FmDate
	}		
	union {		
	u_int		FirmwareRevision
	ccrtngfc_firmware_revision_t		FmRev
	}		

```

int          msi_support
int          irqlevel
double      calibration_reference_voltage

// MsgDMA
ccrtngfc_driver_msgdma_info_t  msgdma_info

// Interrupt
ccrtngfc_driver_int_t          interrupt

// LDIO COS Interrupt
ccrtngfc_driver_ldio_cos_int_t ldio_cos_interrupt

int          Ccrtngfc_Max_Region

// Memory Region
ccrtngfc_dev_region_t          mem_region[CCRTNGFC_MAX_REGION]

// ADC
ccrtngfc_driver_adc_info_t     adc_info

// DAC
ccrtngfc_driver_dac_info_t     dac_info

// LDIO
int          ldio_max_modules
ccrtngfc_driver_ldio_info_t    ldio_info[CCRTNGFC_LDIO_MAX_MODULES]

// CLOCK
ccrtngfc_driver_clock_info_t   clock_info
u_int       board_serial_number;

// Chip Temperature
ccrtngfc_chip_temperature_t    fpga_chip_temperature

ccrtngfc_chip_voltages_t       fpga_chip_volts

char        double_word_support
char        FpgawbSupport

union {
    u_int    FirmwareTime
    ccrtngfc_firmware_time_t  FmTime
}
union {
    u_int    FirmwareFlavorCode
    ccrtngfc_firmware_option_code_t  FmOptionCode
}
u_int       NumberAdvancedIPCores

u_short    RunLevelSectorNumber
char        FirmwareReloadFailed
char        MultiFirmwareSupport

union {
    _ccrtngfc_ipcore_t    IpCore[CCRTNGFC_MAX_IO_CORES_ORIG]
    _ccrtngfc_ipcore_t    IpCore_Orig[CCRTNGFC_MAX_IO_CORES_ORIG]
}

union {
    u_int    Dummy_time_t[2]
}

```

```

        time_t                DriverLoadCurrentTime
    }

    u_int32_t                FirmwareBoardSerialNumber

    u_int32_t                MaxMsgDmaDescriptors
    ccrtngfc_msgdma_specific_t  MsgDmaSpecific[CCRTNGFC_MAX_MSGDMA_ENGINES]

    u_int32_t                FpgawbRevision

    _ccrtngfc_ipcore_t      IpCore[CCRTNGFC_TOTAL_NUMBER_OF_IP_CORES]

    int                      CloningSupport

    u_short                 MaximumLinkWidth
    u_short                 NegotiatedLinkWidth

Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN         (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_IOCTL_FAILED    (driver ioctl call failed)
*****/

```

2.2.73 ccrtNGFC_Get_Interrupt_Status()

This call returns the current status of the various interrupts.

```

/*****
    _ccrtngfc_lib_error_number_t
    ccrtNGFC_Get_Interrupt_Status (void          *Handle,
                                   ccrtngfc_interrupt_t *intr)

Description: Get Interrupt Status information

Input:  void          *Handle      (handle pointer)
Output: ccrtngfc_interrupt_t *intr  (pointer to interrupt status)
        _ccrtngfc_intsta_ldio_cos_t ldio_cos_module_int[CCRTNGFC_LDIO_MAX_MODULES];
        # CCRTNGFC_INT_LDIO_COS_NONE
        # CCRTNGFC_INT_LDIO_COS_OCCURRED

Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN         (device not open)
        # CCRTNGFC_LIB_NO_LOCAL_REGION  (local region error)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
*****/

```

2.2.74 ccrtNGFC_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.

```

/*****
    _ccrtngfc_lib_error_number_t
    ccrtNGFC_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: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN      (device not open)
        # CCRTNGFC_LIB_INVALID_ARG   (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTNGFC_LIB_IOCTL_FAILED  (ioctl error)
*****/

```

2.2.75 ccrtNGFC_Get_Lib_Error()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Get_Lib_Error (void          *Handle,
                       ccrtngfc_lib_error_t *lib_error)

Description: Get last error generated by the library.

Input:  void          *Handle          (Handle pointer)
Output: ccrtngfc_lib_error_t *lib_error (error struct pointer)
        -- uint error                (last library error number)
        -- char name[CCRTNGFC_LIB_ERROR_NAME_SIZE] (last library error name)
        -- char desc[CCRTNGFC_LIB_ERROR_DESC_SIZE] (last library error description)
        -- int line_number            (last library error line number in lib)
        -- char function[CCRTNGFC_LIB_ERROR_FUNC_SIZE]
                                       (library function in error)
        -- ccrtngfc_lib_error_backtrace_t BT[CCRTNGFC_BACK_TRACE_DEPTH]
                                       (backtrace of errors)
        -- int line_number            (line number in library)
        -- char function[CCRTNGFC_LIB_ERROR_FUNC_SIZE]
                                       (library function)

Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN      (device not open)
        # CCRTNGFC_LIB_AUTHORIZATION_FAILURE (device authorization failure)
*****/

```

```

typedef struct
{
    int line_number;                /* line number in library */
    char function[CCRTNGFC_LIB_ERROR_FUNC_SIZE]; /* library function */
    uint RESERVED_INT[256];        /* Spare registers for future use */
} ccrtngfc_lib_error_backtrace_t;
typedef struct
{
    uint error;                    /* last library error number */
    char name[CCRTNGFC_LIB_ERROR_NAME_SIZE]; /* last library error name */
    char desc[CCRTNGFC_LIB_ERROR_DESC_SIZE]; /* last library error description */
    int line_number;              /* last library error line number in lib */
    char function[CCRTNGFC_LIB_ERROR_FUNC_SIZE]; /* library function in error */
    uint RESERVED_INT[256];      /* Spare registers for future use */

    ccrtngfc_lib_error_backtrace_t BT[CCRTNGFC_BACK_TRACE_DEPTH];
                                       /* backtrace of errors */
} ccrtngfc_lib_error_t;

```

Possible library errors:

CCRTNGFC_LIB_NO_ERROR	= 0, /* Successful */
CCRTNGFC_LIB_INVALID_ARG	= -1, /* Invalid argument */
CCRTNGFC_LIB_ALREADY_OPEN	= -2, /* Already open */
CCRTNGFC_LIB_OPEN_FAILED	= -3, /* Open failed */
CCRTNGFC_LIB_BAD_HANDLE	= -4, /* Bad handle */
CCRTNGFC_LIB_NOT_OPEN	= -5, /* Device not opened */
CCRTNGFC_LIB_MMAP_SELECT_FAILED	= -6, /* Mmap selection failed */
CCRTNGFC_LIB_MMAP_FAILED	= -7, /* Mmap failed */
CCRTNGFC_LIB_MUNMAP_FAILED	= -8, /* Munmap failed */
CCRTNGFC_LIB_NOT_MAPPED	= -9, /* Not mapped */
CCRTNGFC_LIB_ALREADY_MAPPED	= -10, /* Device already mapped */
CCRTNGFC_LIB_IOCTL_FAILED	= -11, /* Device IOCTL failed */
CCRTNGFC_LIB_IO_ERROR	= -12, /* I/O error */
CCRTNGFC_LIB_INTERNAL_ERROR	= -13, /* Internal library error */
CCRTNGFC_LIB_NOT_IMPLEMENTED	= -14, /* Call not implemented */
CCRTNGFC_LIB_LOCK_FAILED	= -15, /* Failed to get lib lock */
CCRTNGFC_LIB_NO_LOCAL_REGION	= -16, /* Local region not present */
CCRTNGFC_LIB_NO_CONFIG_REGION	= -17, /* Config region not present */
CCRTNGFC_LIB_NO_SOLUTION_FOUND	= -18, /* No solution found */
CCRTNGFC_LIB_NO_RESOURCE	= -19, /* Resource not available */
CCRTNGFC_LIB_CANNOT_OPEN_FILE	= -20, /* Cannot open file */
CCRTNGFC_LIB_MSGDMA_BUSY	= -21, /* MSGDMA busy */
CCRTNGFC_LIB_AVALON_TRANSLATION_TABLE	= -22, /* Avalon translation table error */
CCRTNGFC_LIB_ADDRESS_RANGE_ERROR	= -23, /* Physical DMA address exceeds memory size */
CCRTNGFC_LIB_NO_SPACE_IN_TABLE	= -24, /* No space available to allocate any more physical memory */
CCRTNGFC_LIB_CANNOT_ALLOCATE_PHYS_MEM	= -25, /* Cannot allocate physical memory */
CCRTNGFC_LIBMSG_MSGDMA_FAILED	= -26, /* MSGDMA failed */
CCRTNGFC_LIB_THREAD_CREATE_FAILED	= -27, /* Thread Creation failed */
CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE	= -28, /* Clock Generator is not active */
CCRTNGFC_LIB_CANNOT_COMPUTE_OUTPUT_FREQ	= -29, /* Cannot compute output frequency */
CCRTNGFC_LIB_N_DIVIDERS_EXCEEDED	= -30, /* Number of N-Dividers exceeded */
CCRTNGFC_LIB_CLOCK_GENERATION_FAILED	= -31, /* Clock generation failed */
CCRTNGFC_LIB_CALIBRATION_RANGE_ERROR	= -32, /* Calibration voltage out of range */
CCRTNGFC_LIB_BAD_DATA_IN_CAL_FILE	= -33, /* Bad data in calibration file */
CCRTNGFC_LIB_VOLTAGE_NOT_IN_RANGE	= -34, /* Voltage not in range */
CCRTNGFC_LIB_ADC_IS_NOT_ACTIVE	= -35, /* ADC is not active */
CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE	= -36, /* DAC is not active */
CCRTNGFC_LIB_ADC_INCORRECTLY_CONFIGURED	= -37, /* ADC incorrectly configured for DAC readback */
CCRTNGFC_LIB_LIO_TEST_MODE_SETTING_ERROR	= -39, /* Only one output port must be set in LIO test mode */
CCRTNGFC_LIB_DAC_FIFO_UNDERFLOW	= -40, /* DAC FIFO underflow */
CCRTNGFC_LIB_DAC_FIFO_OVERFLOW	= -41, /* DAC FIFO overflow */
CCRTNGFC_LIB_DAC_IS_BUSY	= -42, /* DAC is busy */
CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE	= -43, /* LDIO is not active */
CCRTNGFC_LIB_SERIAL_PROM_FAILURE	= -44, /* Serial PROM failure - malfunction or not present */
CCRTNGFC_LIB_SERIAL_PROM_BUSY	= -45, /* Serial PROM busy */
CCRTNGFC_LIB_SERIAL_PROM_WRITE_PROTECTED	= -46, /* Serial PROM is write protected */
CCRTNGFC_LIB_AUTHORIZATION_FAILURE	= -47, /* Failure to authorize opening of device */
CCRTNGFC_LIB_INTHDLR_CREATE_FAILURE	= -48, /* Interrupt handler creation failure */
CCRTNGFC_LIB_INTHDLR_ALREADY_RUNNING	= -49, /* Interrupt handler already running */
CCRTNGFC_LIB_IPCORE_COS_IS_NOT_ACTIVE	= -50, /* IP Core COS is Not active */
CCRTNGFC_LIB_NO_FREE_DESCRIPTOR_AVAILABLE	= -51, /* No Free Descriptors Available */
CCRTNGFC_LIB_ERROR_IN_DESCRIPTOR_LIST	= -52, /* Error in Descriptor List */
CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED	= -53, /* Modular Scatter-Gather DMA Not Supported */
CCRTNGFC_LIB_MSGDMA_READS_NOT_ALLOWED_FOR_SELECTED_ADDRESS	= -54, /* MSG DMA Reads Not Allowed for Selected Address */

```

CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA           = -55, /* Not Owner of Modular Scatter-Gather DMA */
CCRTNGFC_LIB_MSGDMA_IN_USE                 = -56, /* Modular Scatter-Gather DMA In Use */
CCRTNGFC_LIB_MSGDMA_NOT_SETUP              = -57, /* Modular Scatter-Gather DMA not setup */
CCRTNGFC_LIB_MSGDMA_FAILED                 = -58, /* Modular Scatter-Gather DMA failed */
CCRTNGFC_LIB_MSGDMA_ACCESS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                           = -59, /* MSGDMA access not allowed for selected
                                           address */

CCRTNGFC_LIB_REGION_ADDRESSING_NOT_SUPPORTED
                                           = -60, /* Region addressing not supported by driver */
CCRTNGFC_LIB_CLONING_NOT_SUPPORTED         = -61, /* Cloning not supported by the card */
CCRTNGFC_LIB_SERIAL_PROM_NOT_PRESENT       = -62, /* Serial PROM not present */
CCRTNGFC_LIB_DC_EEPROM_FAILURE             = -63, /* Daughter Card EEPROM Failure */
CCRTNGFC_LIB_DC_EEPROM_BUSY                = -64, /* Daughter Card EEPROM Busy */
CCRTNGFC_LIB_MODULE_NOT_OPEN               = -65, /* Module needs to be opened first */
CCRTNGFC_LIB_MODULE_ALREADY_OPEN           = -66, /* Module Already Open */
CCRTNGFC_LIB_MODULE_NOT_PRESENT            = -67, /* Module Not Present */
CCRTNGFC_LIB_INVALID_MODULE                = -68, /* Invalid Module */
CCRTNGFC_LIB_DC_BAD_HANDLE                 = -69, /* Bad daughter card handle supplied */
CCRTNGFC_LIB_INSTEAD_OF_DC_HANDLE          = -70, /* Library instead of daughter card handle
                                           supplied */

CCRTNGFC_LIB_MODULE_OPEN_FAILED            = -71, /* Module Open Failed */
CCRTNGFC_LIB_DATA_WIDTH_NOT_MULTIPLE_OR_ALIGNED
                                           = -72, /* Data Width Not Multiple or Aligned */

CCRTNGFC_LIB_FPGA_PM_FAILURE                = -73, /* FPGA Power Module Failure */
CCRTNGFC_LIB_FPGA_PM_BUSY                  = -74, /* FPGA Power Module Busy */
CCRTNGFC_LIB_NO_DAUGHTER_CARD_PRESENT      = -75, /* No Daughter Card Present */
CCRTNGFC_LIB_PLL_SYNC_NOT_SUPPORTED        = -76, /* PLL Synchronization not supported */

```

2.2.76 ccrtNGFC_Get_Lib_Error_Description()

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

```

/*****
void
ccrtNGFC_Get_Lib_Error_Description (int          ErrorNumber,
                                   ccrtngfc_lib_error_description_t *lib_error_desc)

Description: Get Error Description of supplied error number.

Input:  int          ErrorNumber      (Library error number)
Output: ccrtngfc_lib_error_description_t *lib_error_desc (error description struct pointer)
        -- int found
        -- char name[CCRTNGFC_LIB_ERROR_NAME_SIZE] (last library error name)
        -- char desc[CCRTNGFC_LIB_ERROR_DESC_SIZE] (last library error description)

Return: none
*****/

```

2.2.77 ccrtNGFC_Get_Library_Info()

This call returns useful library information to the user.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Get_Library_Info (void          *Handle,
                           ccrtngfc_library_info_t *info)

Description: Get library information

Input:  void          *Handle      (Handle pointer)
Output: ccrtngfc_library_info_t    *info      (info struct pointer)
        int          fp
        ccrtngfc_local_ctrl_data_t *local_ptr
        -- structure in ccrtngfc_user.h
*****/

```

```

void                *munmap_local_ptr
int                local_mmap_size
ccrtngfc_config_local_data_t  *config_ptr
    -- structure in ccrtngfc_user.h
void                *munmap_config_ptr
int                config_mmap_size
ccrtngfc_user_phys_mem_t    PhysMem[CCRTNGFC_MAX_AVALON_NUM_TRANS_TBL_ENTRIES]
    -- structure in ccrtngfc_user.h
ccrtngfc_driver_library_common_t  *driver_lib_ptr
    -- structure in ccrtngfc_user.h
void                *munmap_driver_lib_ptr
int                driver_lib_mmap_size
void                *FpgaWbLib;
_ccrtngfc_ipcore_t    IpCoreSpecific[CCRTNGFC_TOTAL_NUMBER_OF_IP_CORES];
uint                UserPid;
void                *IpCore_0_ptr
void                *IpCore_1_ptr
void                *IpCore_2_ptr
void                *IpCore_3_ptr
void                *IpCore_4_ptr
void                *IpCore_5_ptr
void                *IpCore_6_ptr
void                *IpCore_7_ptr
void                *IpCore_8_ptr
void                *IpCore_9_ptr
void                *IpCore_10_ptr
.
.
.
void                *IpCore_140_ptr
void                *IpCore_141_ptr
void                *IpCore_142_ptr
void                *IpCore_143_ptr
void                *IpCore_144_ptr
void                *IpCore_145_ptr
void                *IpCore_146_ptr
void                *IpCore_147_ptr
u_short    IpCore_MaxChannels_In_License[CCRTNGFC_TOTAL_NUMBER_OF_IP_CORES];

```

```

Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)

```

*****/

2.2.78 ccrtNGFC_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 *ccrtngfc_user.h* include file that is supplied with the driver.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Get_Mapped_Config_Ptr (void                *Handle,
                               ccrtngfc_config_local_data_t  **config_ptr)

```

Description: Get mapped configuration pointer.

```
Input:  void                                *Handle      (Handle pointer)
Output: ccrtnngfc_config_local_data_t      **config_ptr  (config struct ptr)
        -- structure in ccrtnngfc_user.h
Return: _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR             (successful)
        # CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN            (device not open)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
        # CCRTNGFC_LIB_NO_CONFIG_REGION    (config region not present)
*****/
```

2.2.79 ccrtnngfc_Get_Mapped_Driver_Library_Ptr()

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

```
/******
_ccrtnngfc_lib_error_number_t
ccrtnngfc_Get_Mapped_Driver_Library_Ptr (void          *Handle,
                                         ccrtnngfc_driver_library_common_t **driver_lib_ptr)
```

Description: Get mapped Driver/Library structure pointer.

```
Input:  void                                *Handle      (Handle pointer)
Output: ccrtnngfc_driver_library_common_t **driver_lib_ptr (driver_lib struct ptr)
        -- structure in ccrtnngfc_user.h
Return: _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR             (successful)
        # CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN            (device not open)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION     (local region not present)
*****/
```

2.2.80 ccrtnngfc_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 *ccrtnngfc_user.h* include file that is supplied with the driver.

```
/******
_ccrtnngfc_lib_error_number_t
ccrtnngfc_Get_Mapped_Local_Ptr (void          *Handle,
                                ccrtnngfc_local_ctrl_data_t **local_ptr)
```

Description: Get mapped local pointer.

```
Input:  void                                *Handle      (Handle pointer)
Output: ccrtnngfc_local_ctrl_data_t      **local_ptr  (local struct ptr)
        -- structure in ccrtnngfc_user.h
Return: _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR             (successful)
        # CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN            (device not open)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION     (local region not present)
*****/
```


2.2.81 `ccrtNGFC_Get_Open_File_Descriptor()`

When the library `ccrtNGFC_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.

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtNGFC_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: _ccrtngfc_lib_error_number_t  
        # CCRTNGFC_LIB_NO_ERROR (successful)  
        # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)  
        # CCRTNGFC_LIB_NOT_OPEN (device not open)  
        # CCRTNGFC_LIB_INVALID_ARG (invalid argument)  
*****/
```

2.2.82 `ccrtNGFC_Get_Physical_Memory()`

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

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtNGFC_Get_Physical_Memory (void *Handle,  
                              ccrtngfc_user_phys_mem_t *phys_mem)  
  
Description: Get previously mmaped() physical memory address and size  
  
Input:  void *Handle (Handle pointer)  
        ccrtngfc_user_phys_mem_t *phys_mem (mem struct pointer)  
        void *mmaped_user_mem_ptr (mmaped user virtual memory)  
        uint entry_num_in_tran_table (entry number in translation table)  
Output: ccrtngfc_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: _ccrtngfc_lib_error_number_t  
        # CCRTNGFC_LIB_NO_ERROR (successful)  
        # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)  
        # CCRTNGFC_LIB_NOT_OPEN (device not open)  
        # CCRTNGFC_LIB_INVALID_ARG (invalid argument)  
        # CCRTNGFC_LIB_IOCTL_FAILED (driver ioctl call failed)  
*****/
```

2.2.83 ccrtNGFC_Get_PowerModule_Info()

Get useful Power Module Information.

```

/*****
ccrtNGFC_Get_PowerModule_Info()

Description: Get Power Module Information

Input: void *Handle (Handle pointer)
        ccrtngfc_power_module_info_t *PM_Info (power module info)
        _ccrtngfc_fpga_pm_command_t Command; // input
        # CCRTNGFC_PM_CMD_INPUT_SUPPLY_VOLTAGE
        # CCRTNGFC_PM_CMD_OUTPUT_VOLTAGE
        # CCRTNGFC_PM_CMD_OUTPUT_CURRENT
        # CCRTNGFC_PM_CMD_OUTPUT_TEMPERATURE
        # CCRTNGFC_PM_CMD_DIE_TEMPERATURE
        # CCRTNGFC_PM_CMD_PWM_FREQUENCY
        # CCRTNGFC_PM_CMD_OUTPUT_POWER
        # CCRTNGFC_PM_CMD_PEAK_OUTPUT_CURRENT
        _ccrtngfc_fpga_pm_module_select_t ModuleSelect; // input
        # CCRTNGFC_FPGA_PM_MODULE_0
        # CCRTNGFC_FPGA_PM_MODULE_1
        _ccrtngfc_fpga_pm_xfer_size_t XferSize; // input
        # CCRTNGFC_FPGA_PM_WORD_TRANSFER
        # CCRTNGFC_FPGA_PM_BYTE_TRANSFER
Output: ccrtngfc_power_module_info_t *PM_Info (power module info)
        uint AlertStatus; // output
        uint RawValue; // output
        double FormattedValue; // output
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR (successful)
        # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
        # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN (device not open)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTNGFC_LIB_FPGA_PM_BUSY (power module busy)
        # CCRTNGFC_LIB_FPGA_PM_FAILURE (power module failure)
*****/
```

2.2.84 ccrtNGFC_Get_RunCount_UserProcess()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_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: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR (successful)
        # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
*****/
```

2.2.85 ccrtNGFC_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.

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtNGFC_Get_Value (void          *Handle,  
                    CCRTNGFC_CONTROL cmd,  
                    void          *value)
```

Description: Return the value of the specified board register.

```
Input:  void          *Handle          (Handle pointer)  
        CCRTNGFC_CONTROL cmd          (register definition)  
        -- structure in ccrtngfc_lib.h  
Output: void          *value;         (pointer to value)  
Return: _ccrtngfc_lib_error_number_t  
        # CCRTNGFC_LIB_NO_ERROR      (successful)  
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)  
        # CCRTNGFC_LIB_NOT_OPEN      (device not open)  
        # CCRTNGFC_LIB_INVALID_ARG   (invalid argument)  
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
```

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

2.2.86 ccrtNGFC_Hex_To_Fraction()

This call converts a hexadecimal value to a fractional decimal.

```
/******  
double  
ccrtNGFC_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)
```

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

2.2.87 ccrtNGFC_Identify_Board()

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

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtNGFC_Identify_Board (void          *Handle,  
                        _ccrtngfc_identify_t Identify)
```

Description: Identify the board by setting the front LED

```
Input:  void          *Handle          (Handle pointer)  
        _ccrtngfc_identify_t Identify (Identify board settings)  
        # CCRTNGFC_IDENTIFY_OFF  
        # CCRTNGFC_IDENTIFY_ON  
        # Number of seconds to blink  
Output: none  
Return: _ccrtngfc_lib_error_number_t  
        # CCRTNGFC_LIB_NO_ERROR      (successful)
```

```

# CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
# CCRTNGFC_LIB_NO_LOCAL_REGION     (local region not present)
# CCRTNGFC_LIB_NOT_OPEN           (device not open)
# CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
*****/

```

2.2.88 ccrtNGFC_Initialize_Board()

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Initialize_Board (void *Handle)

Description: Initialize the board.

Input:   void          *Handle          (Handle pointer)
Output:  none
Return:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR       (successful)
         # CCRTNGFC_LIB_BAD_HANDLE     (no/bad handler supplied)
         # CCRTNGFC_LIB_NOT_OPEN       (device not open)
         # CCRTNGFC_LIB_IOCTL_FAILED   (driver ioctl call failed)
         # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
*****/

```

2.2.89 ccrtNGFC_IpCore_Get_Info()

This call returns information of all the IP Core modules available.

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. Note that the IpCoreCodes below can be renamed from their default names to the new names as and when new Ip Codes are implemented.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_IpCore_Get_Info (void          *Handle,
                          ccrtnfgc_ipcore_info_t *ip_info)

Description: Get IP Core Information

Input:   void          *Handle (Handle pointer)
Output:  ccrtnfgc_ipcore_info_t *ip_info (pointer to Ip Core Struct)
         u_int32_t          NumberAdvancedIPCores;
         _ccrtngfc_ipcore_info_t IpInfo[CCRTNGFC_TOTAL_NUMBER_OF_IP_CORES]
         _ccrtngfc_ipcore_t IpCore;
         u_int32_t IpCoreCode
         # CCRTNGFC_IPCODE_0
         # CCRTNGFC_IPCODE_1
         # CCRTNGFC_IPCODE_2
         # CCRTNGFC_IPCODE_3
         # CCRTNGFC_IPCODE_4
         # CCRTNGFC_IPCODE_5
         # CCRTNGFC_IPCODE_6
         # CCRTNGFC_IPCODE_7
         # CCRTNGFC_IPCODE_8
         # CCRTNGFC_IPCODE_9
         .
         .

```

```

        .
        # CCRTNGFC_IPCODE_140
        # CCRTNGFC_IPCODE_141
        # CCRTNGFC_IPCODE_142
        # CCRTNGFC_IPCODE_143
        # CCRTNGFC_IPCODE_144
        # CCRTNGFC_IPCODE_145
        # CCRTNGFC_IPCODE_146
        # CCRTNGFC_IPCODE_147
    union {
        u_int32_t                IpCoreRevision
        ccrtnngfc_ipcore_revision_t IpCRev
    }
    u_int32_t    IpCoreOffset
    u_int32_t    IpCoreInformation
    void        *IpCoreMappedPtr
    char        IpCoreName[CCRTNGFC_IPCORE_NAME_SIZE]
    char        IpCoreDescription[CCRTNGFC_IPCORE_DESC_SIZE]
Return:  _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR            (successful)
        # CCRTNGFC_LIB_BAD_HANDLE         (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION   (local region not present)
    *****/

```

2.2.90 ccrtnngfc_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. Note that the IpCoreCodes below can be renamed from their default names to the new names as and when new Ip Codes are implemented.

```

/*****
_ccrtnngfc_lib_error_number_t
ccrtnngfc_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
        # CCRTNGFC_IPCODE_0
        # CCRTNGFC_IPCODE_1
        # CCRTNGFC_IPCODE_2
        # CCRTNGFC_IPCODE_3
        # CCRTNGFC_IPCODE_4
        # CCRTNGFC_IPCODE_5
        # CCRTNGFC_IPCODE_6
        # CCRTNGFC_IPCODE_7
        # CCRTNGFC_IPCODE_8
        # CCRTNGFC_IPCODE_9
        .
        .
        # CCRTNGFC_IPCODE_140
        # CCRTNGFC_IPCODE_141
        # CCRTNGFC_IPCODE_142
        # CCRTNGFC_IPCODE_143

```

```

        # CCRTNGFC_IPCODE_144
        # CCRTNGFC_IPCODE_145
        # CCRTNGFC_IPCODE_146
        # CCRTNGFC_IPCODE_147
Output: void                **ipcore_ptr (ipcore ptr)
        -- structure in ccrtnngfc_user.h
Return: _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR                (successful)
        # CCRTNGFC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN                (device not open)
        # CCRTNGFC_LIB_INVALID_ARG             (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION         (local region not present)
        # CCRTNGFC_LIB_NO_RESOURCE             (Ip core not present)
*****/

```

2.2.91 ccrtnngfc_LDIO_Activate()

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

Currently, there are only two LDIO modules available and reside on the FPGA mother board. DIO Module 0, and LIO Module 1. In the future we could have additional DIO and LIO modules in the form of add-on daughter cards. This call Enables and Disables ALL available LDIO modules. There is no ability to enable/disable an individual module.

If *current_state* is set to NULL, then no current state information is returned.

```

/*****
_ccrtnngfc_lib_error_number_t
ccrtnngfc_LDIO_Activate (void                *Handle,
                        _ccrtnngfc_ldio_module_enable_t activate,
                        _ccrtnngfc_ldio_module_enable_t *current_state)

```

Description: Activate/DeActivate LVDS and Digital I/O module

```

Input: void                *Handle                (Handle pointer)
        _ccrtnngfc_ldio_module_enable_t         activate                (activate/deactivate)
        # CCRTNGFC_LDIO_MODULE_DISABLE
        # CCRTNGFC_LDIO_MODULE_ENABLE
        # CCRTNGFC_LDIO_MODULE_RESET             (disable followed by enable)
        # CCRTNGFC_LDIO_MODULE_DO_NOT_CHANGE
Output: _ccrtnngfc_ldio_module_enable_t         *current_state (active/deactive)
        # CCRTNGFC_LDIO_MODULE_DISABLE
        # CCRTNGFC_LDIO_MODULE_ENABLE
Return: _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR                (successful)
        # CCRTNGFC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN                (device not open)
        # CCRTNGFC_LIB_INVALID_ARG             (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION         (local region not present)
*****/

```

2.2.92 ccrtNGFC_LDIO_Get_Channels_Polarity()

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

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

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

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

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtngfc_LDIO_Get_Channels_Polarity(void          *Handle,  
                                     ccrtngfc_ldio_modules_t  LDIO_ChannelsPolarity,  
                                     ccrtngfc_ldio_modules_t  ChannelSelectMask)
```

Description: Get Channels Polarity

```
Input:  void          *Handle          (handle pointer)  
        ccrtngfc_ldio_modules_t      ChannelSelectMask  (custom channel selection)  
        # NULL                        (select all channels)  
        # u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]  
          # CCRTNGFC_LDIO_CHANNEL_MASK_0  
          # CCRTNGFC_LDIO_CHANNEL_MASK_1  
          # CCRTNGFC_LDIO_CHANNEL_MASK_2  
          # CCRTNGFC_LDIO_CHANNEL_MASK_3  
          # CCRTNGFC_LDIO_CHANNEL_MASK_4  
          # CCRTNGFC_LDIO_CHANNEL_MASK_5  
          # CCRTNGFC_LDIO_CHANNEL_MASK_6  
          # CCRTNGFC_LDIO_CHANNEL_MASK_7  
          # CCRTNGFC_LDIO_CHANNEL_MASK_8  
          # CCRTNGFC_LDIO_CHANNEL_MASK_9  
          # CCRTNGFC_LDIO_CHANNEL_MASK_10  
          # CCRTNGFC_LDIO_CHANNEL_MASK_11  
          # CCRTNGFC_LDIO_CHANNEL_MASK_12  
          # CCRTNGFC_LDIO_CHANNEL_MASK_13  
          # CCRTNGFC_LDIO_CHANNEL_MASK_14  
          # CCRTNGFC_LDIO_CHANNEL_MASK_15  
          # CCRTNGFC_LDIO_CHANNEL_MASK_16  
          # CCRTNGFC_LDIO_CHANNEL_MASK_17  
          # CCRTNGFC_LDIO_CHANNEL_MASK_18  
          # CCRTNGFC_LDIO_CHANNEL_MASK_19  
          # CCRTNGFC_LDIO_CHANNEL_MASK_20  
          # CCRTNGFC_LDIO_CHANNEL_MASK_21  
          # CCRTNGFC_LDIO_CHANNEL_MASK_22  
          # CCRTNGFC_LDIO_CHANNEL_MASK_23  
          # CCRTNGFC_LDIO_CHANNEL_MASK_24  
          # CCRTNGFC_LDIO_CHANNEL_MASK_25  
          # CCRTNGFC_LDIO_CHANNEL_MASK_26  
          # CCRTNGFC_LDIO_CHANNEL_MASK_27  
          # CCRTNGFC_LDIO_CHANNEL_MASK_28  
          # CCRTNGFC_LDIO_CHANNEL_MASK_29  
          # CCRTNGFC_LDIO_CHANNEL_MASK_30
```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Output: ccrtnngfc_ldio_modules_t LDIO_ChannelsPolarity (channels polarity registers)
- CCRTNGFC_LDIO_INPUT_LOW_TRUE = (0) // input direction
- CCRTNGFC_LDIO_INPUT_HIGH_TRUE = (1) // input direction
- CCRTNGFC_LDIO_OUTPUT_LOW = (0) // output direction
- CCRTNGFC_LDIO_OUTPUT_HIGH = (1) // output direction
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Return: _ccrtnngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)

```



```
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/
```

2.2.93 ccrtNGFC_LDIO_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 LVDS and DIO channels. The *ChannelSelectMask* is used to retrieve edge sense settings for selected channels. A value of *CCRTNGFC_LDIO_COS_FALLING_EDGE* or '0' represents sensing of falling edge of input signal while a value of *CCRTNGFC_LDIO_COS_RISING_EDGE* or '1' represents sensing of rising edge of input signal.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```
/******
_ccrtngfc_lib_error_number_t
ccrtNGFC_LDIO_Get_COS_Channels_Edge_Sense(void          *Handle,
                                             ccrtngfc_ldio_modules_t  LDIO_COS_ChannelsEdgeSense,
                                             ccrtngfc_ldio_modules_t  ChannelSelectMask)
```

Description: Get COS Channels Edge Sense

```
Input:  void          *Handle          (handle pointer)
        ccrtngfc_ldio_modules_t      ChannelSelectMask  (custom channel selection)
        # NULL                        (select all channels)
        # u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
          # CCRTNGFC_LDIO_CHANNEL_MASK_0
          # CCRTNGFC_LDIO_CHANNEL_MASK_1
          # CCRTNGFC_LDIO_CHANNEL_MASK_2
          # CCRTNGFC_LDIO_CHANNEL_MASK_3
          # CCRTNGFC_LDIO_CHANNEL_MASK_4
          # CCRTNGFC_LDIO_CHANNEL_MASK_5
          # CCRTNGFC_LDIO_CHANNEL_MASK_6
          # CCRTNGFC_LDIO_CHANNEL_MASK_7
          # CCRTNGFC_LDIO_CHANNEL_MASK_8
          # CCRTNGFC_LDIO_CHANNEL_MASK_9
          # CCRTNGFC_LDIO_CHANNEL_MASK_10
          # CCRTNGFC_LDIO_CHANNEL_MASK_11
          # CCRTNGFC_LDIO_CHANNEL_MASK_12
          # CCRTNGFC_LDIO_CHANNEL_MASK_13
          # CCRTNGFC_LDIO_CHANNEL_MASK_14
          # CCRTNGFC_LDIO_CHANNEL_MASK_15
          # CCRTNGFC_LDIO_CHANNEL_MASK_16
          # CCRTNGFC_LDIO_CHANNEL_MASK_17
          # CCRTNGFC_LDIO_CHANNEL_MASK_18
          # CCRTNGFC_LDIO_CHANNEL_MASK_19
          # CCRTNGFC_LDIO_CHANNEL_MASK_20
          # CCRTNGFC_LDIO_CHANNEL_MASK_21
          # CCRTNGFC_LDIO_CHANNEL_MASK_22
          # CCRTNGFC_LDIO_CHANNEL_MASK_23
          # CCRTNGFC_LDIO_CHANNEL_MASK_24
          # CCRTNGFC_LDIO_CHANNEL_MASK_25
          # CCRTNGFC_LDIO_CHANNEL_MASK_26
          # CCRTNGFC_LDIO_CHANNEL_MASK_27
          # CCRTNGFC_LDIO_CHANNEL_MASK_28
          # CCRTNGFC_LDIO_CHANNEL_MASK_29
          # CCRTNGFC_LDIO_CHANNEL_MASK_30
```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Output: ccrtnngfc_ldio_modules_t LDIO_COS_ChannelsEdgeSense (COS channels edge sense
registers)
- CCRTNGFC_LDIO_COS_FALLING_EDGE = (0)
- CCRTNGFC_LDIO_COS_RISING_EDGE = (1)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Return: _ccrtnngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)

```

*****/

2.2.94 ccrtNGFC_LDIO_Get_COS_Channels_Enable()

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

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

/*****

```
_ccrtngfc_lib_error_number_t
ccrtngfc_LDIO_Get_COS_Channels_Enable(void          *Handle,
                                         ccrtngfc_ldio_modules_t  LDIO_COS_ChannelsEnable,
                                         ccrtngfc_ldio_modules_t  ChannelSelectMask)
```

Description: Get COS Channels Enable Setting

```
Input: void          *Handle          (handle pointer)
       ccrtngfc_ldio_modules_t  ChannelSelectMask (custom channel selection)
       # NULL                  (select all channels)
       # u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
       # CCRTNGFC_LDIO_CHANNEL_MASK_0
       # CCRTNGFC_LDIO_CHANNEL_MASK_1
       # CCRTNGFC_LDIO_CHANNEL_MASK_2
       # CCRTNGFC_LDIO_CHANNEL_MASK_3
       # CCRTNGFC_LDIO_CHANNEL_MASK_4
       # CCRTNGFC_LDIO_CHANNEL_MASK_5
       # CCRTNGFC_LDIO_CHANNEL_MASK_6
       # CCRTNGFC_LDIO_CHANNEL_MASK_7
       # CCRTNGFC_LDIO_CHANNEL_MASK_8
       # CCRTNGFC_LDIO_CHANNEL_MASK_9
       # CCRTNGFC_LDIO_CHANNEL_MASK_10
       # CCRTNGFC_LDIO_CHANNEL_MASK_11
       # CCRTNGFC_LDIO_CHANNEL_MASK_12
       # CCRTNGFC_LDIO_CHANNEL_MASK_13
       # CCRTNGFC_LDIO_CHANNEL_MASK_14
       # CCRTNGFC_LDIO_CHANNEL_MASK_15
       # CCRTNGFC_LDIO_CHANNEL_MASK_16
       # CCRTNGFC_LDIO_CHANNEL_MASK_17
       # CCRTNGFC_LDIO_CHANNEL_MASK_18
       # CCRTNGFC_LDIO_CHANNEL_MASK_19
       # CCRTNGFC_LDIO_CHANNEL_MASK_20
       # CCRTNGFC_LDIO_CHANNEL_MASK_21
       # CCRTNGFC_LDIO_CHANNEL_MASK_22
       # CCRTNGFC_LDIO_CHANNEL_MASK_23
       # CCRTNGFC_LDIO_CHANNEL_MASK_24
       # CCRTNGFC_LDIO_CHANNEL_MASK_25
       # CCRTNGFC_LDIO_CHANNEL_MASK_26
       # CCRTNGFC_LDIO_CHANNEL_MASK_27
       # CCRTNGFC_LDIO_CHANNEL_MASK_28
       # CCRTNGFC_LDIO_CHANNEL_MASK_29
       # CCRTNGFC_LDIO_CHANNEL_MASK_30
       # CCRTNGFC_LDIO_CHANNEL_MASK_31
       # CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
```

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.

```

# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Output: ccrtnngfc_ldio_modules_t LDIO_COS_ChannelsEnable (COS channels Enable registers)
        - CCRTNGFC_LDIO_COS_IGNORE = (0)
        - CCRTNGFC_LDIO_COS_ENABLE = (1)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Return: _ccrtnngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.95 ccrtNGFC_LDIO_Get_COS_Channels_Mode()

This call returns to the user the settings for the change-of-state mode registers all the LVDS and DIO channels. The *ChannelSelectMask* is used to retrieve mode settings for selected channels. A value of *CCRTNGFC_LDIO_COS_ANY_TRANSITION* or '0' detects change-of-state on any edge transition while a value of *CCRTNGFC_LDIO_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.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtngfc_LDIO_Get_COS_Channels_Mode(void          *Handle,  
                                     ccrtngfc_ldio_modules_t  LDIO_COS_ChannelsMode,  
                                     ccrtngfc_ldio_modules_t  ChannelSelectMask)
```

Description: Get COS Channels Mode

```
Input:  void          *Handle          (handle pointer)  
        ccrtngfc_ldio_modules_t      ChannelSelectMask (custom channel selection)  
        # NULL                        (select all channels)  
        # u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]  
          # CCRTNGFC_LDIO_CHANNEL_MASK_0  
          # CCRTNGFC_LDIO_CHANNEL_MASK_1  
          # CCRTNGFC_LDIO_CHANNEL_MASK_2  
          # CCRTNGFC_LDIO_CHANNEL_MASK_3  
          # CCRTNGFC_LDIO_CHANNEL_MASK_4  
          # CCRTNGFC_LDIO_CHANNEL_MASK_5  
          # CCRTNGFC_LDIO_CHANNEL_MASK_6  
          # CCRTNGFC_LDIO_CHANNEL_MASK_7  
          # CCRTNGFC_LDIO_CHANNEL_MASK_8  
          # CCRTNGFC_LDIO_CHANNEL_MASK_9  
          # CCRTNGFC_LDIO_CHANNEL_MASK_10  
          # CCRTNGFC_LDIO_CHANNEL_MASK_11  
          # CCRTNGFC_LDIO_CHANNEL_MASK_12  
          # CCRTNGFC_LDIO_CHANNEL_MASK_13  
          # CCRTNGFC_LDIO_CHANNEL_MASK_14  
          # CCRTNGFC_LDIO_CHANNEL_MASK_15  
          # CCRTNGFC_LDIO_CHANNEL_MASK_16  
          # CCRTNGFC_LDIO_CHANNEL_MASK_17  
          # CCRTNGFC_LDIO_CHANNEL_MASK_18  
          # CCRTNGFC_LDIO_CHANNEL_MASK_19  
          # CCRTNGFC_LDIO_CHANNEL_MASK_20  
          # CCRTNGFC_LDIO_CHANNEL_MASK_21  
          # CCRTNGFC_LDIO_CHANNEL_MASK_22  
          # CCRTNGFC_LDIO_CHANNEL_MASK_23  
          # CCRTNGFC_LDIO_CHANNEL_MASK_24  
          # CCRTNGFC_LDIO_CHANNEL_MASK_25  
          # CCRTNGFC_LDIO_CHANNEL_MASK_26  
          # CCRTNGFC_LDIO_CHANNEL_MASK_27  
          # CCRTNGFC_LDIO_CHANNEL_MASK_28  
          # CCRTNGFC_LDIO_CHANNEL_MASK_29  
          # CCRTNGFC_LDIO_CHANNEL_MASK_30  
          # CCRTNGFC_LDIO_CHANNEL_MASK_31  
          # CCRTNGFC_LDIO_ALL_CHANNELS_MASK  
        CCRTNGFC_LDIO_MAX_MODULES can be one of:  
          # CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
```

```

# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Output: ccrtnngfc_ldio_modules_t LDIO_COS_ChannelsMode (COS channels Mode registers)
- CCRTNGFC_LDIO_COS_ANY_TRANSITION = (0)
- CCRTNGFC_LDIO_COS_RISING_OR_FALLING_TRANSITION = (1)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Return: _ccrtnngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.96 ccrtNGFC_LDIO_Get_COS_Channels_Overflow()

This call returns to the user the state of the change-of-state overflow registers for all the LVDS and DIO channels. The *ChannelSelectMask* is used to retrieve overflow settings for selected channels. A value of *CCRTNGFC_LDIO_COS_OVERFLOW_DID_NOT_OCCUR* or '0' indicates that no overflow occurred while a value of *CCRTNGFC_LDIO_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 COS status being cleared. Any read of this register will automatically clear the change of state overflow bits that were set in the read data.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtngfc_LDIO_Get_COS_Channels_Overflow(void          *Handle,  
                                           ccrtngfc_ldio_modules_t  LDIO_COS_ChannelsOverflow,  
                                           ccrtngfc_ldio_modules_t  ChannelSelectMask)
```

Description: Get COS Channels Overflow

```
Input:  void          *Handle          (handle pointer)  
        ccrtngfc_ldio_modules_t      ChannelSelectMask  (custom channel selection)  
        # NULL                          (select all channels)  
        # u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]  
          # CCRTNGFC_LDIO_CHANNEL_MASK_0  
          # CCRTNGFC_LDIO_CHANNEL_MASK_1  
          # CCRTNGFC_LDIO_CHANNEL_MASK_2  
          # CCRTNGFC_LDIO_CHANNEL_MASK_3  
          # CCRTNGFC_LDIO_CHANNEL_MASK_4  
          # CCRTNGFC_LDIO_CHANNEL_MASK_5  
          # CCRTNGFC_LDIO_CHANNEL_MASK_6  
          # CCRTNGFC_LDIO_CHANNEL_MASK_7  
          # CCRTNGFC_LDIO_CHANNEL_MASK_8  
          # CCRTNGFC_LDIO_CHANNEL_MASK_9  
          # CCRTNGFC_LDIO_CHANNEL_MASK_10  
          # CCRTNGFC_LDIO_CHANNEL_MASK_11  
          # CCRTNGFC_LDIO_CHANNEL_MASK_12  
          # CCRTNGFC_LDIO_CHANNEL_MASK_13  
          # CCRTNGFC_LDIO_CHANNEL_MASK_14  
          # CCRTNGFC_LDIO_CHANNEL_MASK_15  
          # CCRTNGFC_LDIO_CHANNEL_MASK_16  
          # CCRTNGFC_LDIO_CHANNEL_MASK_17  
          # CCRTNGFC_LDIO_CHANNEL_MASK_18  
          # CCRTNGFC_LDIO_CHANNEL_MASK_19  
          # CCRTNGFC_LDIO_CHANNEL_MASK_20  
          # CCRTNGFC_LDIO_CHANNEL_MASK_21  
          # CCRTNGFC_LDIO_CHANNEL_MASK_22  
          # CCRTNGFC_LDIO_CHANNEL_MASK_23  
          # CCRTNGFC_LDIO_CHANNEL_MASK_24  
          # CCRTNGFC_LDIO_CHANNEL_MASK_25  
          # CCRTNGFC_LDIO_CHANNEL_MASK_26  
          # CCRTNGFC_LDIO_CHANNEL_MASK_27  
          # CCRTNGFC_LDIO_CHANNEL_MASK_28  
          # CCRTNGFC_LDIO_CHANNEL_MASK_29  
          # CCRTNGFC_LDIO_CHANNEL_MASK_30  
          # CCRTNGFC_LDIO_CHANNEL_MASK_31
```

```

# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Output: ccrtnngfc_ldio_modules_t LDIO_COS_ChannelsOverflow (COS channels overflow registers)
        - CCRTNGFC_LDIO_COS_OVERFLOW_DID_NOT_OCCUR = (0)
        - CCRTNGFC_LDIO_COS_OVERFLOW_OCCURRED = (1)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Return: _ccrtnngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```


2.2.97 ccrtNGFC_LDIO_Get_COS_Channels_Status()

This call returns to the user the state of the change-of-state status registers for all the LVDS and DIO channels. The *ChannelSelectMask* is used to retrieve status settings for selected channels. A value of *CCRTNGFC_LDIO_COS_DID_NOT_OCCUR* or '0' indicates that no change-of-state occurred while a value of *CCRTNGFC_LDIO_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. Any read of this register will automatically clear the change of state status bits that were set in the read data.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtngfc_LDIO_Get_COS_Channels_Status(void                *Handle,  
                                       ccrtngfc_ldio_modules_t LDIO_COS_ChannelsStatus,  
                                       ccrtngfc_ldio_modules_t ChannelSelectMask)
```

Description: Get COS Channels Status

```
Input:  void                *Handle                (handle pointer)  
        ccrtngfc_ldio_modules_t  ChannelSelectMask (custom channel selection)  
        # NULL                (select all channels)  
        # u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]  
          # CCRTNGFC_LDIO_CHANNEL_MASK_0  
          # CCRTNGFC_LDIO_CHANNEL_MASK_1  
          # CCRTNGFC_LDIO_CHANNEL_MASK_2  
          # CCRTNGFC_LDIO_CHANNEL_MASK_3  
          # CCRTNGFC_LDIO_CHANNEL_MASK_4  
          # CCRTNGFC_LDIO_CHANNEL_MASK_5  
          # CCRTNGFC_LDIO_CHANNEL_MASK_6  
          # CCRTNGFC_LDIO_CHANNEL_MASK_7  
          # CCRTNGFC_LDIO_CHANNEL_MASK_8  
          # CCRTNGFC_LDIO_CHANNEL_MASK_9  
          # CCRTNGFC_LDIO_CHANNEL_MASK_10  
          # CCRTNGFC_LDIO_CHANNEL_MASK_11  
          # CCRTNGFC_LDIO_CHANNEL_MASK_12  
          # CCRTNGFC_LDIO_CHANNEL_MASK_13  
          # CCRTNGFC_LDIO_CHANNEL_MASK_14  
          # CCRTNGFC_LDIO_CHANNEL_MASK_15  
          # CCRTNGFC_LDIO_CHANNEL_MASK_16  
          # CCRTNGFC_LDIO_CHANNEL_MASK_17  
          # CCRTNGFC_LDIO_CHANNEL_MASK_18  
          # CCRTNGFC_LDIO_CHANNEL_MASK_19  
          # CCRTNGFC_LDIO_CHANNEL_MASK_20  
          # CCRTNGFC_LDIO_CHANNEL_MASK_21  
          # CCRTNGFC_LDIO_CHANNEL_MASK_22  
          # CCRTNGFC_LDIO_CHANNEL_MASK_23  
          # CCRTNGFC_LDIO_CHANNEL_MASK_24  
          # CCRTNGFC_LDIO_CHANNEL_MASK_25  
          # CCRTNGFC_LDIO_CHANNEL_MASK_26  
          # CCRTNGFC_LDIO_CHANNEL_MASK_27  
          # CCRTNGFC_LDIO_CHANNEL_MASK_28  
          # CCRTNGFC_LDIO_CHANNEL_MASK_29  
          # CCRTNGFC_LDIO_CHANNEL_MASK_30  
          # CCRTNGFC_LDIO_CHANNEL_MASK_31
```

```

# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Output: ccrtnngfc_ldio_modules_t LDIO_COS_ChannelsStatus (COS channels Status registers)
        - CCRTNGFC_LDIO_COS_DID_NOT_OCCUR = (0)
        - CCRTNGFC_LDIO_COS_OCCURRED = (1)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Return: _ccrtnngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.98 ccrtNGFC_LDIO_Get_Input_Channels_Filter()

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

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtngfc_LDIO_Get_Input_Channels_Filter(void                *Handle,  
                                         ccrtngfc_ldio_modules_t  LDIO_InputChannelsFilter,  
                                         ccrtngfc_ldio_modules_t  ChannelSelectMask)
```

Description: Get Input Channels Filter

```
Input:  void                *Handle                (handle pointer)  
        ccrtngfc_ldio_modules_t  ChannelSelectMask (custom channel selection)  
        # NULL                (select all channels)  
        # u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]  
          # CCRTNGFC_LDIO_CHANNEL_MASK_0  
          # CCRTNGFC_LDIO_CHANNEL_MASK_1  
          # CCRTNGFC_LDIO_CHANNEL_MASK_2  
          # CCRTNGFC_LDIO_CHANNEL_MASK_3  
          # CCRTNGFC_LDIO_CHANNEL_MASK_4  
          # CCRTNGFC_LDIO_CHANNEL_MASK_5  
          # CCRTNGFC_LDIO_CHANNEL_MASK_6  
          # CCRTNGFC_LDIO_CHANNEL_MASK_7  
          # CCRTNGFC_LDIO_CHANNEL_MASK_8  
          # CCRTNGFC_LDIO_CHANNEL_MASK_9  
          # CCRTNGFC_LDIO_CHANNEL_MASK_10  
          # CCRTNGFC_LDIO_CHANNEL_MASK_11  
          # CCRTNGFC_LDIO_CHANNEL_MASK_12  
          # CCRTNGFC_LDIO_CHANNEL_MASK_13  
          # CCRTNGFC_LDIO_CHANNEL_MASK_14  
          # CCRTNGFC_LDIO_CHANNEL_MASK_15  
          # CCRTNGFC_LDIO_CHANNEL_MASK_16  
          # CCRTNGFC_LDIO_CHANNEL_MASK_17  
          # CCRTNGFC_LDIO_CHANNEL_MASK_18  
          # CCRTNGFC_LDIO_CHANNEL_MASK_19  
          # CCRTNGFC_LDIO_CHANNEL_MASK_20  
          # CCRTNGFC_LDIO_CHANNEL_MASK_21  
          # CCRTNGFC_LDIO_CHANNEL_MASK_22  
          # CCRTNGFC_LDIO_CHANNEL_MASK_23  
          # CCRTNGFC_LDIO_CHANNEL_MASK_24  
          # CCRTNGFC_LDIO_CHANNEL_MASK_25  
          # CCRTNGFC_LDIO_CHANNEL_MASK_26  
          # CCRTNGFC_LDIO_CHANNEL_MASK_27  
          # CCRTNGFC_LDIO_CHANNEL_MASK_28  
          # CCRTNGFC_LDIO_CHANNEL_MASK_29  
          # CCRTNGFC_LDIO_CHANNEL_MASK_30  
          # CCRTNGFC_LDIO_CHANNEL_MASK_31  
          # CCRTNGFC_LDIO_ALL_CHANNELS_MASK  
        CCRTNGFC_LDIO_MAX_MODULES can be one of:  
          # CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
```

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.

```

# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Output: ccrtnngfc_ldio_modules_t LDIO_InputChannelsFilter (input channel registers)
        - CCRTNGFC_LDIO_INPUT_FILTER_DISABLED = (0)
        - CCRTNGFC_LDIO_INPUT_FILTER_ENABLED = (1)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Return: _ccrtnngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.99 ccrtNGFC_LDIO_Get_Input_Snapshot()

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

The module selection for the DIO located on the mother-board is CCRTNGFC_MAIN_DIO_MODULE_0 and for the LIO located on the mother-board is CCRTNGFC_MAIN_LIO_MODULE_1. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the CCRTNGFC_DC_DIO_MODULE_2 and/or CCRTNGFC_DC_DIO_MODULE_3 option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

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

```
  _ccrtngfc_lib_error_number_t
  ccrtNGFC_LDIO_Get_Input_Snapshot(void          *Handle,
                                     _ccrtngfc_ldio_input_snapshot_t *ldio_snapshot)
```

Description: Get LDIO Input Snapshot

```
Input:  void          *Handle          (handle pointer)
Output: _ccrtngfc_ldio_input_snapshot_t *ldio_snapshot (LVDS and Digital I/O snapshot)
        # CCRTNGFC_LDIO_INPUT_OPERATION_CONTINUOUS
        # CCRTNGFC_LDIO_INPUT_OPERATION_SNAPSHOT
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION  (local region not present)
        # CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (DIO is not active)
```

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

2.2.100 ccrtNGFC_LDIO_Get_Output_Sync()

This call returns the current state of the output sync flag for all the LVDS and DIO modules. 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.

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

```
  _ccrtngfc_lib_error_number_t
  ccrtNGFC_LDIO_Get_Output_Sync(void          *Handle,
                                    _ccrtngfc_ldio_output_sync_t *ldio_sync)
```

Description: Get LVDS and Digital I/O Output Sync

```
Input:  void          *Handle          (handle pointer)
Output: _ccrtngfc_ldio_output_sync_t *ldio_sync (LVDS and Digital I/O sync)
        # CCRTNGFC_LDIO_OUTPUT_OPERATION_CONTINUOUS = (0)
        # CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC      = (1)
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION  (local region not present)
        # CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
```

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

2.2.101 ccrtNGFC_LDIO_Read_Input_Channel_Registers()

This call reads the contents of the input channel registers for all the LVDS and DIO modules 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 various LDIO modules. For performance improvements with *this* operational mode, it is recommended that the user sets the continuous option using the *ccrtNGFC_LDIO_Set_Input_Snapshot()* call once and then supply *CCRTNGFC_LDIO_INPUT_OPERATION_DO_NOT_CHANGE* to this call for more reads. In this way, an additional register access will not occur everytime this call is issued.

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

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

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

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.

In the case of Digital I/O module, 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).

In the case of LVDS I/O module, when the direction for channels are set to output, then reading the channels input registers will result in invalid channel information being returned as the only way to read back (loopback) the channel information is to use the LIO Test Mode as specified with the *ccrtNGFC_LIO_Set_Ports_Direction()* call. If the user wishes to read back (loopback) the channel information for the LVDS I/O module, then they will need to use the specific *ccrtNGFC_LIO_Read_Output_Loopbacked_Channels()* call as that is specifically written for this purpose.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtNGFC_LDIO_Read_Input_Channel_Registers(void                *Handle,  
                                             _ccrtngfc_ldio_input_snapshot_t ldio_snapshot,  
                                             ccrtngfc_bool                skip_ldio_disabled_check,  
                                             ccrtngfc_ldio_modules_t      LDIO_InputChannels,  
                                             ccrtngfc_ldio_modules_t      ChannelSelectMask)
```

Description: Read LDIO Input Channel Registers

```
Input:  void                *Handle                (handle pointer)  
        _ccrtngfc_ldio_input_snapshot_t ldio_snapshot    (ldio_snapshot operation)  
        # CCRTNGFC_LDIO_INPUT_OPERATION_CONTINUOUS
```

```

# CCRTNGFC_LDIO_INPUT_OPERATION_SNAPSHOT
# CCRTNGFC_LDIO_INPUT_OPERATION_DO_NOT_CHANGE
ccrtngfc_bool skip_ldio_disabled_check (skip LVDS and Digital
I/O disabled check)

# CCRTNGFC_TRUE
# CCRTNGFC_FALSE
ccrtngfc_ldio_modules_t ChannelSelectMask (custom channel selection)
# NULL (select all channels)
# u_int32_t ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Output: ccrtngfc_ldio_modules_t LDIO_InputChannels (input channel
registers)
# u_int32_t ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.102 ccrtNGFC_LDIO_Read_Output_Channel_Registers()

This call reads the contents of the output channel registers for all the LVDS and DIO modules and returns to the user. This simply represents the contents of the last write to the output registers.

The module selection for the DIO located on the mother-board is CCRTNGFC_MAIN_DIO_MODULE_0 and for the LIO located on the mother-board is CCRTNGFC_MAIN_LIO_MODULE_1. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the CCRTNGFC_DC_DIO_MODULE_2 and/or CCRTNGFC_DC_DIO_MODULE_3 option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_LDIO_Read_Output_Channel_Registers(void *Handle,
                                              ccrtngfc_ldio_modules_t LDIO_OutputChannels,
                                              ccrtngfc_ldio_modules_t ChannelSelectMask)

```

Description: Read LDIO Output Channel Registers

```

Input: void *Handle (handle pointer)
       ccrtngfc_ldio_modules_t ChannelSelectMask (custom channel selection)

```

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.


```

# NULL (select all channels)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Output: ccrtnngfc_ldio_modules_t LDIO_OutputChannels (output channel registers)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.103 ccrtNGFC_LDIO_Set_Channels_Polarity()

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

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

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

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_LDIO_Set_Channels_Polarity(void *Handle,
                                     ccrtngfc_ldio_modules_t LDIO_ChannelsPolarity,
                                     ccrtngfc_ldio_modules_t ChannelSelectMask)

```

Description: Set Channels Polarity

```

Input: void *Handle (handle pointer)
        ccrtngfc_ldio_modules_t LDIO_ChannelsPolarity (channels polarity registers)
        - CCRTNGFC_LDIO_INPUT_LOW_TRUE = (0) // input direction

```

```

- CCRTNGFC_LDIO_INPUT_HIGH_TRUE = (1) // input direction
- CCRTNGFC_LDIO_OUTPUT_LOW      = (0) // output direction
- CCRTNGFC_LDIO_OUTPUT_HIGH    = (1) // output direction
# u_int32_t  ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
ccrtnngfc_ldio_modules_t ChannelSelectMask (custom channel selection)
# NULL (select all channels)
# u_int32_t  ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.104 ccrtNGFC_LDIO_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 of all the LVDS and DIO modules. The *ChannelSelectMask* is used to set the edge sense settings for selected channels. A value of *CCRTNGFC_LDIO_COS_FALLING_EDGE* or '0' represents sensing of falling edge of input signal while a value of *CCRTNGFC_LDIO_COS_RISING_EDGE* or '1' represents sensing of rising edge of input signal.

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

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_LDIO_Set_COS_Channels_Edge_Sense(void *Handle,
                                           ccrtngfc_ldio_modules_t LDIO_COS_ChannelsEdgeSense,
                                           ccrtngfc_ldio_modules_t ChannelSelectMask)

```

Description: Set COS Channels Edge Sense

```

Input: void *Handle (handle pointer)
       ccrtnngfc_ldio_modules_t LDIO_COS_ChannelsEdgeSense (COS channels edge sense registers)
                                     - CCRTNGFC_LDIO_COS_FALLING_EDGE = (0)
                                     - CCRTNGFC_LDIO_COS_RISING_EDGE = (1)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
ccrtnngfc_ldio_modules_t ChannelSelectMask (custom channel selection)
# NULL (select all channels)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.105 ccrtNGFC_LDIO_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 of all the LVDS and DIO modules 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 *CCRTNGFC_LDIO_COS_IGNORE* or '0' ignores change-of-state while a value of *CCRTNGFC_LDIO_COS_ENABLE* or '1' represents enabling change-of-state for the selected channels.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_LDIO_Set_COS_Channels_Enable (void *Handle,
                                        ccrtngfc_ldio_modules_t LDIO_COS_ChannelsEnable,
                                        ccrtngfc_ldio_modules_t ChannelSelectMask)

```

Description: Set COS Channels Enable

```

Input: void *Handle (handle pointer)
       ccrtnngfc_ldio_modules_t LDIO_COS_ChannelsEnable (COS channels enable registers)
                                     - CCRTNGFC_LDIO_COS_IGNORE = (0)
                                     - CCRTNGFC_LDIO_COS_ENABLE = (1)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
ccrtnngfc_ldio_modules_t ChannelSelectMask (custom channel selection)
# NULL (select all channels)
# u_int32_t ccrtnngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.106 ccrtNGFC_LDIO_Set_COS_Channels_Mode()

This call sets the change-of-state mode registers for all the channels of all the LVDS and DIO modules. The *ChannelSelectMask* is used to set the mode settings for selected channels. A value of *CCRTNGFC_LDIO_COS_ANY_TRANSITION* or '0' sets change-of-state on any edge transition while a value of *CCRTNGFC_LDIO_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.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_LDIO_Set_COS_Channels_Mode(void *Handle,
                                     ccrtngfc_ldio_modules_t LDIO_COS_ChannelsMode,
                                     ccrtngfc_ldio_modules_t ChannelSelectMask)

```

Description: Set COS Channels Mode

Input: void *Handle (handle pointer)


```

ccrtngfc_ldio_modules_t      LDIO_COS_ChannelsMode (COS channels mode registers)
                             - CCRTNGFC_LDIO_COS_ANY_TRANSITION      = (0)
                             - CCRTNGFC_LDIO_COS_RISING_OR_FALLING_TRANSITION = (1)
# u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0      // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1     // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2       // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3       // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2       // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3       // optional LIO Daughter Card
ccrtngfc_ldio_modules_t          ChannelSelectMask (custom channel selection)
# NULL                            (select all channels)
# u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
/*****/

```

2.2.107 ccrtNGFC_LDIO_Set_Input_Channels_Filter()

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

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_LDIO_Set_Input_Channels_Filter(void *Handle,
                                         ccrtngfc_ldio_modules_t LDIO_InputChannelsFilter,
                                         ccrtngfc_ldio_modules_t ChannelSelectMask)

```

Description: Set Input Channels Filter

```

Input: void *Handle (handle pointer)
       ccrtngfc_ldio_modules_t LDIO_InputChannelsFilter (input channel registers)

```

```

- CCRTNGFC_LDIO_INPUT_FILTER_DISABLE = (0)
- CCRTNGFC_LDIO_INPUT_FILTER_ENABLE = (1)
# u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
ccrtngfc_ldio_modules_t ChannelSelectMask (custom channel selection)
# NULL (select all channels)
# u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.108 ccrtNGFC_LDIO_Set_Input_Snapshot()

This call allows the user to set the board in snapshot mode where the channel inputs to all the LDIO modules 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 *ccrtNGFC_LDIO_Read_Input_Channel_Registers()* has an option to set it in the call.

If the user wants to collect data in the continuous mode, they should issue this call once with the *CCRTNGFC_LDIO_INPUT_OPERATION_CONTINUOUS* option and then call the read of the input channels with the *CCRTNGFC_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 *CCRTNGFC_LDIO_INPUT_OPERATION_CONTINUOUS* option and then followup with continuous input channel reads using the *ccrtNGFC_LDIO_Read_Input_Channel_Register()* call with the *CCRTNGFC_LDIO_INPUT_OPERATION_DO_NOT_CHANGE* option for *ldio_snapshot*.

Recommended procedure for snapshot mode is to issue continuous input channel reads using the *ccrtNGFC_LDIO_Read_Input_Channel_Register()* call with the *CCRTNGFC_LDIO_INPUT_OPERATION_SNAPSHOT* option for *ldio_snapshot*. In this case, there is really no need to issue this *ccrtNGFC_LDIO_Set_Input_Snapshot()* call.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or

CCRTNGFC_DC_DIO_MODULE_3 option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtngfc_LDIO_Set_Input_Snapshot(void                *Handle,
                                _ccrtngfc_ldio_input_snapshot_t  ldio_snapshot)

```

Description: Set LDIO Input Snapshot

```

Input:  void                *Handle        (handle pointer)
Output: _ccrtngfc_ldio_input_snapshot_t  ldio_snapshot  (LVDS and Digital I/O snapshot)
        # CCRTNGFC_LDIO_INPUT_OPERATION_CONTINUOUS
        # CCRTNGFC_LDIO_INPUT_OPERATION_SNAPSHOT
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR            (successful)
        # CCRTNGFC_LIB_BAD_HANDLE         (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN           (device not open)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION    (local region not present)
        # CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.109 ccrtNGFC_LDIO_Set_Output_Sync()

This call allows the user to set the digital output channels to either *continuous* mode or *sync* mode for all the LVDS and DIO modules. When the board is in *continuous* mode, any update to any of the LDIO modules will be immediately sent to the output lines. There will be no synchronization between the LDIO modules. 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 the LDIO modules will be sent simultaneously to the output lines.

Recommended procedure for *continuous* mode is to issue this call only once with the *CCRTNGFC_LDIO_OUTPUT_OPERATION_CONTINUOUS* option and then followup with continuous output channel writes using the *ccrtNGFC_LDIO_Write_Output_Channel_Register()* call with the *CCRTNGFC_LDIO_INPUT_OPERATION_DO_NOT_CHANGE* option for *ldio_sync*.

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

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtngfc_LDIO_Set_Output_Sync(void                *Handle,
                                _ccrtngfc_ldio_output_sync_t  ldio_sync)

```

Description: Set LDIO Output Sync

```

Input:  void                *Handle        (handle pointer)
        _ccrtngfc_ldio_output_sync_t  ldio_sync  (LVDS and Digital I/O sync)
        # CCRTNGFC_LDIO_OUTPUT_OPERATION_CONTINUOUS
        # CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC
Output: none

```

```

Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.110 ccrtNGFC_LDIO_Write_Output_Channel_Registers()

This call writes the user supplied channel settings to the output channel registers for all the LVDS and DIO modules. 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 LDIO modules. For performance improvements with this operational mode, it is recommended that the user sets the *continuous* option using the *ccrtNGFC_LDIO_Set_Output_Sync()* call once and then supply *CCRTNGFC_LDIO_OUTPUT_OPERATION_DO_NOT_CHANGE* to this call for more writes. In this way, an additional register access will not occur everytime this call is issued.

When the user selects the *sync* operation, they need to issue the *ccrtNGFC_LDIO_Set_Output_Sync()* call once with the *CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC* option, followed by issuing this call with the *CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC* option in *ldio_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 LDIO modules.

The *skip_ldio_disable_check* (when set to *CCRTNGFC_FALSE*) causes the call to test for LDIO being enabled prior to proceeding. If this option is set to *CCRTNGFC_TRUE*, then no validation is performed. If the LDIO 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 LDIO is enabled prior to issuing this call, they can set this option to *CCRTNGFC_TRUE* so that no validation is performed and hence, improve performance.

In order to skip a LDIO module, simply set *ChannelSelectMask* to zero for that module. If *ChannelSelectMask* is set to NULL, then all channels for all LDIO modules are selected.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_LDIO_Write_Output_Channel_Registers(void          *Handle,
                                                _ccrtngfc_ldio_output_sync_t ldio_sync,
                                                ccrtngfc_bool   skip_ldio_disabled_check,
                                                ccrtngfc_ldio_modules_t LDIO_OutputChannels,
                                                ccrtngfc_ldio_modules_t ChannelSelectMask)

```

Description: Write LDIO Output Channel Registers

```

Input:  void          *Handle          (handle pointer)
        _ccrtngfc_ldio_output_sync_t ldio_sync      (ldio_sync operation)
        # CCRTNGFC_LDIO_OUTPUT_OPERATION_CONTINUOUS
        # CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC
        # CCRTNGFC_LDIO_OUTPUT_OPERATION_DO_NOT_CHANGE
        ccrtngfc_bool   skip_ldio_disabled_check  (skip LVDS and Digital

```

```

# CCRTNGFC_TRUE
# CCRTNGFC_FALSE
ccrtngfc_ldio_modules_t LDIO_OutputChannels      (output channel registers)
# u_int32_t   ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0    // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1    // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2      // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3      // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2      // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3      // optional LIO Daughter Card
ccrtngfc_ldio_modules_t ChannelSelectMask      (output channel selection)
# NULL                                          (select all channels)
# u_int32_t   ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.111 ccrtNGFC_LDIO_Write_Output_Channel_Registers_High()

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

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

When the user selects the *sync* operation, they need to issue the *ccrtNGFC_LDIO_Set_Output_Sync()* call once with the *CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC* option, followed by issuing this call with the *CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC* option in *ldio_sync*.

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

The *skip_ldio_disable_check* (when set to *CCRTNGFC_FALSE*) causes the call to test for LDIO being enabled prior to proceeding. If this option is set to *CCRTNGFC_TRUE*, then no validation is performed. If the LDIO 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 LDIO is enabled prior to issuing this call, they can set this option to *CCRTNGFC_TRUE* so that no validation is performed and hence, improve performance.

In order to skip a LDIO module, simply set *LDIO_OutputChannels* to zero for that module.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

/******

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_LDIO_Write_Output_Channel_Registers_High(void                *Handle,
                                                    _ccrtngfc_ldio_output_sync_t ldio_sync,
                                                    ccrtngfc_bool skip_ldio_disabled_check,
                                                    ccrtngfc_ldio_modules_t LDIO_OutputChannels)

```

Description: Write LDIO Output Channel to Registers High

```

Input:  void                *Handle                (handle pointer)
        _ccrtngfc_ldio_output_sync_t ldio_sync      (ldio_sync operation)
        # CCRTNGFC_LDIO_OUTPUT_OPERATION_CONTINUOUS
        # CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC
        # CCRTNGFC_LDIO_OUTPUT_OPERATION_DO_NOT_CHANGE
        ccrtngfc_bool        skip_ldio_disabled_check (skip LVDS and Digital
                                                    I/O disabled check)
        # CCRTNGFC_TRUE
        # CCRTNGFC_FALSE
        ccrtngfc_ldio_modules_t LDIO_OutputChannels (output channel registers)
        # u_int32_t ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
        # CCRTNGFC_LDIO_CHANNEL_MASK_0
        # CCRTNGFC_LDIO_CHANNEL_MASK_1
        # CCRTNGFC_LDIO_CHANNEL_MASK_2
        # CCRTNGFC_LDIO_CHANNEL_MASK_3
        # CCRTNGFC_LDIO_CHANNEL_MASK_4
        # CCRTNGFC_LDIO_CHANNEL_MASK_5
        # CCRTNGFC_LDIO_CHANNEL_MASK_6
        # CCRTNGFC_LDIO_CHANNEL_MASK_7
        # CCRTNGFC_LDIO_CHANNEL_MASK_8
        # CCRTNGFC_LDIO_CHANNEL_MASK_9
        # CCRTNGFC_LDIO_CHANNEL_MASK_10
        # CCRTNGFC_LDIO_CHANNEL_MASK_11
        # CCRTNGFC_LDIO_CHANNEL_MASK_12
        # CCRTNGFC_LDIO_CHANNEL_MASK_13
        # CCRTNGFC_LDIO_CHANNEL_MASK_14
        # CCRTNGFC_LDIO_CHANNEL_MASK_15
        # CCRTNGFC_LDIO_CHANNEL_MASK_16
        # CCRTNGFC_LDIO_CHANNEL_MASK_17
        # CCRTNGFC_LDIO_CHANNEL_MASK_18
        # CCRTNGFC_LDIO_CHANNEL_MASK_19
        # CCRTNGFC_LDIO_CHANNEL_MASK_20
        # CCRTNGFC_LDIO_CHANNEL_MASK_21
        # CCRTNGFC_LDIO_CHANNEL_MASK_22
        # CCRTNGFC_LDIO_CHANNEL_MASK_23
        # CCRTNGFC_LDIO_CHANNEL_MASK_24
        # CCRTNGFC_LDIO_CHANNEL_MASK_25
        # CCRTNGFC_LDIO_CHANNEL_MASK_26
        # CCRTNGFC_LDIO_CHANNEL_MASK_27
        # CCRTNGFC_LDIO_CHANNEL_MASK_28

```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
# CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.112 ccrtNGFC_LDIO_Write_Output_Channel_Registers_Low()

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

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

When the user selects the *sync* operation, they need to issue the *ccrtNGFC_LDIO_Set_Output_Sync()* call once with the *CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC* option, followed by issuing this call with the *CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC* option in *ldio_sync*.

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

The *skip_ldio_disable_check* (when set to *CCRTNGFC_FALSE*) causes the call to test for LDIO being enabled prior to proceeding. If this option is set to *CCRTNGFC_TRUE*, then no validation is performed. If the LDIO 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 LDIO is enabled prior to issuing this call, they can set this option to *CCRTNGFC_TRUE* so that no validation is performed and hence, improve performance.

In order to skip a LDIO module, simply set *LDIO_OutputChannels* to zero for that module.

The module selection for the DIO located on the mother-board is *CCRTNGFC_MAIN_DIO_MODULE_0* and for the LIO located on the mother-board is *CCRTNGFC_MAIN_LIO_MODULE_1*. Additionally, if an optional DIO daughter-card is installed, then it can be selected by the *CCRTNGFC_DC_DIO_MODULE_2* and/or *CCRTNGFC_DC_DIO_MODULE_3* option depending on where the daughter-card is installed. *Currently there is not LIO daughter-card available.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_LDIO_Write_Output_Channel_Registers_Low(void *Handle,
                                                    _ccrtngfc_ldio_output_sync_t ldio_sync,
                                                    ccrtngfc_bool skip_ldio_disabled_check,

```

ccrtngfc_ldio_modules_t LDIO_OutputChannels)

Description: Write LDIO Output Channel to Registers Low

```
Input: void *Handle (handle pointer)
       _ccrtngfc_ldio_output_sync_t ldio_sync (ldio_sync operation)
       # CCRTNGFC_LDIO_OUTPUT_OPERATION_CONTINUOUS
       # CCRTNGFC_LDIO_OUTPUT_OPERATION_SYNC
       # CCRTNGFC_LDIO_OUTPUT_OPERATION_DO_NOT_CHANGE
ccrtngfc_bool skip_ldio_disabled_check (skip LVDS and Digital
                                        I/O disabled check)

       # CCRTNGFC_TRUE
       # CCRTNGFC_FALSE
ccrtngfc_ldio_modules_t LDIO_OutputChannels (output channel registers)
       # u_int32_t ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
       # CCRTNGFC_LDIO_CHANNEL_MASK_0
       # CCRTNGFC_LDIO_CHANNEL_MASK_1
       # CCRTNGFC_LDIO_CHANNEL_MASK_2
       # CCRTNGFC_LDIO_CHANNEL_MASK_3
       # CCRTNGFC_LDIO_CHANNEL_MASK_4
       # CCRTNGFC_LDIO_CHANNEL_MASK_5
       # CCRTNGFC_LDIO_CHANNEL_MASK_6
       # CCRTNGFC_LDIO_CHANNEL_MASK_7
       # CCRTNGFC_LDIO_CHANNEL_MASK_8
       # CCRTNGFC_LDIO_CHANNEL_MASK_9
       # CCRTNGFC_LDIO_CHANNEL_MASK_10
       # CCRTNGFC_LDIO_CHANNEL_MASK_11
       # CCRTNGFC_LDIO_CHANNEL_MASK_12
       # CCRTNGFC_LDIO_CHANNEL_MASK_13
       # CCRTNGFC_LDIO_CHANNEL_MASK_14
       # CCRTNGFC_LDIO_CHANNEL_MASK_15
       # CCRTNGFC_LDIO_CHANNEL_MASK_16
       # CCRTNGFC_LDIO_CHANNEL_MASK_17
       # CCRTNGFC_LDIO_CHANNEL_MASK_18
       # CCRTNGFC_LDIO_CHANNEL_MASK_19
       # CCRTNGFC_LDIO_CHANNEL_MASK_20
       # CCRTNGFC_LDIO_CHANNEL_MASK_21
       # CCRTNGFC_LDIO_CHANNEL_MASK_22
       # CCRTNGFC_LDIO_CHANNEL_MASK_23
       # CCRTNGFC_LDIO_CHANNEL_MASK_24
       # CCRTNGFC_LDIO_CHANNEL_MASK_25
       # CCRTNGFC_LDIO_CHANNEL_MASK_26
       # CCRTNGFC_LDIO_CHANNEL_MASK_27
       # CCRTNGFC_LDIO_CHANNEL_MASK_28
       # CCRTNGFC_LDIO_CHANNEL_MASK_29
       # CCRTNGFC_LDIO_CHANNEL_MASK_30
       # CCRTNGFC_LDIO_CHANNEL_MASK_31
       # CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
       # CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
       # CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
       # CCRTNGFC_DC_DIO_MODULE_2 // optional DIO Daughter Card
       # CCRTNGFC_DC_DIO_MODULE_3 // optional DIO Daughter Card
       # CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
       # CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
       # CCRTNGFC_LIB_NO_ERROR (successful)
       # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
       # CCRTNGFC_LIB_NOT_OPEN (device not open)
       # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
```

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.

```
# CCRTNGFC_LIB_NO_LOCAL_REGION    (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
```

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

2.2.113 ccrtNGFC_LIO_On_Off()

This call is to turn ON or OFF the LVDS DIO Module. Normally, if the LVDS module is not used, it needs to be set to OFF as it generates a lot of heat.

The module selection for the LIO located on the mother-board is CCRTNGFC_MAIN_LIO_MODULE_1. The CCRTNGFC_MAIN_DIO_MODULE_0 is for the DIO module that is located on the mother-board and is therefore invalid for this LIO API. *Currently there is no LIO daughter-card available.*

If the user needs to only inquire about the current state of a LVDS module, they need to call this API with *lio_on_off* set to CCRTNGFC_LIO_MODULE_DO_NOT_CHANGE for the selected module. Its current state is returned in *lio_state*.

If *lio_state* is set to NULL, then no current state information is returned.

The module selection for the DIO located on the mother-board is CCRTNGFC_MAIN_DIO_MODULE_0 which is not valid for this API, and for the LIO located on the mother-board is CCRTNGFC_MAIN_LIO_MODULE_1. *Currently there is not LIO daughter-card available.*

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

```
_ccrtngfc_lib_error_number_t
ccrtngfc_LIO_On_Off (void          *Handle,
                    ccrtngfc_ldio_modules_t  lio_on_off,
                    ccrtngfc_ldio_modules_t  lio_state)
```

Description: Turn On/Off LVDS I/O module

```
Input:  void          *Handle          (Handle pointer)
        ccrtngfc_ldio_modules_t      lio_on_off      (turn on/off LVDS module)
        # CCRTNGFC_LIO_MODULE_OFF
        # CCRTNGFC_LIO_MODULE_ON
        # CCRTNGFC_LIO_MODULE_DO_NOT_CHANGE
        CCRTNGFC_LDIO_MAX_MODULES can be one of:
        # CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
        # CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
        # CCRTNGFC_DC_LIO_MODULE_2   // optional LIO Daughter Card
        # CCRTNGFC_DC_LIO_MODULE_3   // optional LIO Daughter Card
```

```
Output: ccrtngfc_ldio_modules_t      *lio_state      (on/off LVDS Module)
        # CCRTNGFC_LIO_MODULE_OFF
        # CCRTNGFC_LIO_MODULE_ON
        CCRTNGFC_LDIO_MAX_MODULES can be one of:
        # CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
        # CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
        # CCRTNGFC_DC_LIO_MODULE_2   // optional LIO Daughter Card
        # CCRTNGFC_DC_LIO_MODULE_3   // optional LIO Daughter Card
```

```
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN     (device not open)
        # CCRTNGFC_LIB_INVALID_ARG  (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
```

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

2.2.114 ccrtNGFC_LIO_Get_Ports_Direction()

This call returns to the user the direction of the LVDS I/O channels.

The module selection for the LIO located on the mother-board is CCRTNGFC_MAIN_LIO_MODULE_1. The CCRTNGFC_MAIN_DIO_MODULE_0 is for the DIO module that is located on the mother-board and is therefore invalid for this LIO API. *Currently there is no LIO daughter-card available.*

Unlike the Digital I/O ports where each Digital I/O ports is on a per channel basis, the LVDS I/O ports are grouped into 4 channels each. E.g. CCRTNGFC_LIO_PORT_MASK_P0 controls channels 0..3, CCRTNGFC_LIO_PORT_MASK_P1 controls channels 4..7, etc.

The LIO test mode register determines whether a LIO port of LVDS signals are in test mode or not. The test mode is used to turn-around a single LIO port (*four channel group*). It is used in conjunction with only ONE direction port setting at a time to test the LVDS drivers and receivers. Any attempt to turn-around more than one LIO port at a time will be ignored.

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 LVDS I/O lines.

When the direction for channels are set to output, then reading the channels input registers will result in invalid channel information being returned as the only way to read back (loopback) the channel information is to use the LIO Test Mode as specified with the *ccrtNGFC_LIO_Set_Ports_Direction()* call and select only ONE port to read at a time. If the user wishes to read back (loopback) the channel information for the LVDS I/O module, then they will need to use the specific *ccrtNGFC_LIO_Read_Output_Loopbacked_Channels()* call as that is specifically written for this purpose.

The module selection for the DIO located on the mother-board is CCRTNGFC_MAIN_DIO_MODULE_0 which is not valid for this API, and for the LIO located on the mother-board is CCRTNGFC_MAIN_LIO_MODULE_1. *Currently there is not LIO daughter-card available.*

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtngfc_LIO_Get_Ports_Direction(void *Handle,  
                                   ccrtngfc_lio_test_modes_t LIO_TestMode,  
                                   ccrtngfc_lio_ports_t LIO_PortDirection,  
                                   ccrtngfc_lio_ports_t PortSelectionMask)
```

Description: Get LIO Ports Direction Mask

```
Input: void *Handle (handle pointer)  
       ccrtngfc_lio_ports_t PortSelectionMask (port selection mask)  
       # NULL (select all ports)  
       # _ccrtngfc_lio_port_mask_t  
ccrtngfc_lio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]  
   # CCRTNGFC_LIO_PORT_MASK_P0  
   # CCRTNGFC_LIO_PORT_MASK_P1  
   # CCRTNGFC_LIO_PORT_MASK_P2  
   # CCRTNGFC_LIO_PORT_MASK_P3  
   # CCRTNGFC_LIO_PORT_MASK_P4  
   # CCRTNGFC_LIO_PORT_MASK_P5  
   # CCRTNGFC_LIO_PORT_MASK_P6  
   # CCRTNGFC_LIO_PORT_MASK_P7  
   # CCRTNGFC_LIO_ALL_PORTS_MASK  
CCRTNGFC_LDIO_MAX_MODULES can be one of:  
   # CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO  
   # CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO  
   # CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card  
   # CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card
```

```

Output:  ccrtnngfc_ldio_test_modes_t      LIO_TestMode          (LVDS I/O test mode)
        # NULL Pointer                    (skip testmode)
        # _ccrtnngfc_ldio_port_mode_t ccrtnngfc_ldio_test_modes_t[CCRTNGFC_LDIO_MAX_MODULES]
        # CCRTNGFC_LIO_NORMAL_MODE
        # CCRTNGFC_LIO_TEST_MODE
ccrtnngfc_ldio_ports_t      LIO_PortDirection      (port direction)
        - CCRTNGFC_LDIO_DIRECTION_INPUT   = (0)
        - CCRTNGFC_LDIO_DIRECTION_OUTPUT  = (1)
        # _ccrtnngfc_ldio_port_mask_t ccrtnngfc_ldio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
        # CCRTNGFC_LIO_PORT_MASK_P0
        # CCRTNGFC_LIO_PORT_MASK_P1
        # CCRTNGFC_LIO_PORT_MASK_P2
        # CCRTNGFC_LIO_PORT_MASK_P3
        # CCRTNGFC_LIO_PORT_MASK_P4
        # CCRTNGFC_LIO_PORT_MASK_P5
        # CCRTNGFC_LIO_PORT_MASK_P6
        # CCRTNGFC_LIO_PORT_MASK_P7
        # CCRTNGFC_LIO_ALL_PORTS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
        # CCRTNGFC_MAIN_DIO_MODULE_0      // On Main Board - DIO
        # CCRTNGFC_MAIN_LIO_MODULE_1      // On Main Board - LIO
        # CCRTNGFC_DC_LIO_MODULE_2        // optional LIO Daughter Card
        # CCRTNGFC_DC_LIO_MODULE_3        // optional LIO Daughter Card

Return:  _ccrtnngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR            (successful)
        # CCRTNGFC_LIB_BAD_HANDLE         (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN           (device not open)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION    (local region not present)
        # CCRTNGFC_LIB_LIO_IS_NOT_ACTIVE (LIO is not active)
*****/

```

2.2.115 ccrtnngfc_ldio_Read_Output_Loopbacked_Channels()

This call returns to the user the loopbacked channel information for the selected the LVDS I/O modules.

The module selection for the LIO located on the mother-board is CCRTNGFC_MAIN_LIO_MODULE_1. The CCRTNGFC_MAIN_DIO_MODULE_0 is for the DIO module that is located on the mother-board and is therefore invalid for this LIO API. *Currently there is no LIO daughter-card available.*

The LVDS I/O module is unable to read loopback information from channels in the normally way as is done by the Digital I/O module due to hardware limitations. For this reason, a special routine is written to read loopbacked channel information.

```

/*****
_ccrtnngfc_lib_error_number_t
ccrtnngfc_ldio_Read_Output_Loopbacked_Channels(void          *Handle,
                                                    _ccrtnngfc_ldio_input_snapshot_t ldio_snapshot,
                                                    ccrtnngfc_bool          skip_ldio_disabled_check,
                                                    ccrtnngfc_ldio_modules_t LIO_LoopbackedChannels,
                                                    ccrtnngfc_ldio_modules_t ChannelSelectMask)

```

Description: Read LIO Output Loopbacked Channels

```

Input:  void          *Handle          (handle pointer)
        _ccrtnngfc_ldio_input_snapshot_t ldio_snapshot (ldio_snapshot operation)
        # CCRTNGFC_LDIO_INPUT_OPERATION_CONTINUOUS
        # CCRTNGFC_LDIO_INPUT_OPERATION_SNAPSHOT
        # CCRTNGFC_LDIO_INPUT_OPERATION_DO_NOT_CHANGE
        ccrtnngfc_bool          skip_ldio_disabled_check (skip LVDS and Digital

```

I/O disabled check)

```
# CCRTNGFC_TRUE
# CCRTNGFC_FALSE
ccrtngfc_ldio_modules_t      ChannelSelectMask      (custom channel selection)
# NULL                        (select all channels)
# u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0      // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1      // On Main Board - LIO
# CCRTNGFC_DC_LIO_MODULE_2        // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3        // optional LIO Daughter Card
Output: ccrtngfc_ldio_modules_t  LIO_LoopbackedChannels  (LIO Loopbacked channels)
# u_int32_t  ccrtngfc_ldio_modules_t [CCRTNGFC_LDIO_MAX_MODULES]
# CCRTNGFC_LDIO_CHANNEL_MASK_0
# CCRTNGFC_LDIO_CHANNEL_MASK_1
# CCRTNGFC_LDIO_CHANNEL_MASK_2
# CCRTNGFC_LDIO_CHANNEL_MASK_3
# CCRTNGFC_LDIO_CHANNEL_MASK_4
# CCRTNGFC_LDIO_CHANNEL_MASK_5
# CCRTNGFC_LDIO_CHANNEL_MASK_6
# CCRTNGFC_LDIO_CHANNEL_MASK_7
# CCRTNGFC_LDIO_CHANNEL_MASK_8
# CCRTNGFC_LDIO_CHANNEL_MASK_9
# CCRTNGFC_LDIO_CHANNEL_MASK_10
# CCRTNGFC_LDIO_CHANNEL_MASK_11
# CCRTNGFC_LDIO_CHANNEL_MASK_12
# CCRTNGFC_LDIO_CHANNEL_MASK_13
# CCRTNGFC_LDIO_CHANNEL_MASK_14
```

```

# CCRTNGFC_LDIO_CHANNEL_MASK_15
# CCRTNGFC_LDIO_CHANNEL_MASK_16
# CCRTNGFC_LDIO_CHANNEL_MASK_17
# CCRTNGFC_LDIO_CHANNEL_MASK_18
# CCRTNGFC_LDIO_CHANNEL_MASK_19
# CCRTNGFC_LDIO_CHANNEL_MASK_20
# CCRTNGFC_LDIO_CHANNEL_MASK_21
# CCRTNGFC_LDIO_CHANNEL_MASK_22
# CCRTNGFC_LDIO_CHANNEL_MASK_23
# CCRTNGFC_LDIO_CHANNEL_MASK_24
# CCRTNGFC_LDIO_CHANNEL_MASK_25
# CCRTNGFC_LDIO_CHANNEL_MASK_26
# CCRTNGFC_LDIO_CHANNEL_MASK_27
# CCRTNGFC_LDIO_CHANNEL_MASK_28
# CCRTNGFC_LDIO_CHANNEL_MASK_29
# CCRTNGFC_LDIO_CHANNEL_MASK_30
# CCRTNGFC_LDIO_CHANNEL_MASK_31
# CCRTNGFC_LDIO_ALL_CHANNELS_MASK
CCRTNGFC_LDIO_MAX_MODULES can be one of:
# CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
# CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
# CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
# CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card

Return: _ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR (successful)
# CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN (device not open)
# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
# CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
*****/

```

2.2.116 ccrtNGFC_LIO_Set_Ports_Direction()

This call sets the direction of the LVDS I/O channels.

The module selection for the LIO located on the mother-board is CCRTNGFC_MAIN_LIO_MODULE_1. The CCRTNGFC_MAIN_DIO_MODULE_0 is for the DIO module that is located on the mother-board and is therefore invalid for this LIO API. *Currently there is no LIO daughter-card available.*

Unlike the Digital I/O ports where each Digital I/O ports is on a per channel basis, the LVDS I/O ports are grouped into 4 channels each. E.g. CCRTNGFC_LIO_PORT_MASK_P0 controls channels 0..3, CCRTNGFC_LIO_PORT_MASK_P1 controls channels 4..7, etc.

The LIO test mode register determines whether a LIO port of LVDS signals are in test mode or not. The test mode is used to turn-around a single LIO port (*four channel group*). It is used in conjunction with only ONE direction port setting at a time to test the LVDS drivers and receivers. Any attempt to turn-around more than one LIO port at a time will be ignored.

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 LVDS I/O lines.

When the direction for channels are set to output, then reading the channels input registers will result in invalid channel information being returned as the only way to read back (loopback) the channel information is to use the LIO Test Mode as specified with the *ccrtNGFC_LIO_Set_Ports_Direction()* call and select only ONE port to read at a time. If the user wishes to read back (loopback) the channel information for the LVDS I/O module, then they will need to use the specific *ccrtNGFC_LIO_Read_Output_Loopbacked_Channels()* call as that is specifically written for this purpose.

When issuing this call, the users must initialize *all* available LIO modules properly, otherwise the call will fail. If an LIO module is to be skipped in this call, the *LDIO_PortSelectionMask* for a LIO module must be set to zero (*i.e. no ports are selected*).

/******

```

_ccrtngfc_lib_error_number_t
ccrtngfc_LIO_Set_Ports_Direction(void          *Handle,
                                   ccrtnngfc_lio_test_modes_t LIO_TestMode,
                                   ccrtnngfc_lio_ports_t      LIO_PortDirection,
                                   ccrtnngfc_lio_ports_t      PortSelectionMask)

```

Description: Set LIO Port Direction Mask

```

Input:  void          *Handle          (handle pointer)
        ccrtnngfc_lio_test_modes_t    LIO_TestMode      (LVDS I/O test mode)
        # _ccrtnngfc_lio_port_mode_t  ccrtnngfc_lio_test_modes_t[CCRTNGFC_LDIO_MAX_MODULES]
        # CCRTNGFC_LIO_NORMAL_MODE
        # CCRTNGFC_LIO_TEST_MODE
        ccrtnngfc_lio_ports_t         LIO_PortDirection (port direction)
        - CCRTNGFC_LDIO_DIRECTION_INPUT   = (0)
        - CCRTNGFC_LDIO_DIRECTION_OUTPUT  = (1)
        # _ccrtnngfc_lio_port_mask_t   ccrtnngfc_lio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
        # CCRTNGFC_LIO_PORT_MASK_P0
        # CCRTNGFC_LIO_PORT_MASK_P1
        # CCRTNGFC_LIO_PORT_MASK_P2
        # CCRTNGFC_LIO_PORT_MASK_P3
        # CCRTNGFC_LIO_PORT_MASK_P4
        # CCRTNGFC_LIO_PORT_MASK_P5
        # CCRTNGFC_LIO_PORT_MASK_P6
        # CCRTNGFC_LIO_PORT_MASK_P7
        # CCRTNGFC_LIO_ALL_PORTS_MASK
        CCRTNGFC_LDIO_MAX_MODULES can be one of:
        # CCRTNGFC_MAIN_DIO_MODULE_0    // On Main Board - DIO
        # CCRTNGFC_MAIN_LIO_MODULE_1    // On Main Board - LIO
        # CCRTNGFC_DC_LIO_MODULE_2      // optional LIO Daughter Card
        # CCRTNGFC_DC_LIO_MODULE_3      // optional LIO Daughter Card
        ccrtnngfc_lio_ports_t         PortSelectionMask (port selection mask)
        # NULL                          (select all ports)
        # _ccrtnngfc_lio_port_mask_t   ccrtnngfc_lio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
        # CCRTNGFC_LIO_PORT_MASK_P0
        # CCRTNGFC_LIO_PORT_MASK_P1
        # CCRTNGFC_LIO_PORT_MASK_P2
        # CCRTNGFC_LIO_PORT_MASK_P3
        # CCRTNGFC_LIO_PORT_MASK_P4
        # CCRTNGFC_LIO_PORT_MASK_P5
        # CCRTNGFC_LIO_PORT_MASK_P6
        # CCRTNGFC_LIO_PORT_MASK_P7
        # CCRTNGFC_LIO_ALL_PORTS_MASK
        CCRTNGFC_LDIO_MAX_MODULES can be one of:
        # CCRTNGFC_MAIN_DIO_MODULE_0    // On Main Board - DIO
        # CCRTNGFC_MAIN_LIO_MODULE_1    // On Main Board - LIO
        # CCRTNGFC_DC_LIO_MODULE_2      // optional LIO Daughter Card
        # CCRTNGFC_DC_LIO_MODULE_3      // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION    (local region not present)
        # CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)

```

```
# CCRTNGFC_LIB_LIO_TEST_MODE_SETTING_ERROR
(Only one output port must be set)
*****/
```

2.2.117 ccrtNGFC_LIO_Set_Ports_Direction_To_Input()

This call sets the direction of the LVDS I/O channels to input.

The module selection for the LIO located on the mother-board is CCRTNGFC_MAIN_LIO_MODULE_1. The CCRTNGFC_MAIN_DIO_MODULE_0 is for the DIO module that is located on the mother-board and is therefore invalid for this LIO API. *Currently there is no LIO daughter-card available.*

Unlike the Digital I/O ports where each Digital I/O ports is on a per channel basis, the LVDS I/O ports are grouped into 4 channels each. E.g. CCRTNGFC_LIO_PORT_MASK_P0 controls channels 0..3, CCRTNGFC_LIO_PORT_MASK_P1 controls channels 4..7, etc.

When the direction for channels are set to output, then reading the channels input registers will result in invalid channel information being returned as the only way to read back (loopback) the channel information is to use the LIO Test Mode as specified with the *ccrtNGFC_LIO_Set_Ports_Direction()* call and select only ONE port to read at a time. If the user wishes to read back (loopback) the channel information for the LVDS I/O module, then they will need to use the specific *ccrtNGFC_LIO_Read_Output_Loopbacked_Channels()* call as that is specifically written for this purpose.

If the LIO test mode is previously set, then only one port (4 channels) can be set for output. If more than one port is set or no ports are set for output, then the firmware action is unpredictable.

In order to skip a DIO module, simply set *LIO_InputPortDirection* to zero for that module.

```
/******
_ccrtngfc_lib_error_number_t
ccrtNGFC_LIO_Set_Ports_Direction_To_Input(void *Handle,
                                           ccrtngfc_lio_ports_t LIO_InputPortDirection)

Description: Set LIO Port Direction To Input

Input: void *Handle (handle pointer)
       ccrtngfc_lio_ports_t LIO_InputPortDirection (input port direction)
       # _ccrtngfc_lio_port_mask_t ccrtngfc_lio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
       # CCRTNGFC_LIO_PORT_MASK_P0
       # CCRTNGFC_LIO_PORT_MASK_P1
       # CCRTNGFC_LIO_PORT_MASK_P2
       # CCRTNGFC_LIO_PORT_MASK_P3
       # CCRTNGFC_LIO_PORT_MASK_P4
       # CCRTNGFC_LIO_PORT_MASK_P5
       # CCRTNGFC_LIO_PORT_MASK_P6
       # CCRTNGFC_LIO_PORT_MASK_P7
       # CCRTNGFC_LIO_ALL_PORTS_MASK
       CCRTNGFC_LDIO_MAX_MODULES can be one of:
       # CCRTNGFC_MAIN_DIO_MODULE_0 // On Main Board - DIO
       # CCRTNGFC_MAIN_LIO_MODULE_1 // On Main Board - LIO
       # CCRTNGFC_DC_LIO_MODULE_2 // optional LIO Daughter Card
       # CCRTNGFC_DC_LIO_MODULE_3 // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
       # CCRTNGFC_LIB_NO_ERROR (successful)
       # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
       # CCRTNGFC_LIB_NOT_OPEN (device not open)
       # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
       # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
       # CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
       # CCRTNGFC_LIB_LIO_TEST_MODE_SETTING_ERROR
```

(Only one output port must be set)

*****/

2.2.118 ccrtNGFC_LIO_Set_Ports_Direction_To_Output()

This call sets the direction of the LVDS I/O channels to output.

The module selection for the LIO located on the mother-board is CCRTNGFC_MAIN_LIO_MODULE_1. The CCRTNGFC_MAIN_DIO_MODULE_0 is for the DIO module that is located on the mother-board and is therefore invalid for this LIO API. *Currently there is no LIO daughter-card available.*

Unlike the Digital I/O ports where each Digital I/O ports is on a per channel basis, the LVDS I/O ports are grouped into 4 channels each. E.g. CCRTNGFC_LIO_PORT_MASK_P0 controls channels 0..3, CCRTNGFC_LIO_PORT_MASK_P1 controls channels 4..7, etc.

When the direction for channels are set to output, then reading the channels input registers will result in invalid channel information being returned as the only way to read back (loopback) the channel information is to use the LIO Test Mode as specified with the *ccrtNGFC_LIO_Set_Ports_Direction()* call and select only ONE port to read at a time. If the user wishes to read back (loopback) the channel information for the LVDS I/O module, then they will need to use the specific *ccrtNGFC_LIO_Read_Output_Loopbacked_Channels()* call as that is specifically written for this purpose.

If the LIO test mode is previously set, then only one port (4 channels) can be set for output. If more than one port is set or no ports are set for output, then the firmware action is unpredictable.

In order to skip a DIO module, simply set *LIO_OutputPortDirection* to zero for that module.

/******

```
_ccrtngfc_lib_error_number_t
ccrtngfc_LIO_Set_Ports_Direction_To_Output(void          *Handle,
                                           ccrtngfc_lio_ports_t  LIO_OutputPortDirection)
```

Description: Set LIO Port Direction To Output

```

Input:  void          *Handle          (handle pointer)
        ccrtngfc_lio_ports_t  LIO_OutputPortDirection  (output port direction)
        # _ccrtngfc_lio_port_mask_t  ccrtngfc_lio_ports_t[CCRTNGFC_LDIO_MAX_MODULES]
        # CCRTNGFC_LIO_PORT_MASK_P0
        # CCRTNGFC_LIO_PORT_MASK_P1
        # CCRTNGFC_LIO_PORT_MASK_P2
        # CCRTNGFC_LIO_PORT_MASK_P3
        # CCRTNGFC_LIO_PORT_MASK_P4
        # CCRTNGFC_LIO_PORT_MASK_P5
        # CCRTNGFC_LIO_PORT_MASK_P6
        # CCRTNGFC_LIO_PORT_MASK_P7
        # CCRTNGFC_LIO_ALL_PORTS_MASK
        CCRTNGFC_LDIO_MAX_MODULES can be one of:
        # CCRTNGFC_MAIN_DIO_MODULE_0    // On Main Board - DIO
        # CCRTNGFC_MAIN_LIO_MODULE_1    // On Main Board - LIO
        # CCRTNGFC_DC_LIO_MODULE_2      // optional LIO Daughter Card
        # CCRTNGFC_DC_LIO_MODULE_3      // optional LIO Daughter Card

Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
        # CCRTNGFC_LIB_NO_LOCAL_REGION   (local region not present)
```

```

# CCRTNGFC_LIB_LDIO_IS_NOT_ACTIVE (LDIO is not active)
# CCRTNGFC_LIB_LIO_TEST_MODE_SETTING_ERROR
(Only one output port must be set)
*****/

```

2.2.119 ccrtNGFC_MMap_Physical_Memory()

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

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

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

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



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

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

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

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

```

/*****

```

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_MMap_Physical_Memory (void          *Handle,
                               int           size,
                               ccrtngfc_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: ccrtngfc_user_phys_mem_t     *phys_mem      (mem struct pointer)

```

```

uint          user_pid
void          *phys_mem_ptr
void          *driver_virt_mem_ptr
void          *mmaped_user_mem_ptr
uint         phys_mem_size
uint         phys_mem_size_freed
uint         entry_num_in_tran_table
ushort       flags
             # CCRTNGFC_DEVICE_ADDRESS_ENTRY
             # CCRTNGFC_DISABLE_ADDRESS_CACHE
             # CCRTNGFC_DISABLE_REGION_PROTECTION
ushort       num_of_entries_used
Return:      _ccrtngfc_lib_error_number_t
             # CCRTNGFC_LIB_NO_ERROR           (successful)
             # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
             # CCRTNGFC_LIB_NOT_OPEN         (device not open)
             # CCRTNGFC_LIB_INVALID_ARG      (invalid argument)
             # CCRTNGFC_LIB_MMAP_SELECT_FAILED (mmap selection failed)
             # CCRTNGFC_LIB_MMAP_FAILED      (mmap failed)
             # CCRTNGFC_LIB_NO_SPACE_IN_TABLE (no space in phys memory table)
             # CCRTNGFC_LIB_REGION_ADDRESSING_NOT_SUPPORTED
                                                     (region addressing not
                                                     supported by the card)
             # CCRTNGFC_LIB_MSGDMA_ACCESS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                                     (access not allowed for selected address)
*****/

```

2.2.120 ccrtNGFC_MsgDma_Clone() (US Patent No.: US 11,281,584 B1®)

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

There are currently six MsgDma engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four MsgDma engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a MsgDma operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific MsgDma engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available MsgDma engine is returned to the user. This is the MsgDma engine number that needs to be used for all the MsgDma operations until the MsgDma is freed via the *ccrtNGFC_MsgDma_Release()* API.

The following are the operation modes for this call:

- CCRTNGFC_MSGDMA_CLONE_INITIALIZE
- CCRTNGFC_MSGDMA_CLONE_ONE_CYCLE_WAIT
- CCRTNGFC_MSGDMA_CLONE_START
- CCRTNGFC_MSGDMA_CLONE_STOP

In order to perform a Cloning operation, the user first performs the same functions of MsgDma to seize, configure descriptors and MsgDma setup using the `ccrtNGFC_MsgDma_Seize()`, `ccrtNGFC_MsgDma_Configure_Descriptor()` and `ccrtNGFC_MsgDma_Setup()` calls. Once that is done, the user needs to stop any previous MsgDma operation and initialize the cloning operation using (`CCRTNGFC_MSGDMA_CLONE_STOP` | `CCRTNGFC_MSGDMA_CLONE_INITIALIZE`) modes.

Now, whenever the user is ready, they can commence cloning operation with the `CCRTNGFC_MSGDMA_CLONE_START` mode. At this point, MsgDma transfers start occurring continuously at the hardware level. If a chained MsgDma is configured, the entire chain is completed before it is repeated. Once Cloning has commenced, it can be stopped with the help of the `CCRTNGFC_MSGDMA_CLONE_STOP` mode.

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

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

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



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

```

/*****

```

```

ccrtNGFC_MsgDma_Clone()
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Clone(void          *Handle,
                                _ccrtngfc_msgdma_engine_t      MsgDmaEngine,
                                _ccrtngfc_msgdma_clone_mode_mask_t ModeMask,
                                ccrtngfc_msgdma_clone_args_t    *CloneArgs)

```

Description: Clone Modular Scatter-Gather DMA

```

Input:  void          *Handle (Handle pointer)
        _ccrtngfc_msgdma_engine_t      MsgDmaEngine
        # CCRTNGFC_MSGDMA_ENGINE_0    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4    // single-word transfers

```

```

    # CCRTNGFC_MSGDMA_ENGINE_5           // single-word transfers
    _ccrtngfc_msgdma_clone_mode_mask_t   ModeMask (Mode Mask)
    # CCRTNGFC_MSGDMA_CLONE_INITIALIZE
    # CCRTNGFC_MSGDMA_CLONE_ONE_CYCLE_WAIT
    # CCRTNGFC_MSGDMA_CLONE_START
    # CCRTNGFC_MSGDMA_CLONE_STOP
    ccrtngfc_msgdma_clone_args_t         *CloneArgs
    - unsigned long long                 AdditionalOneCycleDelay
                                        (Additional blocking for One Cycle Delay
                                        (nanoseconds))
Output: ccrtngfc_msgdma_clone_args_t    *CloneArgs
    - unsigned long long                 MsgDmaExtDesOneCycleDelay
                                        (MsgDma Extended Descriptor One Cycle
                                        Delay (nanoseconds))

Return: _ccrtngfc_lib_error_number_t
    # CCRTNGFC_LIB_NO_ERROR              (successful)
    # CCRTNGFC_LIB_INVALID_ARG           (invalid argument)
    # CCRTNGFC_LIB_MSGDMA_FAILED         (MsgDma failed)
    # CCRTNGFC_LIB_MSGDMA_IN_USE        (MsgDma in use)
    # CCRTNGFC_LIB_MSGDMA_NOT_SETUP     (MsgDma not setup)
    # CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA  (not owner of MsgDma)
    # CCRTNGFC_LIB_MSGDMA_FAILED        (MsgDma failed to start)
    # CCRTNGFC_LIB_MSGDMA_ACCESS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                        (MsgDma not allowed for selected address)
    # CCRTNGFC_LIB_CLONING_NOT_SUPPORTED (Cloning not supported by the card)
*****/

```

2.2.121 ccrtNGFC_MsgDma_Configure_Descriptor()

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

There are currently six MsgDma engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four MsgDma engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a MsgDma operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific MsgDma engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available MsgDma engine is returned to the user. This is the MsgDma engine number that needs to be used for all the MsgDma operations until the MsgDma is freed via the *ccrtNGFC_MsgDma_Release()* API.

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

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

DMA transfers can occur from either of the following:

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

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

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

```

/*****
ccrtngfc_MsgDma_Configure_Descriptor()
_ccrtngfc_lib_error_number_t
ccrtngfc_MsgDma_Configure_Descriptor (void                                *Handle,
                                     _ccrtngfc_msgdma_engine_t          MsgDmaEngine,
                                     _ccrtngfc_msgdma_descriptors_id_t *DescriptorID,
                                     ccrtngfc_msgdma_descriptor_t       *Descriptor,
                                     _ccrtngfc_msgdma_descriptors_id_t AttachToDescriptorID)

```

Description: Configure Modular Scatter-Gather DMA descriptor

```

Input:  void                                *Handle (Handle pointer)
        _ccrtngfc_msgdma_engine_t          MsgDmaEngine
        # CCRTNGFC_MSGDMA_ENGINE_0        // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1        // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2        // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3        // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4        // single-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_5        // single-word transfers
        _ccrtngfc_msgdma_descriptors_id_t *DescriptorID (Set to NULL or valid ID)
        # 0                                  (let function find a free ID)
        # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1 ... CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31
        ccrtngfc_msgdma_descriptor_t *Descriptor (pointer to descriptor)
        __u64 ReadAddress
        __u64 WriteAddress
        __u32 Length
        _ccrtngfc_msgdma_descriptors_id_t AttachToDescriptorID (Attach to descriptor ID)
        # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1 ... CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31
Output: _ccrtngfc_msgdma_descriptors_id_t *DescriptorID (returned ID)
        # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1 ... CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR              (successful)
        # CCRTNGFC_LIB_BAD_HANDLE            (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN              (device not open)
        # CCRTNGFC_LIB_INVALID_ARG           (invalid argument)
        # CCRTNGFC_LIB_NO_FREE_DESCRIPTOR_AVAILABLE (no free descriptors available)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
        # CCRTNGFC_LIB_MSGDMA_READS_NOT_ALLOWED_FOR_SELECTED_ADDRESS

```



```

(MSG DMA Reads not allowed for selected
address)
# CCRTNGFC_LIB_MSGDMA_BUSY           (MsgDma Busy, cannot be reset)
# CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA  (not owner of modular scatter-gather)
# CCRTNGFC_LIB_DATA_WIDTH_NOT_MULTIPLE_OR_ALIGNED
(Data Width Not Multiple or Aligned)
*****/

```

2.2.122 ccrtNGFC_MsgDma_Configure_Single()

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

There are currently six MsgDma engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four MsgDma engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a MsgDma operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific MsgDma engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available MsgDma engine is returned to the user. This is the MsgDma engine number that needs to be used for all the MsgDma operations until the MsgDma is freed via the *ccrtNGFC_MsgDma_Release()* API.

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

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

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Configure_Single (void                *Handle,
                                _ccrtngfc_msgdma_engine_t  MsgDmaEngine,
                                ccrtngfc_msgdma_descriptor_t *Descriptor)

```

Description: Configure Single Modular Scatter-Gather DMA descriptor

```

Input:  void                *Handle (Handle pointer)
        _ccrtngfc_msgdma_engine_t  MsgDmaEngine
        # CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers
        ccrtngfc_msgdma_descriptor_t *Descriptor (pointer to descriptor)
        __u64  ReadAddress
        __u64  WriteAddress

```

```

    __u32 Length
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
        # CCRTNGFC_LIB_MSGDMA_BUSY       (MsgDma Busy, cannot be reset)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
        # CCRTNGFC_LIB_MSGDMA_READS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                           (MSG DMA Reads not allowed for selected
                                           address)
        # CCRTNGFC_LIB_MSGDMA_BUSY       (MsgDma Busy, cannot be reset)
        # CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA (not owner of modular scatter-gather)
        # CCRTNGFC_LIB_DATA_WIDTH_NOT_MULTIPLE_OR_ALIGNED
                                           (Data Width Not Multiple or Aligned)
*****/

```

2.2.123 ccrtNGFC_MsgDma_Fire()

This call initiates a scatter-gather DMA operation that has been previously configured and setup by the *ccrtNGFC_MsgDma_Configure()* and *ccrtNGFC_MsgDma_Setup()* call.

There are currently six MsgDma engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four MsgDma engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a MsgDma operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific MsgDma engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available MsgDma engine is returned to the user. This is the MsgDma engine number that needs to be used for all the MsgDma operations until the MsgDma is freed via the *ccrtNGFC_MsgDma_Release()* API.

The *StartDescriptorID* can be set to either '0' or a valid Descriptor ID. Normally, the user will set the *StartDescriptorID* in the *ccrtNGFC_MsgDma_Setup()* API during initialization and set it to '0' in this *ccrtNGFC_MsgDma_Fire()* API. In this way, this call will not suffer the overhead of loading the *StartDescriptorID* in the internal prefetcher register when repeatedly calling the *ccrtNGFC_MsgDma_Fire()* API. If the user specifies a valid *StartDescriptorID* that is already setup as a scatter-gather chain using the *ccrtNGFC_MsgDma_Configure()* call, then this *ccrtNGFC_MsgDma_Fire()* API will initiate the DMA starting with the user supplied start descriptor ID.

The *DescriptorIDMask* is a mask of all the valid descriptor ID's specified in the scatter-gather chain that was created earlier with the *ccrtNGFC_MsgDma_Configure()* API. If this is incorrectly specified, the DMA operation will be unpredictable. This *ccrtNGFC_MsgDma_Fire()* API call uses this mask to set the *ControlWord* for each of the IDs. Specifying this mask reduces the overhead in the call by not searching the scatter-gather chain to set the individual control words.

ControlWord for each descriptor is set based on the *DescriptorIDMask* mask. Normally, the following two flags are set:

- CCRTNGFC_MSGD_DESC_CONTROL_GO

- CCRTNGFC_MSGD_DESC_CONTROL_OWNED_BY_HW

LastIdForInterrupts is set to 0 if the DMA operation will use polling instead of using interrupts to detect completion of the operation. If interrupts are to be used, the ID of the last descriptor in the DMA chain is to be specified. This is the ID that will be interrupted when the entire chain is completed. Incorrect ID entered will result in unpredictable results. Normally, interrupt handling adds additional overhead and reduces performance, however, it reduces the overhead experienced by the CPU and PCIe bus during polling.

Once the scatter-gather DMA operation commences, it performs DMA operations starting with the *StartDescriptorID* and traversing through the chain sequentially until it reaches the last descriptor ID in the chain, at which point the DMA operation concludes.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Fire (void                *Handle,
                        _ccrtngfc_msgdma_engine_t    MsgDmaEngine,
                        _ccrtngfc_msgdma_descriptors_id_t    StartDescriptorID,
                        _ccrtngfc_msgdma_descriptors_id_mask_t    DescriptorIDMask,
                        int                ControlWord,
                        _ccrtngfc_msgdma_descriptors_id_t    LastIdForInterrupts)

```

Description: Fire Modular Scatter-Gather DMA descriptor

```

Input:  void                *Handle (Handle pointer)
        _ccrtngfc_msgdma_engine_t    MsgDmaEngine
        # CCRTNGFC_MSGDMA_ENGINE_0    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4    // single-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_5    // single-word transfers
        _ccrtngfc_msgdma_descriptors_id_t    StartDescriptorID (Set to valid ID)
        # 0                (don't set start descriptor ID in prefetcher)
        # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1 ... CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31
        _ccrtngfc_msgdma_descriptors_id_mask_t    DescriptorIDMask (descriptor ID mask)
        # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1_MASK ...
CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31_MASK
        # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_ALL_MASK
int                ControlWord
        # CCRTNGFC_MSGD_DESC_CONTROL_GO
        # CCRTNGFC_MSGD_DESC_CONTROL_OWNED_BY_HW
        _ccrtngfc_msgdma_descriptors_id_t    LastIdForInterrupts (Set 0 or Last ID
                                                                    for interrupts)
        # 0                (don't fire interrupts)
        # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1 ... CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31
Output:  none
Return:  _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR        (successful)
        # CCRTNGFC_LIB_INVALID_ARG    (invalid argument)
        # CCRTNGFC_LIB_MSGDMA_FAILED  (MsgDma failed)
        # CCRTNGFC_LIB_MSGDMA_BUSY    (MsgDma Busy, cannot be reset)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
        # CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA (not owner of modular scatter-gather)
*****/

```

2.2.124 ccrtNGFC_MsgDma_Fire_Fifo()

This call is similar in functionality to the *ccrtNGFC_MsgDma_Fire()* call with the exception that it expects the entire memory area for transfer to be a read or write FIFO. It can be used when users need to perform FIFO transfers. This call can be called once the *ccrtNGFC_MsgDma_Configure_Descriptor()* call has been issued to set

up the read/write address offset and length of transfer. Unless the read/write address offset or length of transfer is changed, the *ccrtNGFC_MsgDma_Fire_Fifo()* call can be made repeatedly to perform the same DMA transfer.

There are currently six *MsgDma* engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four *MsgDma* engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a *MsgDma* operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific *MsgDma* engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available *MsgDma* engine is returned to the user. This is the *MsgDma* engine number that needs to be used for all the *MsgDma* operations until the *MsgDma* is freed via the *ccrtNGFC_MsgDma_Release()* API

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Fire_Fifo (void                                *Handle,
                           _ccrtngfc_msgdma_engine_t         MsgDmaEngine,
                           _ccrtngfc_msgdma_descriptors_id_t LastDescriptorId,
                           int                                 UseInterrupts)
Description: Fire ADC or DAC Fifo Modular Scatter-Gather DMA descriptor

Input:  void                                *Handle           (Handle pointer)
        _ccrtngfc_msgdma_engine_t          MsgDmaEngine
        # CCRTNGFC_MSGDMA_ENGINE_0        // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1        // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2        // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3        // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4        // single-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_5        // single-word transfers
        _ccrtngfc_msgdma_descriptors_id_t LastDescriptorId    (Last Descriptor ID)
        int                                 UseInterrupts      (Use interrupts flag)
        # CCRTNGFC_TRUE
        # CCRTNGFC_FALSE

Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
        # CCRTNGFC_LIB_MSGDMA_FAILED      (MsgDma failed)
        # CCRTNGFC_LIB_MSGDMA_BUSY        (MsgDma Busy, cannot be reset)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
        # CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA (not owner of modular scatter-gather)
*****/

```

2.2.125 ccrtNGFC_MsgDma_Fire_Single()

This call is similar in functionality to the *ccrtNGFC_MsgDma_Fire()* call with the exception that it operates on the single descriptor ID-1. It can be used when a single DMA rather than scatter-gather DMA operation needs to be performed. This call can be called once the *ccrtNGFC_MsgDma_Configure_Single()* call has been issued to set up the read/write address offset and length of transfer. Unless the read/write address offset or length of transfer is changed, the *ccrtNGFC_MsgDma_Fire_Single()* call can be made repeatedly to perform the same DMA transfer.

There are currently six `MsgDma` engines available to the user. They can be selected via the `MsgDmaEngine` option. Since the first four `MsgDma` engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- `CCRTNGFC_MSGDMA_ENGINE_0` // quad-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_1` // quad-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_2` // quad-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_3` // quad-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_4` // single-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_5` // single-word transfers

The normal process to setup a `MsgDma` operation is to first call the `ccrtNGFC_MsgDma_Seize()` API with either a specific `MsgDma` engine or the optional argument `CCRTNGFC_MSGDMA_ENGINE_IDLE`. In this case, the first available `MsgDma` engine is returned to the user. This is the `MsgDma` engine number that needs to be used for all the `MsgDma` operations until the `MsgDma` is freed via the `ccrtNGFC_MsgDma_Release()` API.

/******

```
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Fire_Single (void *Handle,
                             _ccrtngfc_msgdma_engine_t MsgDmaEngine,
                             int UseInterrupts)
```

Description: Fire Single Modular Scatter-Gather DMA descriptor

```
Input: void *Handle (Handle pointer)
       _ccrtngfc_msgdma_engine_t MsgDmaEngine
       # CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers
       int UseInterrupts (Use interrupts flag)
       # CCRTNGFC_TRUE
       # CCRTNGFC_FALSE

Output: none
Return: _ccrtngfc_lib_error_number_t
       # CCRTNGFC_LIB_NO_ERROR (successful)
       # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
       # CCRTNGFC_LIB_MSGDMA_FAILED (MsgDma failed)
       # CCRTNGFC_LIB_MSGDMA_BUSY (MsgDma Busy, cannot be reset)
       # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
       # CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA (not owner of modular scatter-gather)
```

*****/

`UseInterrupts` is a flag that can be set to specify if interrupt handling should be enabled.

2.2.126 `ccrtNGFC_MsgDma_Free_Descriptor()`

This call can be used to free up already used descriptors.

There are currently six `MsgDma` engines available to the user. They can be selected via the `MsgDmaEngine` option. Since the first four `MsgDma` engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- `CCRTNGFC_MSGDMA_ENGINE_0` // quad-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_1` // quad-word transfers

- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a MsgDma operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific MsgDma engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available MsgDma engine is returned to the user. This is the MsgDma engine number that needs to be used for all the MsgDma operations until the MsgDma is freed via the *ccrtNGFC_MsgDma_Release()* API.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Free_Descriptor (void                                *Handle,
                                _ccrtngfc_msgdma_engine_t          MsgDmaEngine,
                                _ccrtngfc_msgdma_descriptors_id_mask_t DescriptorIDMask)

Description: Free Modular Scatter-Gather DMA descriptor

Input:   void                                *Handle (Handle pointer)
         _ccrtngfc_msgdma_engine_t          MsgDmaEngine
         # CCRTNGFC_MSGDMA_ENGINE_0        // quad-word transfers
         # CCRTNGFC_MSGDMA_ENGINE_1        // quad-word transfers
         # CCRTNGFC_MSGDMA_ENGINE_2        // quad-word transfers
         # CCRTNGFC_MSGDMA_ENGINE_3        // quad-word transfers
         # CCRTNGFC_MSGDMA_ENGINE_4        // single-word transfers
         # CCRTNGFC_MSGDMA_ENGINE_5        // single-word transfers
         _ccrtngfc_msgdma_descriptors_id_mask_t DescriptorIDMask (descriptor ID mask)
         # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1_MASK ... CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31_MASK
         # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_ALL_MASK

Output:  none
Return:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR           (successful)
         # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
         # CCRTNGFC_LIB_NOT_OPEN          (device not open)
         # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
         # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
         # CCRTNGFC_LIB_MSGDMA_BUSY       (MsgDma Busy, cannot be reset)
         # CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA (not owner of modular scatter-gather)
*****/

```

2.2.127 ccrtNGFC_MsgDma_Get_Descriptor()

This call returns information on the selected descriptor.

There are currently six MsgDma engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four MsgDma engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a MsgDma operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific MsgDma engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first

available MsgDma engine is returned to the user. This is the MsgDma engine number that needs to be used for all the MsgDma operations until the MsgDma is freed via the *ccrtNGFC_MsgDma_Release()* API.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Get_Descriptor (void                *Handle,
                                _ccrtngfc_msgdma_engine_t    MsgDmaEngine,
                                _ccrtngfc_msgdma_descriptors_id_t DescriptorID,
                                ccrtngfc_msgdma_descriptor_t *Descriptor,
                                __u64                        *DescriptorAddress)

```

Description: Get Modular Scatter-Gather DMA Descriptor

```

Input:  void                *Handle (Handle pointer)
        _ccrtngfc_msgdma_engine_t    MsgDmaEngine
        # CCRTNGFC_MSGDMA_ENGINE_0    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3    // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4    // single-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_5    // single-word transfers
        _ccrtngfc_msgdma_descriptors_id_t DescriptorID (descriptor ID)
        # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1 ... CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31
Output: ccrtngfc_msgdma_descriptor_t *Descriptor (pointer to descriptor)
        __u64  ReadAddress
        __u64  WriteAddress
        __u64  NextDescriptorPointer
        __u32  Length
        __u32  Control
        __u32  ReadBurstCount
        __u32  WriteBurstCount
        __u32  ReadStride
        __u32  WriteStride
        __u32  ActualBytesTransferred
        __u32  Status
        __u32  SequenceNumber
        __u64  *DescriptorAddress (descriptor address)
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR        (successful)
        # CCRTNGFC_LIB_BAD_HANDLE     (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN      (device not open)
        # CCRTNGFC_LIB_INVALID_ARG   (invalid argument)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
*****/

```

Pointer to *DescriptorAddress* can be specified to return its address offset within the configuration space. This argument can be set to *NULL* if address is not required.

2.2.128 ccrtNGFC_MsgDma_Get_Dispatcher_CSR()

This call returns useful control and status register information on the dispatcher.

There are currently six MsgDma engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four MsgDma engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers

- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a MsgDma operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific MsgDma engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available MsgDma engine is returned to the user. This is the MsgDma engine number that needs to be used for all the MsgDma operations until the MsgDma is freed via the *ccrtNGFC_MsgDma_Release()* API.

/*
 /*****
 /*****

```

_ccrtngfc_lib_error_number_t
ccrtngfc_MsgDma_Get_Dispatcher_CSR (void          *Handle,
                                   _ccrtngfc_msgdma_engine_t  MsgDmaEngine,
                                   ccrtngfc_msgdma_dispatcher_t *Dispatcher)

```

Description: Get Modular Scatter-Gather DMA Dispatcher CSR

```

Input:  void          *Handle (Handle pointer)
        _ccrtngfc_msgdma_engine_t  MsgDmaEngine
        # CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers
Output: ccrtngfc_msgdma_dispatcher_t *Dispatcher (pointer to dispatcher)
        __u32 Status
        # CCRTNGFC_MSGD_DISP_STATUS_IRQ           :IRQ
        # CCRTNGFC_MSGD_DISP_STATUS_STOPPED_ETERM :Stopped on Early Termination
        # CCRTNGFC_MSGD_DISP_STATUS_STOPPED_ERROR :Stopped on Error
        # CCRTNGFC_MSGD_DISP_STATUS_RESETTING    :Resetting
        # CCRTNGFC_MSGD_DISP_STATUS_STOPPED      :Stopped
        # CCRTNGFC_MSGD_DISP_STATUS_RESP_BUF_FULL :Response Buffer Full
        # CCRTNGFC_MSGD_DISP_STATUS_RESP_BUF_EMPTY :Response Buffer Empty
        # CCRTNGFC_MSGD_DISP_STATUS_DESC_BUF_FULL :Descriptor Buffer Full
        # CCRTNGFC_MSGD_DISP_STATUS_DESC_BUF_EMPTY :Descriptor Buffer Empty
        # CCRTNGFC_MSGD_DISP_STATUS_BUSY         :Busy
        __u32 Control
        # CCRTNGFC_MSGD_DISP_CONTROL_STOP_DESC   :Stop Descriptors
        # CCRTNGFC_MSGD_DISP_CONTROL_INT_ENA_MASK :Global Interrupt Enable Mask
        # CCRTNGFC_MSGD_DISP_CONTROL_STOP_ETERM  :Stop on Early Termination
        # CCRTNGFC_MSGD_DISP_CONTROL_STOP_ON_ERROR :Stop on Error
        # CCRTNGFC_MSGD_DISP_CONTROL_RESET_DISP  :Reset Dispatcher
        # CCRTNGFC_MSGD_DISP_CONTROL_STOP_DISP   :Stop Dispatcher
        __u32 ReadFillLevel
        __u32 WriteFillLevel
        __u32 ResponseFillLevel
        __u32 ReadSequenceNumber
        __u32 WriteSequenceNumber
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR           (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_INVALID_ARG      (invalid argument)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
*****
/

```

2.2.129 ccrtNGFC_MsgDma_Get_Info()

This call returns useful information about the selected MsgDma engine.

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.

There are currently six `MsgDma` engines available to the user. They can be selected via the `MsgDmaEngine` option. Since the first four `MsgDma` engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- `CCRTNGFC_MSGDMA_ENGINE_0` // quad-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_1` // quad-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_2` // quad-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_3` // quad-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_4` // single-word transfers
- `CCRTNGFC_MSGDMA_ENGINE_5` // single-word transfers

The normal process to setup a `MsgDma` operation is to first call the `ccrtNGFC_MsgDma_Seize()` API with either a specific `MsgDma` engine or the optional argument `CCRTNGFC_MSGDMA_ENGINE_IDLE`. In this case, the first available `MsgDma` engine is returned to the user. This is the `MsgDma` engine number that needs to be used for all the `MsgDma` operations until the `MsgDma` is freed via the `ccrtNGFC_MsgDma_Release()` API.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Get_Info (void                *Handle,
                        _ccrtngfc_msgdma_engine_t MsgDmaEngine,
                        ccrtngfc_msgdma_list_t  *Info)

Description: Get Modular Scatter-Gather DMA Information

Input:  void                *Handle (Handle pointer)
        _ccrtngfc_msgdma_engine_t
        # CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers
Output: ccrtngfc_msgdma_list_t *Info (pointer to info)
        unsigned long          MsgDmaOwnerPid
        unsigned long          MsgDmaOwnerPid
        unsigned long long     MsgDmaExtDesOneCycleDelay
        __u64                  MsgDmaDescriptorBaseOffset
        __u64                  MsgDmaTerminatingDescriptorOffset
        ccrtngfc_msgdma_dispatcher_csr_t *MsgDmaDispatcherCsrDriverPtr
        ccrtngfc_msgdma_prefetcher_csr_t *MsgDmaPrefetcherCsrDriverPtr
        ccrtngfc_msgdma_extended_descriptor_t *MsgDmaExtendedDescriptorDriverPtr
        ccrtngfc_msgdma_extended_descriptor_t *MsgDmaTerminatingDescriptorDriverPtr
        ccrtngfc_msgdma_dispatcher_csr_t *MsgDmaDispatcherCsrLibraryPtr
        ccrtngfc_msgdma_prefetcher_csr_t *MsgDmaPrefetcherCsrLibraryPtr
        ccrtngfc_msgdma_extended_descriptor_t *MsgDmaExtendedDescriptorLibraryPtr
        ccrtngfc_msgdma_extended_descriptor_t *MsgDmaTerminatingDescriptorLibraryPtr
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN          (device not open)
        # CCRTNGFC_LIB_INVALID_ARG       (invalid argument)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
*****/

```

2.2.130 `ccrtNGFC_MsgDma_Get_Prefetcher_CSR()`

This call returns useful control and status register information on the prefetcher.

There are currently six MsgDma engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four MsgDma engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a MsgDma operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific MsgDma engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available MsgDma engine is returned to the user. This is the MsgDma engine number that needs to be used for all the MsgDma operations until the MsgDma is freed via the *ccrtNGFC_MsgDma_Release()* API.

/******

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Get_Prefetcher_CSR (void          *Handle,
                                     _ccrtngfc_msgdma_engine_t  MsgDmaEngine,
                                     ccrtngfc_msgdma_prefetcher_t *Prefetcher)

```

Description: Get Modular Scatter-Gather DMA Prefetcher CSR

```

Input:  void          *Handle (Handle pointer)
        _ccrtngfc_msgdma_engine_t  MsgDmaEngine
        # CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers
Output: ccrtngfc_msgdma_prefetcher_t *Prefetcher (pointer to prefetcher)
        __u32 Status
        # CCRTNGFC_MSGD_PREF_STATUS_IRQ      :IRQ Occurred
        __u32 Control
        # CCRTNGFC_MSGD_PREF_CONTROL_PARK_MODE      :Park Mode
        # CCRTNGFC_MSGD_PREF_CONTROL_INT_ENA_MASK  :Global Interrupt Enable Mask
        # CCRTNGFC_MSGD_PREF_CONTROL_RESET        :Reset Prefetcher Core
        # CCRTNGFC_MSGD_PREF_CONTROL_DESC_POLL_EN  :Descriptor Polling Enable
        # CCRTNGFC_MSGD_PREF_CONTROL_RUN          :Start Descriptor fetching operation
        __u64 NextDescriptorPointer
        __u32 DescriptorPollingFrequency
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (device not open)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
*****/

```

2.2.131 ccrtNGFC_MsgDma_Release()

This *ccrtNGFC_MsgDma_Release()* API call is used to free up the Modular Scatter-Gather DMA resource that has been previously reserved by the *ccrtNGFC_MsgDma_Seize()* API. At this point, another user can take control of the MsgDMA operation by issuing the *ccrtNGFC_MsgDma_Seize()* call.

There are currently six MsgDma engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four MsgDma engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a MsgDma operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific MsgDma engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available MsgDma engine is returned to the user. This is the MsgDma engine number that needs to be used for all the MsgDma operations until the MsgDma is freed via the *ccrtNGFC_MsgDma_Release()* API.

/*****

```
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Release (void *Handle,
                        _ccrtngfc_msgdma_engine_t MsgDmaEngine)
```

Description: Release MsgDMA operation for others to use

```
Input: void *Handle (Handle pointer)
       _ccrtngfc_msgdma_engine_t MsgDmaEngine
       # CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers
```

Output: none

```
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR (successful)
        # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
        # CCRTNGFC_LIB_MSGDMA_BUSY (MsgDma Busy, cannot be reset)
        # CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA (not owner of modular scatter-gather)
        # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
```

*****/

2.2.132 ccrtNGFC_MsgDma_Seize()

Modular Scatter-Gather DMA is a two part operation. The first part is to configure the Scatter-Gather DMA and the second part is to execute the DMA. Various MsgDma API calls have been provided for this. Since this two part operation is not atomic, it is necessary for the user of these calls to prevent other applications from configuring and using the same MsgDMA resources while it is being actively used by another application. For this reason, the *ccrtNGFC_MsgDma_Seize()* and *ccrtNGFC_MsgDma_Release()* API calls have been introduced to assist the user in preventing other applications from accessing the selected Scatter-Gather DMA resource while it is reserved. Basically, before any MsgDma API call is issued that could modify the setting and execution of the MsgDma operation, the user needs to issue the *ccrtNGFC_MsgDma_Seize()* API call once. In this way, no one else will have access to the MsgDma resource until the application has issued the *ccrtNGFC_MsgDma_Release()* API call or has terminated the application.

There are currently six MsgDma engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four MsgDma engines (0 through 3) perform quad-word transfers, they must be aligned on a quad-word (*multiple of 16 bytes*) boundary and the size must be multiples of four words or 16 bytes, however they are

slightly faster than the remaining two (4 or 5) that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_IDLE

Since multiple MsgDma operations can be in use at the same time, the user can select a specific MsgDma engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available MsgDma engine is returned to the user. This is the MsgDma engine number that needs to be used for all the MsgDma operations until the MsgDma is freed via the *ccrtNGFC_MsgDma_Release()* API.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Seize (void                *Handle,
                     _ccrtngfc_msgdma_engine_t *MsgDmaEngine)

```

Description: Seize MsgDMA operation for private to use and become owner

```

Input:  void                *Handle (Handle pointer)
        _ccrtngfc_msgdma_engine_t *MsgDmaEngine
        # CCRTNGFC_MSGDMA_ENGINE_IDLE
        # CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

Output: none

Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR (successful)
        # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN (device not open)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
        # CCRTNGFC_LIB_MSGDMA_IN_USE (modular scatter-gather DMA in use)
        # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
*****/

```

2.2.133 ccrtNGFC_MsgDma_Setup()

This call is used in conjunction with the *ccrtNGFC_MsgDma_Configure()* and *ccrtNGFC_MsgDma_Fire()* calls. This call is made after all the descriptors are first configured with the help of the *ccrtNGFC_MsgDma_Configure()* call. The purpose of this call is to specify the first descriptor in the chain. Additionally, the user can set the *ForceReset* flag to reset the dispatcher and prefetcher. Optionally, the user can request useful active descriptor information if *ActiveDescriptorsInfo* argument is specified (*i.e not NULL*). In addition to returning useful active descriptor information, the descriptor chain and prefetcher settings are also validated for proper configuration.

There are currently six MsgDma engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four MsgDma engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers

- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_MsgDma_Setup (void *Handle,
                        _ccrtngfc_msgdma_engine_t MsgDmaEngine,
                        _ccrtngfc_msgdma_descriptors_id_t StartDescriptorID,
                        int ForceReset,
                        ccrtngfc_msgdma_active_descriptors_info_t *ActiveDescriptorsInfo)

```

Description: Setup MsgDMA Dispatcher and Prefetcher

```

Input: void *Handle (Handle pointer)
       _ccrtngfc_msgdma_engine_t MsgDmaEngine
       # CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
       # CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers
       _ccrtngfc_msgdma_descriptors_id_t *StartDescriptorID (Set to valid ID)
       # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1 ... CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31
       int ForceReset
Output: ccrtngfc_msgdma_active_descriptors_info_t *ActiveDescriptorsInfo;
       _ccrtngfc_msgdma_descriptors_id_t ID
       # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1 ... CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31
       _ccrtngfc_msgdma_descriptors_id_mask_t Mask
       # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_1_MASK ... CCRTNGFC_MSGDMA_DESCRIPTOR_ID_31_MASK
       # CCRTNGFC_MSGDMA_DESCRIPTOR_ID_ALL_MASK
       _u32 NumberOfDescriptors
       _u32 TotalBytes
Return: _ccrtngfc_lib_error_number_t
       # CCRTNGFC_LIB_NO_ERROR (successful)
       # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
       # CCRTNGFC_LIB_NOT_OPEN (device not open)
       # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
       # CCRTNGFC_LIB_MSGDMA_BUSY (MsgDma Busy, cannot be reset)
       # CCRTNGFC_LIB_ERROR_IN_DESCRIPTOR_LIST (invalid descriptor list)
       # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
       # CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA (not owner of modular scatter-gather)
*****/

```

2.2.134 ccrtNGFC_Munmap_Physical_Memory()

This call simply removes a physical memory that was previously allocated by the *ccrtNGFC_MMap_Physical_Memory()* API call.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_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: _ccrtngfc_lib_error_number_t
       # CCRTNGFC_LIB_NO_ERROR (successful)

```

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.

```

# CCRTNGFC_LIB_BAD_HANDLE          (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN            (device not open)
# CCRTNGFC_LIB_INVALID_ARG        (invalid argument)
# CCRTNGFC_LIB_MUNMAP_FAILED      (failed to un-map memory)
# CCRTNGFC_LIB_NOT_MAPPED        (memory not mapped)
# CCRTNGFC_LIB_MSGDMA_IN_USE      (modular scatter-gather DMA in use)
*****/

```

2.2.135 ccrtNGFC_NanoDelay()

This call goes into a tight loop spinning for the requested nano seconds specified by the user.

```

/*****
void
ccrtNGFC_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
*****/

```

2.2.136 ccrtNGFC_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..39] that is associated with a physical card. There must exist a character special file */dev/ccrtngfc<Board_Number>* for the call to be successful. One character special file is created for each board found when the driver is successfully loaded.

The *oflag* is the flag supplied to the *open(2)* system call by this API. It is normally '0' (*zero*), however the user may use the *O_NONBLOCK* option for *read(2)* calls which will change the default reading in block mode.

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

In case of error, *errno* is also set for some non-system related errors encountered.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_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: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_INVALID_ARG   (invalid argument)
        # CCRTNGFC_LIB_ALREADY_OPEN  (device already opened)
        # CCRTNGFC_LIB_OPEN_FAILED   (device open failed)
        # CCRTNGFC_LIB_ALREADY_MAPPED (memory already mapped)
*****/

```

```

# CCRTNGFC_LIB_MMAP_SELECT_FAILED    (mmap selection failed)
# CCRTNGFC_LIB_MMAP_FAILED          (mmap failed)
*****/

```

2.2.137 ccrtNGFC_Pause_UserProcess()

This call causes a running User Process to sleep for user specified micro-seconds. (*This is an experimental API for debugging and testing*).

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_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:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR      (successful)
         # CCRTNGFC_LIB_BAD_HANDLE   (no/bad handler supplied)
*****/

```

2.2.138 ccrtNGFC_PowerModule_Control()

This call issues some control commands to the power module.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_PowerModule_Control (void          *Handle,
                              _ccrtngfc_fpga_pm_command_t Command)

Description: Control Power Module

Input:   void          *Handle      (Handle pointer)
         _ccrtngfc_fpga_pm_command_t Command;
         # CCRTNGFC_PM_CMD_CLEAR_FAULTS
         # CCRTNGFC_PM_CMD_CLEAR_PEAK_VALUES
Output:  none
Return:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR      (successful)
         # CCRTNGFC_LIB_INVALID_ARG   (invalid argument)
         # CCRTNGFC_LIB_BAD_HANDLE   (no/bad handler supplied)
         # CCRTNGFC_LIB_NOT_OPEN     (device not open)
         # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
         # CCRTNGFC_LIB_FPGA_PM_BUSY  (power module busy)
         # CCRTNGFC_LIB_FPGA_PM_FAILURE (power module failure)
*****/

```

2.2.139 ccrtNGFC_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 CCRTNGFC_DEFAULT_INPUT_CLOCK_FREQUENCY value.

The input clock can be one of:

```

CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN0 → 10MHz TCX0 (Temperature Compensated
Oscillator Clock).

```

CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN1 → External Input
 CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN2 → FPGA Supplied
 CCRTNGFC_CG_INPUT_CLOCK_SELECT_INXAXB → Not used

When using this card, the default clock should be set to *CCRTNGFC_CG_INPUT_CLOCK_SELECT_N0* i.e. the 10MHz internal clock.

If the desired output clock frequencies are unable to be computed due to hardware limitation, they may wish to increase the desired tolerance *DesiredTolerancePPT* for the particular clock. Note that this tolerance is only applicable to computing a clock value as close to the desired frequency *DesiredFrequency* and not a representation of the accuracy of the output clocks.

Additionally, the programming could fail if the number of N-Divider resource gets exhausted due to the user selecting several output clocks with widely different output clocks.

/******

```

_ccrtngfc_lib_error_number_t
ccrtngfc_Program_All_Output_Clocks(void          *Handle,
                                   double         InputClockFrequency,
                                   _ccrtngfc_cg_input_clock_select_register_t InputClockSel,
                                   ccrtngfc_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)
	_ccrtngfc_cg_input_clock_select_register_t	InputClockSel	(select input clock)
	# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN0		
	# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN1		
	# CCRTNGFC_CG_INPUT_CLOCK_SELECT_IN2		
	# CCRTNGFC_CG_INPUT_CLOCK_SELECT_INXAXB		
	ccrtngfc_compute_all_output_clocks_t	*AllClocks	(pointer to all Clocks)
	ccrtngfc_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:	ccrtngfc_compute_all_output_clocks_t	*AllClocks	(Pointer to returned output clocks info)
	ccrtngfc_compute_single_output_clock_t	*Clock	(Pointer to returned output clock info)
	_ccrtngfc_clock_generator_output_t	OutputClock	
	# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_0		
	# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_1		
	# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_2		
	# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_3		
	# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_4		
	# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_5	// RESERVED - PLL Clock	
	# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_6		
	# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_7		
	# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_8		
	# CCRTNGFC_CLOCK_GENERATOR_OUTPUT_9	// RESERVED - FeedBack Clock	
	double	InputClockFrequency	
	long double	FrequencyDeviation	
	int	FrequencyFound	


```

long double          ActualFrequency
double              ActualTolerancePPT
__u64               Mdiv_Numerator
__u32               Mdiv_Denominator
__u64               Ndiv_Numerator
__u32               Ndiv_Denominator
__ccrtngfc_cg_outmux_ndiv_select_t
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_0
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_1
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_2
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_3
# CCRTNGFC_CG_OUTPUT_MUX_NDIV_4
__u32               Rdiv_value
__u32               Rdivider
__u32               Pdivider
Return:  __ccrtngfc_lib_error_number_t
# CCRTNGFC_LIB_NO_ERROR          (successful)
# CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN         (device not open)
# CCRTNGFC_LIB_NO_LOCAL_REGION  (local region error)
# CCRTNGFC_LIB_IO_ERROR         (device not ready)
# CCRTNGFC_LIB_N_DIVIDERS_EXCEEDED (number of N-Dividers exceeded)
# CCRTNGFC_LIB_CANNOT_COMPUTE_OUTPUT_FREQ (cannot compute output freq)
# CCRTNGFC_LIB_INVALID_ARG      (invalid argument)
# CCRTNGFC_LIB_CLOCK_GENERATION_FAILED (clock generation failed)
# CCRTNGFC_LIB_NO_RESOURCE      (specified clock is reserved)
*****/

```

2.2.140 ccrtNGFC_Read()

This call performs a programmed I/O driver read of either the ADC *channel registers* or the *FIFO*. Prior to issuing this call, the user needs to set up the desired read mode of operation using the *ccrtNGFC_ADC_Set_Driver_Read_Mode()* with *CCRTNGFC_ADC_PIO_CHANNEL* or *CCRTNGFC_ADC_PIO_FIFO* argument. For *channel register* reads, the size is limited to *CCRTNGFC_MAX_ADC_CHANNELS* words and for *FIFO* reads, it is limited to *CCRTNGFC_ADC_FIFO_DATA_MAX* words.

It basically calls the *read(2)* system call with the exception that it performs necessary *locking* and returns the *errno* returned from the system call in the pointer to the *error* variable. An *errno* of *ENOBUS* can occur for *FIFO* reads when it encounters an overflow condition.

For specific information about the data being returned for the various read modes, refer to the *read(2)* system call description the *Driver Direct Access* section.

```

/*****
__ccrtngfc_lib_error_number_t
ccrtNGFC_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: __ccrtngfc_lib_error_number_t

```

```

# CCRTNGFC_LIB_NO_ERROR      (successful)
# CCRTNGFC_LIB_BAD_HANDLE   (no/bad handler supplied)
# CCRTNGFC_LIB_NOT_OPEN     (device not open)
# CCRTNGFC_LIB_IO_ERROR     (read failed)
*****/

```

2.2.141 ccrtNGFC_Reload_Firmware()

The purpose of this call is to power cycle the board which in turn will reload the latest firmware on the board.

```

/*****
ccrtNGFC_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:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR      (successful)
         # CCRTNGFC_LIB_BAD_HANDLE   (no/bad handler supplied)
         # CCRTNGFC_LIB_NOT_OPEN     (device not open)
         # CCRTNGFC_LIB_IOCTL_FAILED (driver ioctl call failed)
*****/

```

2.2.142 ccrtNGFC_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.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_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:  _ccrtngfc_lib_error_number_t
         # CCRTNGFC_LIB_NO_ERROR      (successful)
         # CCRTNGFC_LIB_BAD_HANDLE   (no/bad handler supplied)
         # CCRTNGFC_LIB_NOT_OPEN     (device not open)
         # CCRTNGFC_LIB_IOCTL_FAILED (driver ioctl call failed)
*****/

```

2.2.143 ccrtNGFC_Reset_Board()

This call resets the board to a known hardware state. It may be a good idea to start an application by first resetting the board so that it is set to a known state.

```

/*****

```

```
_ccrtngfc_lib_error_number_t
ccrtNGFC_Reset_Board (void *Handle)
```

Description: Reset the board.

```
Input:  void *Handle          (Handle pointer)
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN      (device not open)
        # CCRTNGFC_LIB_IOCTL_FAILED  (driver ioctl call failed)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
*****/
```

2.2.144 ccrtNGFC_Reset_Clock()

This call performs a hardware reset of the clock. All active output clocks are stopped and set to default state. The user can activate clocks if they wish after a reset via the *activate* argument.

```
/******
_ccrtngfc_lib_error_number_t
ccrtNGFC_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: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN      (device not open)
        # CCRTNGFC_LIB_IOCTL_FAILED  (driver ioctl call failed)
        # CCRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
*****/
```

2.2.145 ccrtNGFC_Resume_UserProcess()

Use this call to resume an already paused User Process. (*This is an experimental API for debugging and testing.*)

```
/******
_ccrtngfc_lib_error_number_t
ccrtNGFC_Resume_UserProcess(void *UFuncHandle)

Description: Resume paused running user process

Input:  void          *UFuncHandle (UF Handle pointer)
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR      (successful)
        # CCRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
*****/
```

2.2.146 ccrtNGFC_Return_Board_Info_Description()

Return board information description

```
/******
char *
```

ccrtNGFC_Return_Board_Info_Description (_ccrtngfc_board_function_t BoardFunction)

Description: Return Board Information Description

Input: _ccrtngfc_board_function_t BoardFunction (board function)
 # CCRTNGFC_BOARD_FUNCTION_MULTIFUNCTION_IO
 # CCRTNGFC_BOARD_FUNCTION_ENGINE_CONTROL
 # CCRTNGFC_BOARD_FUNCTION_BASE_LEVEL
 # CCRTNGFC_BOARD_FUNCTION_CUSTOM_IPCORE
 # CCRTNGFC_BOARD_FUNCTION_CONFIGURABLE_MFIO
 # CCRTNGFC_BOARD_FUNCTION_UNDEFINED
Output: none
Return: char *BoardFuncDesc (board function description)
*****/

2.2.147 ccrtNGFC_Set_Board_CSR()

This call is used to set the board control register.

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtNGFC_Set_Board_CSR (void                   *Handle,  
                          ccrtngfc_board_csr_t   *bcsr)  
  
Description: Set Board Control and Status information  
  
Input:    void                           *Handle (Handle pointer)  
          ccrtngfc_board_csr_t           *bcsr (pointer to board csr)  
          _ccrtngfc_bcsr_identify_board_t identify_board  
          # CCRTNGFC_BCSR_IDENTIFY_BOARD_DISABLE  
          # CCRTNGFC_BCSR_IDENTIFY_BOARD_ENABLE  
  
Output:   none  
Return:   _ccrtngfc_lib_error_number_t  
          # CCRTNGFC_LIB_NO_ERROR                   (successful)  
          # CCRTNGFC_LIB_BAD_HANDLE               (no/bad handler supplied)  
          # CCRTNGFC_LIB_NOT_OPEN                 (device not open)  
          # CCRTNGFC_LIB_INVALID_ARG             (invalid argument)  
          # CCRTNGFC_LIB_NO_LOCAL_REGION         (local region not present)  
*****/
```

2.2.148 ccrtNGFC_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.

```
/******  
_ccrtngfc_lib_error_number_t  
ccrtNGFC_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:   _ccrtngfc_lib_error_number_t  
          # CCRTNGFC_LIB_NO_ERROR                   (successful)  
          # CCRTNGFC_LIB_BAD_HANDLE               (no/bad handler supplied)  
          # CCRTNGFC_LIB_NOT_OPEN                 (device not open)
```

```

# CCRTNGFC_LIB_INVALID_ARG (invalid argument)
*****/

```

2.2.149 ccrtNGFC_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.

```

/*****

```

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_Set_Value (void          *Handle,
                   CCRTNGFC_CONTROL cmd,
                   void          *value)

```

Description: Set the value of the specified board register.

```

Input:  void          *Handle      (Handle pointer)
        CCRTNGFC_CONTROL cmd      (register definition)
        -- structure in ccrtngfc_lib.h
        void          *value      (pointer to value to be set)
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR    (successful)
        # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN   (device not open)
        # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
*****/

```

2.2.150 ccrtNGFC_Transfer_Data()

This is the main call that the user can use to transfer data from physical memory that the user has previously allocated to a region in the local register, and vice-versa. The operation can be performed via DMA or programmed I/O mode. In the case of DMA mode, the user can select whether interrupts are to be used to wait for DMA to complete instead of polling. User can also specify which DMA engine to use during this operation.

If the board supports modular scatter-gather DMA, then the user can specify that instead of the basic DMA engine. In this case, the user needs to first call the *ccrtNGFC_MsgDma_Configure_Single()* with the *NULL* argument to setup descriptor ID-1 for scatter-gather DMA operation.

There are currently six *MsgDma* engines available to the user. They can be selected via the *MsgDmaEngine* option. Since the first four *MsgDma* engines perform quad-word transfers, they must be aligned on a quad-word boundary and the size must be multiples of four, however they are slightly faster than the remaining two that perform single-word transfers. They can be one of the following values:

- CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
- CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
- CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers

The normal process to setup a *MsgDma* operation is to first call the *ccrtNGFC_MsgDma_Seize()* API with either a specific *MsgDma* engine or the optional argument *CCRTNGFC_MSGDMA_ENGINE_IDLE*. In this case, the first available *MsgDma* engine is returned to the user. This is the *MsgDma* engine number that needs to be used for all the *MsgDma* operations until the *MsgDma* is freed via the *ccrtNGFC_MsgDma_Release()* API.

There are certain limitations to modular scatter-gather feature:

1. Scatter-gather DMA is only supported in certain cards
2. Invalid memory address supplied could result in the scatter-gather IP to lock up and the only way to recover will be to reload the driver or reboot the system.
3. Read and write addresses must be at a minimum full-word aligned and for maximum performance, it is recommended to be quad-word aligned.
4. Lengths are in bytes and must be at a minimum a multiple of a full-word and for maximum performance, it is recommended to be quad-word multiple.
5. Scatter-gather chaining cannot be performed with this call.

/******

```

_ccrtngfc_lib_error_number_t
ccrtNGFC_Transfer_Data(void          *Handle,
                        volatile void *PciDmaMemory,
                        volatile void *AvalonMem,
                        uint           TransferSize,
                        _ccrtngfc_direction_t XferDirection,
                        _ccrtngfc_library_rw_mode_t LibMode,
                        _ccrtngfc_msgdma_engine_t MsgDmaEngine,
                        ccrtngfc_bool UseInterrupts,
                        int           IoControl)

```

Description: Routine to transfer data from PCI memory to Avalon memory or vice-versa

```

Input:  void          *Handle          (Handle pointer)
        volatile void *PciDmaMemory  (pointer to virtual memory)
        volatile void *AvalonMem     (pointer to virtual Avalon memory)
        uint          TransferSize    (size of transfer in bytes)
        _ccrtngfc_direction_t XferDirection (direction of transfer)
        # CCRTNGFC_AVALON_2_PCIMEM
        # CCRTNGFC_PCIMEM_2_AVALON
        _ccrtngfc_library_rw_mode_t LibMode (Lib transfer mode)
        # CCRTNGFC_LIBRARY_PIO_MODE
        # CCRTNGFC_LIBRARY_MSGDMA_MODE
        _ccrtngfc_msgdma_engine_t MsgDmaEngine
        # CCRTNGFC_MSGDMA_ENGINE_0 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_1 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_2 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_3 // quad-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_4 // single-word transfers
        # CCRTNGFC_MSGDMA_ENGINE_5 // single-word transfers
        ccrtngfc_bool UseInterrupts (enable interrupts)
        # CCRTNGFC_TRUE
        # CCRTNGFC_FALSE
        int          IoControl        (DMA or PIO control flags)
        # CCRTNGFC_PIO_CONTROL_RCON (PIO read constant)
        # CCRTNGFC_PIO_CONTROL_WCON (PIO write constant)
        # CCRTNGFC_PIO_CONTROL_INCREMENT (PIO increment)
Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR (no error)
        # CCRTNGFC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN (library not open)
        # CCRTNGFC_LIB_INVALID_ARG (invalid argument)
        # CCRTNGFC_LIB_IOCTL_FAILED (driver ioctl call failed)
        # CCRTNGFC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA not supported)
        # CCRTNGFC_LIB_MSGDMA_READS_NOT_ALLOWED_FOR_SELECTED_ADDRESS (MSG DMA Reads not allowed for

```

```

selected address)
# CCRTNGFC_LIB_NOT_OWNER_OF_MSGDMA (not owner of modular scatter-gather)
# CCRTNGFC_LIB_DATA_WIDTH_NOT_MULTIPLE_OR_ALIGNED
(Data Width Not Multiple or Aligned)
*****/

```

2.2.151 ccrtNGFC_Update_Clock_Generator_Divider()

Update the selected clock generator divider so that its changes take affect. *Normally, this call should not be used. It is recommended that only advanced users should use this call and with extreme care and intimate knowledge of the clock programming, otherwise results would be indeterminate.*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Update_Clock_Generator_Divider (void          *Handle,
                                         _ccrtngfc_clock_generator_divider_t  WhichDivider)

Description: Update Clock Generator Divider

Input:  void          *Handle          (Handle pointer)
        _ccrtngfc_clock_generator_divider_t  WhichDivider  (select divider)
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_M
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_N0
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_N1
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_N2
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_N3
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_N_ALL
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P0
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P1
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P2
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_PFB
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_P_ALL
        # CCRTNGFC_CLOCK_GENERATOR_DIVIDER_PXAXB

Output: none
Return: _ccrtngfc_lib_error_number_t
        # CCRTNGFC_LIB_NO_ERROR          (successful)
        # CCRTNGFC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTNGFC_LIB_NOT_OPEN        (library not open)
        # CCRTNGFC_LIB_NO_LOCAL_REGION  (local region error)
        # CCRTNGFC_LIB_INVALID_ARG     (invalid argument)
        # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.152 ccrtNGFC_UserProcess_Command()

The user can control the execution of the created User Process with the help of this call. *(This is an experimental API for debugging and testing).*

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_UserProcess_Command(void          *Handle,
                              void          *UFuncHandle,
                              ccrtngfc_uf_action_t  Action)

Description: Command User process

Input:  void          *UFuncHandle  (User Process Handle pointer)
        ccrtngfc_uf_action_t  Action  (command action)
        # CCRTNGFC_UF_ACTION_STOP
        # CCRTNGFC_UF_ACTION_RUN

```

```

        # CCRTNGFC_UF_ACTION_TERMINATE
Output: none
Return: none
*****/

```

2.2.153 ccrtNGFC_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
ccrtNGFC_VoltsToData (double          volts,
                    ccrtngfc_volt_convert_t *conv)

Description: Convert Volts to data

Input:  double          volts          (volts to convert)
        ccrtngfc_volt_convert_t *conv  (pointer to conversion struct)
        double          VoltageRange  (maximum voltage range)
        _ccrtngfc_csr_dataformat_t Format (format)
        # CCRTNGFC_OFFSET_BINARY
        # CCRTNGFC_TWOS_COMPLEMENT
        ccrtngfc_bool    BiPolar      (bi-polar)
        # CCRTNGFC_TRUE
        # CCRTNGFC_FALSE
        int              ResolutionBits (Number of resolution bits)
Output: none
Return: uint          data          (returned data)
*****/

```

2.2.154 ccrtNGFC_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.

```

/*****
_ccrtngfc_lib_error_number_t
ccrtNGFC_Wait_For_Interrupt (void          *Handle,
                            ccrtngfc_driver_int_t *drv_int)

Description: Wait For Interrupt

Input:  void          *Handle          (Handle pointer)
        ccrtngfc_driver_int_t *drv_int (pointer to drv_int struct)
        uint          WakeupInterruptMask
        # CCRTNGFC_ALL_MSGDMA_INTMASK
        # CCRTNGFC_MSGDMA0_INTMASK
        # CCRTNGFC_MSGDMA1_INTMASK
        # CCRTNGFC_MSGDMA2_INTMASK
        # CCRTNGFC_MSGDMA3_INTMASK
        # CCRTNGFC_MSGDMA4_INTMASK
        # CCRTNGFC_MSGDMA5_INTMASK
        # CCRTNGFC_ALL_ANALOG_INTMASK
        # CCRTNGFC_ADC0_FIFO_INTMASK
        # CCRTNGFC_DAC0_FIFO_INTMASK
        # CCRTNGFC_ADC1_FIFO_INTMASK
        # CCRTNGFC_DAC1_FIFO_INTMASK
        int          timeout_seconds

```



```

Output:  ccrtnngfc_driver_int_t  *drv_int      (pointer to drv_int struct)
         long long unsigned      count
         long long unsigned      MsgDma_count[CCRRTNGFC_MAX_MSGDMA_ENGINES]
         uint                    InterruptsOccurredMask
         uint                    WakeupInterruptMask

Return:  _ccrtnngfc_lib_error_number_t
         # CCRRTNGFC_LIB_NO_ERROR      (successful)
         # CCRRTNGFC_LIB_BAD_HANDLE    (no/bad handler supplied)
         # CCRRTNGFC_LIB_NOT_OPEN      (device not open)
         # CCRRTNGFC_LIB_NO_LOCAL_REGION (local region not present)
         # CCRRTNGFC_LIB_INVALID_ARG   (invalid argument)
*****/

```

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 ccrtnngfc_disp

Useful program to display the local board registers. This program uses the *curses* library.

```
Usage: ./ccrtnngfc_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:

```
./ccrtnngfc_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 93200101 04172023 00400000 00120001 00000000 00000000 00000020 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 ccrtnngfc_dump

This test is for debugging purpose. It dumps all the hardware registers.

Usage: ccrtnngfc_dump [-b board]
 -b board: board number -- default board is 0

Example display:

```
./ccrtnngfc_dump
```

```
Device Name: /dev/ccrtnngfc0
```

```

LOCAL REGION: Physical Addr=0xfbe00000 Size=524288 (0x00080000)
CONFIG REGION: Physical Addr=0xfbe80000 Size=32768 (0x00080000)

```

```

LOCAL: Register 0x7ffff7f55000 Offset=0x0 Size=0x00080000
CONFIG: Register 0x7ffff7fef000 Offset=0x0 Size=0x00080000
LIBPTR: Register 0x7ffff7fde000 Offset=0x0 Size=0x000105a0

```

```
===== LOCAL BOARD REGISTERS =====
```

```

LBR: @0x000000 --> 0x93200101
LBR: @0x000004 --> 0x04172023
LBR: @0x000008 --> 0x00400000
LBR: @0x00000c --> 0x00120001
LBR: @0x000010 --> 0x00000000
LBR: @0x000014 --> 0x00000000
LBR: @0x000018 --> 0x00000020
LBR: @0x00001c --> 0x00000000
LBR: @0x000020 --> 0x00000000
LBR: @0x000024 --> 0x00000000
LBR: @0x000028 --> 0x00000000
LBR: @0x00002c --> 0x00000000
LBR: @0x000030 --> 0x00000000
.
.
.
LBR: @0x07ffc --> 0x00000000
LBR: @0x07ffd0 --> 0x00000000
LBR: @0x07ffd4 --> 0x00000000
LBR: @0x07ffd8 --> 0x00000000
LBR: @0x07ffdc --> 0x00000000
LBR: @0x07ffe0 --> 0x00000000
LBR: @0x07ffe4 --> 0x00000000
LBR: @0x07ffe8 --> 0x00000000
LBR: @0x07ffec --> 0x00000000
LBR: @0x07fff0 --> 0x00000000
LBR: @0x07fff4 --> 0x00000000
LBR: @0x07fff8 --> 0x00000000
LBR: @0x07fffc --> 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: @0x1034 --> 0x00000000
LCR: @0x1038 --> 0x00000000
LCR: @0x103c --> 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

=====
===== PCI CONFIG REG ADDR MAPPING =====

PCR: @0x0000 --> 0x93201542
PCR: @0x0004 --> 0x00100546
PCR: @0x0008 --> 0x08800001
PCR: @0x000c --> 0x00000008
PCR: @0x0010 --> 0xfbe80000
PCR: @0x0014 --> 0x00000000
PCR: @0x0018 --> 0xfbe00000
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 --> 0x00000000
PCR: @0x0048 --> 0x00000000
PCR: @0x004c --> 0x00000000
PCR: @0x0050 --> 0x00857805
PCR: @0x0054 --> 0xfe000078
PCR: @0x0058 --> 0x00000000
PCR: @0x005c --> 0x00000000
PCR: @0x0060 --> 0x00000000
PCR: @0x0064 --> 0x00000000
PCR: @0x0068 --> 0x00007811
PCR: @0x006c --> 0x00000000
PCR: @0x0070 --> 0x00000000
PCR: @0x0074 --> 0x00000000
PCR: @0x0078 --> 0x00038001
PCR: @0x007c --> 0x00000000
PCR: @0x0080 --> 0x00020010

```

PCR: @0x0084 --> 0x00008001
PCR: @0x0088 --> 0x00002834
PCR: @0x008c --> 0x01406042
PCR: @0x0090 --> 0x10420000
PCR: @0x0094 --> 0x00000000
PCR: @0x0098 --> 0x00000000
PCR: @0x009c --> 0x00000000
PCR: @0x00a0 --> 0x00000000
PCR: @0x00a4 --> 0x0010001f
PCR: @0x00a8 --> 0x00000000
PCR: @0x00ac --> 0x00000006
PCR: @0x00b0 --> 0x00010002
PCR: @0x00b4 --> 0x00000000
PCR: @0x00b8 --> 0x00000000
PCR: @0x00bc --> 0x00000000
PCR: @0x00c0 --> 0x00000000
PCR: @0x00c4 --> 0x00000000
PCR: @0x00c8 --> 0x00000000
PCR: @0x00cc --> 0x00000000
PCR: @0x00d0 --> 0x00000000
PCR: @0x00d4 --> 0x00000000
PCR: @0x00d8 --> 0x00000000
PCR: @0x00dc --> 0x00000000
PCR: @0x00e0 --> 0x00000000
PCR: @0x00e4 --> 0x00000000
PCR: @0x00e8 --> 0x00000000
PCR: @0x00ec --> 0x00000000
PCR: @0x00f0 --> 0x00000000
PCR: @0x00f4 --> 0x00000000
PCR: @0x00f8 --> 0x00000000
PCR: @0x00fc --> 0x00000000

```

3.1.3 ccrtngfc_rdreg

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

```

Usage: ./ccrtngfc_rdreg [-b Board] [-C] [-f] [-o Offset] [-s Size]
-b Board   : Board number -- default board is 0
-C         : Select Config Registers instead of Local Registers
-f         : Fast Memory Reads
-o Offset  : Hex offset to read from -- default offset is 0x0
-s Size    : Number of bytes to read in decimal -- default size is 0x4

```

Example display:

```
./ccrtngfc_rdreg -s64
```

```
Device Name: /dev/ccrtngfc0
```

```

LOCAL REGION: Physical Addr=0xfbe00000 Size=524288 (0x00080000)
CONFIG REGION: Physical Addr=0xfbe80000 Size=32768 (0x00080000)

```

```

LOCAL: Register 0x7ffff7f55000 Offset=0x0 Size=0x00080000
CONFIG: Register 0x7ffff7fef000 Offset=0x0 Size=0x00080000
LIBPTR: Register 0x7ffff7fde000 Offset=0x0 Size=0x000105a0

```

```

#### LOCAL REGS #### (length=64)
+LCL+      0  93200101  04172023  00400000  00120001  *. .... #.@.....*
+LCL+     0x10  00000000  00000000  00000020  00000000  *.....*

```

```
+LCL+    0x20    00000000    00000000    00000000    00000000    *.*****
+LCL+    0x30    00000000    00000000    00000000    00000000    *.*****
      28.978us (    2.21 MB/s)
```

```
./ccrtnngfc_rdreg -C -o4020 -s20
```

```
Device Name: /dev/ccrtnngfc0
```

```
LOCAL REGION: Physical Addr=0xfbe00000 Size=524288 (0x00080000)
CONFIG REGION: Physical Addr=0xfbe80000 Size=32768 (0x00080000)
```

```
LOCAL: Register 0x7ffff7f55000 Offset=0x0 Size=0x00080000
CONFIG: Register 0x7ffff7fef000 Offset=0x0 Size=0x00080000
LIBPTR: Register 0x7ffff7fde000 Offset=0x0 Size=0x000105a0
```

```
#### CONFIG REGS #### (length=20)
+CFG+    0x4020    00000000    00000000    00000000    00000000    *.*****
+CFG+    0x4030    00000000                                *.*****
      8.056us (    2.48 MB/s)
```

3.1.4 ccrtnngfc_reg

This call displays all the boards local and configuration registers.

```
Usage: ./ccrtnngfc_reg [-b board]
      -b board: Board number -- default board is 0
```

Example display:

```
./ccrtnngfc_reg
```

```
Device Name: /dev/ccrtnngfc0
```

```
LOCAL REGION: Physical Addr=0xfb800000 Size=4194304 (0x00400000)
CONFIG REGION: Physical Addr=0xfbc00000 Size=32768 (0x00080000)
```

```
LOCAL: Register 0x7ffff7800000 Offset=0x0 Size=0x00400000
CONFIG: Register 0x7ffff7fb7000 Offset=0x0 Size=0x00080000
LIBPTR: Register 0x7ffff7ebf000 Offset=0x0 Size=0x00013e58
LOCAL Register 0x7ffff7800000 size=0x00400000
```

```
#### LOCAL REGS #### (length=4194304)
+LCL+      0    93200101    08252023    00010000    00120000    *. ...% #.....*
+LCL+    0x10    00000000    00000000    00000020    00010001    *.*****
+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+ 0x3fff00 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fff10 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fff20 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fff30 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fff40 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fff50 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fff60 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fff70 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fff80 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fff90 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fffa0 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fffb0 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fffc0 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fffd0 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3fffe0 00000000 00000000 00000000 00000000 *.
+LCL+ 0x3ffff0 00000000 00000000 00000000 00000000 *.

#### 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+ 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+ 0x10e0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x10f0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1100 00000000 00000000 00000000 00000000 *.....*

```

```

.
.
.

```

```

+CFG+ 0x1f00 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1f10 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1f20 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1f30 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1f40 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 *.....*

```

==== LOCAL REGISTERS =====

```

BoardInfo                =0x93200101    @0x00000000
FirmwareDate              =0x08252023    @0x00000004
FirmwareRevision          =0x00010000    @0x00000008
FirmwareTime              =0x00120000    @0x0000000c
FirmwareFlavorCode        =0x00000000    @0x00000010
NumberAdvancedIPCores     =0x00000000    @0x00000014
NumberMsgDmaDescriptors   =0x00000020    @0x00000018
BoardCSR                  =0x00000000    @0x00002000
InterruptStatus           =0x00000000    @0x00002010
SPI_CommandStatus         =0x03004000    @0x000020f0
SPI_FirmwareAddress       =0x03ffff024   @0x000020f4

```

SPI_Ram[0]	=0x80c02000	@0x00002100
FPGA_ChipIdentification[0]	=0x0059e200	@0x00002400
FPGA_ChipIdentification[1]	=0x08a2f910	@0x00002404
FPGA_ChipTemperature	=0x000001c8	@0x00002410
FPGA_VoltageMonitoring_ADC.VS1_8	=0x0000001e	@0x00002420
FPGA_VoltageMonitoring_ADC.VS3_3	=0x00000038	@0x00002424
FPGA_VoltageMonitoring_ADC.VCC	=0x0000002e	@0x00002428
FPGA_VoltageMonitoring_ADC.VCCP	=0x0000002e	@0x0000242c
FPGA_VoltageMonitoring_ADC.VCCPT	=0x0000002e	@0x00002430
FPGA_VoltageMonitoring_ADC.VCCERAM	=0x0000002e	@0x00002434
FPGA_VoltageMonitoring_ADC.VCCL_HPS	=0x00000000	@0x00002438
FPGA_VoltageMonitoring_ADC.ADCGND	=0x00000000	@0x0000243c
ClockGen_CSR	=0x00000001	@0x00002500
ClockGen_access	=0x00000000	@0x00002504

===== Daughter Card ADC 0 REGISTERS =====

ADC_Enable	=0x00000001	@0x00000000
ADC_ControlStatus[CCRTNGFC_ADC_0]	=0x00000020	@0x00000010
ADC_ControlStatus[CCRTNGFC_ADC_1]	=0x00000020	@0x00000014
ADC_ControlStatus[CCRTNGFC_ADC_2]	=0x00000020	@0x00000018
ADC_FifoCSR	=0x81000000	@0x00000030
ADC_FifoThreshold	=0x00020000	@0x00000034
ADC_FifoChannelSelect	=0x0000ffff	@0x00000038
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_0]	=0x80000000	@0x00000100
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_1]	=0x80000000	@0x00000104
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_2]	=0x80000000	@0x00000108
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_3]	=0x80000000	@0x0000010c
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_4]	=0x80000000	@0x00000110
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_5]	=0x80000000	@0x00000114
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_6]	=0x80000000	@0x00000118
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_7]	=0x80000000	@0x0000011c
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_8]	=0x80000000	@0x00000120
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_9]	=0x80000000	@0x00000124
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_10]	=0x80000000	@0x00000128
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_11]	=0x80000000	@0x0000012c
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_0]	=0x80000000	@0x00000140
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_1]	=0x80000000	@0x00000144
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_2]	=0x80000000	@0x00000148
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_3]	=0x80000000	@0x0000014c
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_4]	=0x80000000	@0x00000150
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_5]	=0x80000000	@0x00000154
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_6]	=0x80000000	@0x00000158
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_7]	=0x80000000	@0x0000015c
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_8]	=0x80000000	@0x00000160
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_9]	=0x80000000	@0x00000164
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_10]	=0x80000000	@0x00000168
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_11]	=0x80000000	@0x0000016c
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_0]	=0x00000000	@0x00000180
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_1]	=0x00000000	@0x00000184
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_2]	=0x00000000	@0x00000188
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_3]	=0x00000000	@0x0000018c
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_4]	=0x00000000	@0x00000190
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_5]	=0x00000000	@0x00000194
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_6]	=0x00000000	@0x00000198
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_7]	=0x00000000	@0x0000019c
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_8]	=0x00000000	@0x000001a0
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_9]	=0x00000000	@0x000001a4
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_10]	=0x00000000	@0x000001a8
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_11]	=0x00000000	@0x000001ac
ADC_Data[CCRTNGFC_ADC_CHANNEL_0]	=0x00000000	@0x00000200
ADC_Data[CCRTNGFC_ADC_CHANNEL_1]	=0x00000000	@0x00000204

```

ADC_Data[CCRTNGFC_ADC_CHANNEL_2]           =0x00000000      @0x00000208
ADC_Data[CCRTNGFC_ADC_CHANNEL_3]           =0x00000000      @0x0000020c
ADC_Data[CCRTNGFC_ADC_CHANNEL_4]           =0x00000000      @0x00000210
ADC_Data[CCRTNGFC_ADC_CHANNEL_5]           =0x00000000      @0x00000214
ADC_Data[CCRTNGFC_ADC_CHANNEL_6]           =0x00000000      @0x00000218
ADC_Data[CCRTNGFC_ADC_CHANNEL_7]           =0x00000000      @0x0000021c
ADC_Data[CCRTNGFC_ADC_CHANNEL_8]           =0x00000000      @0x00000220
ADC_Data[CCRTNGFC_ADC_CHANNEL_9]           =0x00000000      @0x00000224
ADC_Data[CCRTNGFC_ADC_CHANNEL_10]          =0x00000000      @0x00000228
ADC_Data[CCRTNGFC_ADC_CHANNEL_11]          =0x00000000      @0x0000022c

```

===== Daughter Card ADC 1 REGISTERS =====

```

ADC_Enable                                   =0x00000001      @0x00000000
ADC_ControlStatus[CCRTNGFC_ADC_0]          =0x00000020      @0x00000010
ADC_ControlStatus[CCRTNGFC_ADC_1]          =0x00000020      @0x00000014
ADC_ControlStatus[CCRTNGFC_ADC_2]          =0x00000020      @0x00000018
ADC_FifoCSR                                  =0x81000000      @0x00000030
ADC_FifoThreshold                            =0x00020000      @0x00000034
ADC_FifoChannelSelect                       =0x0000ffff      @0x00000038
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_0] =0x80000000      @0x00000100
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_1] =0x80000000      @0x00000104
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_2] =0x80000000      @0x00000108
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_3] =0x80000000      @0x0000010c
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_4] =0x80000000      @0x00000110
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_5] =0x80000000      @0x00000114
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_6] =0x80000000      @0x00000118
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_7] =0x80000000      @0x0000011c
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_8] =0x80000000      @0x00000120
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_9] =0x80000000      @0x00000124
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_10] =0x80000000      @0x00000128
ADC_PositiveCalibration[CCRTNGFC_ADC_CHANNEL_11] =0x80000000      @0x0000012c
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_0] =0x80000000      @0x00000140
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_1] =0x80000000      @0x00000144
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_2] =0x80000000      @0x00000148
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_3] =0x80000000      @0x0000014c
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_4] =0x80000000      @0x00000150
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_5] =0x80000000      @0x00000154
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_6] =0x80000000      @0x00000158
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_7] =0x80000000      @0x0000015c
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_8] =0x80000000      @0x00000160
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_9] =0x80000000      @0x00000164
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_10] =0x80000000      @0x00000168
ADC_NegativeCalibration[CCRTNGFC_ADC_CHANNEL_11] =0x80000000      @0x0000016c
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_0] =0x00000000      @0x00000180
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_1] =0x00000000      @0x00000184
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_2] =0x00000000      @0x00000188
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_3] =0x00000000      @0x0000018c
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_4] =0x00000000      @0x00000190
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_5] =0x00000000      @0x00000194
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_6] =0x00000000      @0x00000198
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_7] =0x00000000      @0x0000019c
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_8] =0x00000000      @0x000001a0
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_9] =0x00000000      @0x000001a4
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_10] =0x00000000      @0x000001a8
ADC_OffsetCalibration[CCRTNGFC_ADC_CHANNEL_11] =0x00000000      @0x000001ac
ADC_Data[CCRTNGFC_ADC_CHANNEL_0]           =0x00000000      @0x00000200
ADC_Data[CCRTNGFC_ADC_CHANNEL_1]           =0x00000000      @0x00000204
ADC_Data[CCRTNGFC_ADC_CHANNEL_2]           =0x00000000      @0x00000208
ADC_Data[CCRTNGFC_ADC_CHANNEL_3]           =0x00000000      @0x0000020c
ADC_Data[CCRTNGFC_ADC_CHANNEL_4]           =0x00000000      @0x00000210
ADC_Data[CCRTNGFC_ADC_CHANNEL_5]           =0x00000000      @0x00000214

```

ADC_Data[CCRTNGFC_ADC_CHANNEL_6]	=0x00000000	@0x00000218
ADC_Data[CCRTNGFC_ADC_CHANNEL_7]	=0x00000000	@0x0000021c
ADC_Data[CCRTNGFC_ADC_CHANNEL_8]	=0x00000000	@0x00000220
ADC_Data[CCRTNGFC_ADC_CHANNEL_9]	=0x00000000	@0x00000224
ADC_Data[CCRTNGFC_ADC_CHANNEL_10]	=0x00000000	@0x00000228
ADC_Data[CCRTNGFC_ADC_CHANNEL_11]	=0x00000000	@0x0000022c

===== Daughter Card DAC 0 REGISTERS =====

DAC_Enable	=0x00000000	@0x00000000
DAC_UpdateSelect	=0x00000000	@0x00000004
DAC_ChannelSelect	=0x00000000	@0x00000008
DAC_ControlStatus[CCRTNGFC_DAC_0]	=0x00000000	@0x00000010
DAC_ControlStatus[CCRTNGFC_DAC_1]	=0x00000000	@0x00000014
DAC_ControlStatus[CCRTNGFC_DAC_2]	=0x00000000	@0x00000018
DAC_ControlStatus[CCRTNGFC_DAC_3]	=0x00000000	@0x0000001c
DAC_ControlStatus[CCRTNGFC_DAC_4]	=0x00000000	@0x00000020
DAC_ControlStatus[CCRTNGFC_DAC_5]	=0x00000000	@0x00000024
DAC_FifoCSR	=0x00000000	@0x00000030
DAC_FifoThreshold	=0x00000000	@0x00000034
DAC_FifoWriteCount	=0x00000000	@0x00000038
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_0]	=0x00000000	@0x00000140
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_1]	=0x00000000	@0x00000144
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_2]	=0x00000000	@0x00000148
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_3]	=0x00000000	@0x0000014c
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_4]	=0x00000000	@0x00000150
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_5]	=0x00000000	@0x00000154
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_6]	=0x00000000	@0x00000158
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_7]	=0x00000000	@0x0000015c
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_8]	=0x00000000	@0x00000160
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_9]	=0x00000000	@0x00000164
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_10]	=0x00000000	@0x00000168
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_11]	=0x00000000	@0x0000016c
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_0]	=0x00000000	@0x00000180
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_1]	=0x00000000	@0x00000184
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_2]	=0x00000000	@0x00000188
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_3]	=0x00000000	@0x0000018c
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_4]	=0x00000000	@0x00000190
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_5]	=0x00000000	@0x00000194
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_6]	=0x00000000	@0x00000198
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_7]	=0x00000000	@0x0000019c
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_8]	=0x00000000	@0x000001a0
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_9]	=0x00000000	@0x000001a4
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_10]	=0x00000000	@0x000001a8
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_11]	=0x00000000	@0x000001ac
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_0]	=0x00000000	@0x00000100
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_1]	=0x00000000	@0x00000104
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_2]	=0x00000000	@0x00000108
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_3]	=0x00000000	@0x0000010c
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_4]	=0x00000000	@0x00000110
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_5]	=0x00000000	@0x00000114
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_6]	=0x00000000	@0x00000118
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_7]	=0x00000000	@0x0000011c
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_8]	=0x00000000	@0x00000120
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_9]	=0x00000000	@0x00000124
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_10]	=0x00000000	@0x00000128
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_11]	=0x00000000	@0x0000012c
DAC_Data[CCRTNGFC_DAC_CHANNEL_0]	=0x00000000	@0x00000200
DAC_Data[CCRTNGFC_DAC_CHANNEL_1]	=0x00000000	@0x00000204
DAC_Data[CCRTNGFC_DAC_CHANNEL_2]	=0x00000000	@0x00000208
DAC_Data[CCRTNGFC_DAC_CHANNEL_3]	=0x00000000	@0x0000020c
DAC_Data[CCRTNGFC_DAC_CHANNEL_4]	=0x00000000	@0x00000210

DAC_Data[CCRTNGFC_DAC_CHANNEL_5]	=0x00000000	@0x00000214
DAC_Data[CCRTNGFC_DAC_CHANNEL_6]	=0x00000000	@0x00000218
DAC_Data[CCRTNGFC_DAC_CHANNEL_7]	=0x00000000	@0x0000021c
DAC_Data[CCRTNGFC_DAC_CHANNEL_8]	=0x00000000	@0x00000220
DAC_Data[CCRTNGFC_DAC_CHANNEL_9]	=0x00000000	@0x00000224
DAC_Data[CCRTNGFC_DAC_CHANNEL_10]	=0x00000000	@0x00000228
DAC_Data[CCRTNGFC_DAC_CHANNEL_11]	=0x00000000	@0x0000022c

===== Daughter Card DAC 1 REGISTERS =====

DAC_Enable	=0x00000000	@0x00000000
DAC_UpdateSelect	=0x00000000	@0x00000004
DAC_ChannelSelect	=0x00000000	@0x00000008
DAC_ControlStatus[CCRTNGFC_DAC_0]	=0x00000000	@0x00000010
DAC_ControlStatus[CCRTNGFC_DAC_1]	=0x00000000	@0x00000014
DAC_ControlStatus[CCRTNGFC_DAC_2]	=0x00000000	@0x00000018
DAC_ControlStatus[CCRTNGFC_DAC_3]	=0x00000000	@0x0000001c
DAC_ControlStatus[CCRTNGFC_DAC_4]	=0x00000000	@0x00000020
DAC_ControlStatus[CCRTNGFC_DAC_5]	=0x00000000	@0x00000024
DAC_FifoCSR	=0x00000000	@0x00000030
DAC_FifoThreshold	=0x00000000	@0x00000034
DAC_FifoWriteCount	=0x00000000	@0x00000038
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_0]	=0x00000000	@0x00000140
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_1]	=0x00000000	@0x00000144
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_2]	=0x00000000	@0x00000148
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_3]	=0x00000000	@0x0000014c
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_4]	=0x00000000	@0x00000150
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_5]	=0x00000000	@0x00000154
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_6]	=0x00000000	@0x00000158
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_7]	=0x00000000	@0x0000015c
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_8]	=0x00000000	@0x00000160
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_9]	=0x00000000	@0x00000164
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_10]	=0x00000000	@0x00000168
DAC_NegativeCalibration[CCRTNGFC_DAC_CHANNEL_11]	=0x00000000	@0x0000016c
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_0]	=0x00000000	@0x00000180
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_1]	=0x00000000	@0x00000184
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_2]	=0x00000000	@0x00000188
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_3]	=0x00000000	@0x0000018c
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_4]	=0x00000000	@0x00000190
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_5]	=0x00000000	@0x00000194
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_6]	=0x00000000	@0x00000198
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_7]	=0x00000000	@0x0000019c
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_8]	=0x00000000	@0x000001a0
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_9]	=0x00000000	@0x000001a4
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_10]	=0x00000000	@0x000001a8
DAC_OffsetCalibration[CCRTNGFC_DAC_CHANNEL_11]	=0x00000000	@0x000001ac
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_0]	=0x00000000	@0x00000100
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_1]	=0x00000000	@0x00000104
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_2]	=0x00000000	@0x00000108
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_3]	=0x00000000	@0x0000010c
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_4]	=0x00000000	@0x00000110
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_5]	=0x00000000	@0x00000114
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_6]	=0x00000000	@0x00000118
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_7]	=0x00000000	@0x0000011c
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_8]	=0x00000000	@0x00000120
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_9]	=0x00000000	@0x00000124
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_10]	=0x00000000	@0x00000128
DAC_PositiveCalibration[CCRTNGFC_DAC_CHANNEL_11]	=0x00000000	@0x0000012c
DAC_Data[CCRTNGFC_DAC_CHANNEL_0]	=0x00000000	@0x00000200
DAC_Data[CCRTNGFC_DAC_CHANNEL_1]	=0x00000000	@0x00000204
DAC_Data[CCRTNGFC_DAC_CHANNEL_2]	=0x00000000	@0x00000208
DAC_Data[CCRTNGFC_DAC_CHANNEL_3]	=0x00000000	@0x0000020c

```

DAC_Data[CCRTNGFC_DAC_CHANNEL_4]           =0x00000000    @0x00000210
DAC_Data[CCRTNGFC_DAC_CHANNEL_5]           =0x00000000    @0x00000214
DAC_Data[CCRTNGFC_DAC_CHANNEL_6]           =0x00000000    @0x00000218
DAC_Data[CCRTNGFC_DAC_CHANNEL_7]           =0x00000000    @0x0000021c
DAC_Data[CCRTNGFC_DAC_CHANNEL_8]           =0x00000000    @0x00000220
DAC_Data[CCRTNGFC_DAC_CHANNEL_9]           =0x00000000    @0x00000224
DAC_Data[CCRTNGFC_DAC_CHANNEL_10]          =0x00000000    @0x00000228
DAC_Data[CCRTNGFC_DAC_CHANNEL_11]          =0x00000000    @0x0000022c

===== Daughter Card ADC and DAC FIFO REGISTERS =====
M0Adc_Fifo                                  =0xbaadbeef    @0x00010000
M0Dac_Fifo                                  =0x00000000    @0x00014000
M1Adc_Fifo                                  =0xbaadbeef    @0x00018000
M1Dac_Fifo                                  =0x00000000    @0x0001c000

===== LDIO REGISTERS =====
LDIO_Enable                                 =0x00000000    @0x00030000
LDIO_InputSnapshot                          =0x00000000    @0x00030008
LDIO_OutputSync                             =0x00000000    @0x0003000c
DIO_Direction[CCRTNGFC_MAIN_DIO_MODULE_0]  =0x00000000    @0x00030020
DIO_Set_OutputDirection[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00030030
DIO_Set_InputDirection[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00030040
LDIO_OutputChannels[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00031000
LDIO_OutputChannels[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x00031004
LDIO_Set_OutputChannelsHigh[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00031010
LDIO_Set_OutputChannelsHigh[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x00031014
LDIO_Set_OutputChannelsLow[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00031020
LDIO_Set_OutputChannelsLow[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x00031024
LDIO_InputChannels[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00031040
LDIO_InputChannels[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x00031044
LDIO_InputChannelsFilter[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00031050
LDIO_InputChannelsFilter[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x00031054
LDIO_ChannelsPolarity[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00031060
LDIO_ChannelsPolarity[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x00031064
LDIO_COS_ChannelsEnable[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00031070
LDIO_COS_ChannelsEnable[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x00031074
LDIO_COS_ChannelsMode[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00031080
LDIO_COS_ChannelsMode[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x00031084
LDIO_COS_ChannelsEdgeSense[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x00031090
LDIO_COS_ChannelsEdgeSense[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x00031094
LDIO_COS_ChannelsOverflow[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x000310a0
LDIO_COS_ChannelsOverflow[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x000310a4
LDIO_COS_ChannelsStatus[CCRTNGFC_MAIN_DIO_MODULE_0] =0x00000000    @0x000310b0
LDIO_COS_ChannelsStatus[CCRTNGFC_MAIN_LIO_MODULE_1] =0x00000000    @0x000310b4

===== DIAG RAM REGISTERS =====
DiagRam[0]                                  =0x00000000    @0x00080000

===== CONFIG REGISTERS =====
PciLinkPartners.a2p_interrupt_status        =0x00000000    @0x00000040
PciLinkPartners.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  *.....*

AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High =0x00000000    @0x00001004

```

AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001000
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x0000100c
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001008
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x00001014
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001010
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x0000101c
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001018
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x00001024
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001020
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x0000102c
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001028
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x00001034
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001030
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x0000103c
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001038
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x00001044
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001040
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x0000104c
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001048
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x00001054
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001050
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x0000105c
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001058
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x00001064
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001060
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x0000106c
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001068
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x00001074
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001070
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_High	=0x00000000	@0x0000107c
AvalonMM_2_PCIeAddrTrans[Entry].PCI_Physical_Address_Low	=0x00000000	@0x00001078

===== MsgDma Engine 0 =====

=== Dispatcher Csr ===

MsgDmaRegs[Eng].MsgDmaDispatcherCsr.Status	=0x0000000a	@0x00004000
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.Control	=0x00000000	@0x00004004
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.ReadFillLevel	=0x00000000	@0x00004008
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.WriteFillLevel	=0x00000000	@0x0000400a
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.ResponseFillLevel	=0x00000000	@0x0000400c
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.ReadSequenceNumber	=0x00000000	@0x00004010
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.WriteSequenceNumber	=0x00000000	@0x00004012

=== Prefetcher Csr ===

MsgDmaRegs[Eng].MsgDmaPrefetcherCsr.Control	=0x00000000	@0x00004020
MsgDmaRegs[Eng].MsgDmaPrefetcherCsr.NextDescriptorPointerLow	=0x00000000	@0x00004024
MsgDmaRegs[Eng].MsgDmaPrefetcherCsr.NextDescriptorPointerHigh	=0x00000000	@0x00004028
MsgDmaRegs[Eng].MsgDmaPrefetcherCsr.DescriptorPollingFrequency	=0x00000000	@0x0000402c
MsgDmaRegs[Eng].MsgDmaPrefetcherCsr.Status	=0x00000000	@0x00004030

=== Descriptor at offset 0 ===

MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].ReadAddressLow	=0x00000000	@0x00005000
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].WriteAddressLow	=0x00000000	@0x00005004
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].Length	=0x00000000	@0x00005008
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].NextDescriptorPointerLow	=0x00000000	@0x0000500c
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].ActualBytesTransferred	=0x00000000	@0x00005010
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].Status	=0x00000000	@0x00005014
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].SequenceNumber	=0x00000000	@0x0000501c
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].ReadBurstCount	=0x00000000	@0x0000501e
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].WriteBurstCount	=0x00000000	@0x0000501f
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].ReadStride	=0x00000000	@0x00005020

```

MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].WriteStride           =0x00000000    @0x00005022
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].ReadAddressHigh      =0x00000000    @0x00005024
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].WriteAddressHigh    =0x00000000    @0x00005028
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh=0x00000000    @0x0000502c
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].Control              =0x00000000    @0x0000503c

```

```

.
.
.

```

=== Terminating Descriptor at offset 31 ===

```

MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.ReadAddressLow         =0x00000000    @0x000057c0
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.WriteAddressLow       =0x00000000    @0x000057c4
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.Length                 =0x00000000    @0x000057c8
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.NextDescriptorPointerLow =0x00000000    @0x000057cc
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.ActualBytesTransferred =0x00000000    @0x000057d0
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.Status                 =0x00000000    @0x000057d4
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.SequenceNumber         =0x00000000    @0x000057dc
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.ReadBurstCount        =0x00000000    @0x000057de
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.WriteBurstCount       =0x00000000    @0x000057df
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.ReadStride             =0x00000000    @0x000057e0
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.WriteStride           =0x00000000    @0x000057e2
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.ReadAddressHigh       =0x00000000    @0x000057e4
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.WriteAddressHigh      =0x00000000    @0x000057e8
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.NextDescriptorPointerHigh =0x00000000    @0x000057ec
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.Control                =0x00000000    @0x000057fc

```

```

.
.
.

```

===== MsgDma Engine 5 =====

=== Dispatcher Csr ===

```

MsgDmaRegs[Eng].MsgDmaDispatcherCsr.Status                                  =0x0000000a    @0x00004140
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.Control                               =0x00000000    @0x00004144
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.ReadFillLevel                         =0x00000000    @0x00004148
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.WriteFillLevel                       =0x00000000    @0x0000414a
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.ResponseFillLevel                    =0x00000000    @0x0000414c
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.ReadSequenceNumber                   =0x00000000    @0x00004150
MsgDmaRegs[Eng].MsgDmaDispatcherCsr.WriteSequenceNumber                  =0x00000000    @0x00004152

```

=== Prefetcher Csr ===

```

MsgDmaRegs[Eng].MsgDmaPrefetcherCsr.Control                               =0x00000000    @0x00004160
MsgDmaRegs[Eng].MsgDmaPrefetcherCsr.NextDescriptorPointerLow             =0x00000000    @0x00004164
MsgDmaRegs[Eng].MsgDmaPrefetcherCsr.NextDescriptorPointerHigh            =0x00000000    @0x00004168
MsgDmaRegs[Eng].MsgDmaPrefetcherCsr.DescriptorPollingFrequency           =0x00000000    @0x0000416c
MsgDmaRegs[Eng].MsgDmaPrefetcherCsr.Status                                =0x00000000    @0x00004170

```

=== Descriptor at offset 0 ===

```

MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].ReadAddressLow     =0x00000000    @0x00007800
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].WriteAddressLow    =0x00000000    @0x00007804
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].Length              =0x00000000    @0x00007808
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].NextDescriptorPointerLow =0x00000000    @0x0000780c
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].ActualBytesTransferred =0x00000000    @0x00007810
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].Status              =0x00000000    @0x00007814
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].SequenceNumber      =0x00000000    @0x0000781c
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].ReadBurstCount     =0x00000000    @0x0000781e
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].WriteBurstCount    =0x00000000    @0x0000781f
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].ReadStride          =0x00000000    @0x00007820
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].WriteStride         =0x00000000    @0x00007822
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].ReadAddressHigh    =0x00000000    @0x00007824
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].WriteAddressHigh   =0x00000000    @0x00007828
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh =0x00000000    @0x0000782c
MsgDmaDescriptorRegs[Eng].MsgDmaExtendedDescriptor[Id].Control              =0x00000000    @0x0000783c

```



```

.
.
.
=== Terminating Descriptor at offset 31 ===
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.ReadAddressLow      =0x00000000      @0x00007fc0
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.WriteAddressLow     =0x00000000      @0x00007fc4
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.Length              =0x00000000      @0x00007fc8
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.NextDescriptorPointerLow =0x00000000      @0x00007fcc
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.ActualBytesTransferred =0x00000000      @0x00007fd0
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.Status              =0x00000000      @0x00007fd4
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.SequenceNumber      =0x00000000      @0x00007fdc
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.ReadBurstCount      =0x00000000      @0x00007fde
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.WriteBurstCount     =0x00000000      @0x00007fdf
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.ReadStride          =0x00000000      @0x00007fe0
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.WriteStride         =0x00000000      @0x00007fe2
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.ReadAddressHigh     =0x00000000      @0x00007fe4
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.WriteAddressHigh    =0x00000000      @0x00007fe8
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.NextDescriptorPointerHigh =0x00000000      @0x00007fec
MsgDmaDescriptorRegs[Eng].MsgDmaTerminatingDescriptor.Control             =0x00000000      @0x00007ffc

```

3.1.5 ccrtnngfc_regedit

This is an interactive test to display and write to local, configuration and physical memory.

Usage: ./ccrtnngfc_regedit [-b board]
 -b board: Board number -- default board is 0

Example display:

```
./ccrtnngfc_regedit
```

```
Device Name: /dev/ccrtnngfc0
```

```
LOCAL REGION: Physical Addr=0xfb800000 Size=4194304 (0x00400000)
CONFIG REGION: Physical Addr=0xfbc00000 Size=32768 (0x0008000)
```

```
LOCAL: Register 0x7ffff7800000 Offset=0x0 Size=0x00400000
CONFIG: Register 0x7ffff7fb7000 Offset=0x0 Size=0x0008000
LIBPTR: Register 0x7ffff7ebf000 Offset=0x0 Size=0x00013e58
```

```
Initialize_Board: Firmware Rev. 0x10000 successful
```

```
Virtual Address: 0x7ffff7800000
```

```

1 = Create Physical Memory          2 = Destroy Physical memory
3 = Display Channel Data            4 = Display Driver Information
5 = Display Physical Memory Info    6 = Display Registers (CONFIG)
7 = Display Registers (LOCAL)      8 = Dump Physical Memory
9 = Reset Board                    10 = Write Register (LOCAL)
11 = Write Register (CONFIG)       12 = Write Physical Memory

```

```
Main Selection ('h'=display menu, 'q'=quit)->
```

3.1.6 ccrtnngfc_tst

This is an interactive test to exercise some of the driver features.

Usage: ./ccrtnngfc_tst [-b board]
 -b board: Board number -- default board is 0

Example display:

```
./ccrtnngfc_tst
```

```
Device Name: /dev/ccrtnngfc0
```

```
LOCAL REGION: Physical Addr=0xfb800000 Size=4194304 (0x00400000)  
CONFIG REGION: Physical Addr=0xfbc00000 Size=32768 (0x00008000)
```

```
LOCAL: Register 0x7ffff7800000 Offset=0x0 Size=0x00400000  
CONFIG: Register 0x7ffff7fb7000 Offset=0x0 Size=0x00008000  
LIBPTR: Register 0x7ffff7ebf000 Offset=0x0 Size=0x00013e58
```

```
Initialize_Board: Firmware Rev. 0x10000 successful
```

```
01 = add irq                02 = disable pci interrupts  
03 = enable pci interrupts  04 = get device error  
05 = get driver info        06 = get physical memory  
07 = init board             08 = mmap select  
09 = mmap(CONFIG registers) 10 = mmap(LOCAL registers)  
11 = mmap(physical memory)  12 = munmap(physical memory)  
13 = no command             14 = read operation  
15 = remove irq             16 = reset board  
17 = restore config registers 18 = write operation
```

```
Main Selection ('h'=display menu, 'q'=quit)->
```

3.1.7 ccrtnngfc_wreg

This is a simple test to write to the local registers at the user specified offset.

```
Usage: ./ccrtnngfc_wreg [-b Board] [-C] [-o Offset] [-s Size] [-v Value] [-x]  
-b Board   : Board selection -- default board is 0  
-C         : Select Config Registers instead of Local Registers  
-o Offset  : Hex offset to write to -- default offset is 0x0  
-s Size    : Number of bytes to write in decimal -- default size is 0x4  
-v Value   : Hex value to write at offset -- default value is 0x0  
-x         : Do not read back just written values -- default read back values
```

Example display:

```
./ccrtnngfc_wreg -v12345678 -o0x8000 -s400
```

```
Device Name: /dev/ccrtnngfc0
```

```
LOCAL REGION: Physical Addr=0xfb800000 Size=4194304 (0x00400000)  
CONFIG REGION: Physical Addr=0xfbc00000 Size=32768 (0x00008000)
```

```
LOCAL: Register 0x7ffff7800000 Offset=0x0 Size=0x00400000  
CONFIG: Register 0x7ffff7fb7000 Offset=0x0 Size=0x00008000  
LIBPTR: Register 0x7ffff7ebf000 Offset=0x0 Size=0x00013e58
```

```
Writing 0x12345678 to offset 0x8000 for 400 bytes
```

```
#### LOCAL REGS #### (length=400)  
+LCL+ 0x8000 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*  
+LCL+ 0x8010 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*  
+LCL+ 0x8020 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*  
+LCL+ 0x8030 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*  
+LCL+ 0x8040 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*  
+LCL+ 0x8050 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*  
+LCL+ 0x8060 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
```

```

+LCL+ 0x8070 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8080 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8090 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80a0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80b0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80c0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80d0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80e0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80f0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8100 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8110 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8120 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8130 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8140 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8150 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8160 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8170 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8180 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*

```

3.1.8 Flash/ccrtnghfc_flash

This program is used to burn new firmware or update the license of an already installed firmware. It can also be used to reload the firmware on the card. This must only be done at the direction of Concurrent Real-Time support team, otherwise, they could render the board useless.

```

./ccrtnghfc_flash -b[Board] -B -F[!] -i -L -q -Q -r[OutFile] -R -v -w[InFile] -X
-b [Board]          : Board number. Must be specified
-B                 : Reload Base Level Firmware if MultiFirmware support present
-F                 : Force Read Flash: Overwrite output file if exists
-F                 : Force Write Flash: Do not abort Flash burn for header label mismatch
-F!                : Force Write Flash: Serious override required to continue burning
-i                 : Query chip, on-board flash and InFile if specified
-L                 : Update License only. (default is to update entire firmware)
-q                 : Quite (non-interactive) mode
-Q                 : Quite (non-interactive) mode. Also dump FPGAWB message
-r                 : Read Flash and write to output file created by ./ccrtnghfc_flash
-r [OutFile]       : Read Flash and write to output file 'OutFile'
-R                 : Reload Firmware at sector address in Flash
-R [SectorNumber] : Reload Firmware at sector address 'SectorNumber'
-v                 : Enable verbose mode
-w [InFile]        : Read input FPGA file and Flash the board
-X                 : Use Full File. Do not truncate for firmware write

```

```

===== Notes =====
Board must be specified. Use '-b' option
Query option '-i' not allowed with '-B', '-R#', '-L', 'r' or '-X' options
Firmware reload '-B' or '-R' not allowed with '-i', '-L', '-r', '-w' or '-X' options
Firmware read flash '-r' not allowed with '-B', '-i', '-L', '-R', '-w' or '-X' options
Base Run Level '-B' or '-R#' option not allowed with '-i', '-L', 'r', '-w' or '-X' options
Must specify write flash option '-w' when License only option '-L' is specified
License only option '-L' not allowed with '-B', '-i', '-R', '-w' or '-X' options
Don't truncate file option '-X' cannot be selected with the license only update '-L' option
Don't truncate file option '-X' can only be used with the '-w' option
Inquiry '-i' can be used '-w' options
=====

```

```

e.g. ./ccrtnghfc_flash -b0          (Query chip and on-board Flash)
     ./ccrtnghfc_flash -b0 -i       (Query chip and on-board Flash)
     ./ccrtnghfc_flash -b0 -i -w InFile (Query chip, on-board Flash and InFile)
     ./ccrtnghfc_flash -b0 -r OutFile (On-board FPGA ==> OutFile)
     ./ccrtnghfc_flash -b0 -w InFile (InFile ==> On-board FPGA - use truncated file)

```

```

./ccrtnngfc_flash -b0 -w InFile -v      (InFile ==> On-board FPGA - use truncated file
- verbose)
./ccrtnngfc_flash -b0 -w InFile -X      (InFile ==> On-board FPGA - use entire file)
./ccrtnngfc_flash -b0 -w InFile -L      (InFile ==> On-board FPGA - only license
updated - interactive)
./ccrtnngfc_flash -b0 -w InFile -L -q   (InFile ==> On-board FPGA - only license
updated - non-interactive)
./ccrtnngfc_flash -b0 -R                (Reload Firmware - i.e. power-cycle the card)
- Run Level
./ccrtnngfc_flash -b0 -B                (Reload Firmware - i.e. power-cycle the card)
- Base Level
./ccrtnngfc_flash -b0 -R 0              (Reload Firmware - i.e. power-cycle the card)
- Base Level
./ccrtnngfc_flash -b0 -R 200            (Reload Firmware - i.e. power-cycle the card)
- at sector 200

```



*If the installed firmware is a Multi-Level firmware and you are running at Base Level, then the only utility that will be able to access the card will be this **ccrtnngfc_flash** utility. You will need to switch to Run Level before un-restricted access is allowed to the card.*

3.1.9 Flash/ccrtnngfc_label

This utility is only supplied for those customers that are creating their own firmware and need to install in a RedHawk system. In its simplest form, the customer will request a License file from Concurrent Real-Time for the option to burn their custom firmware. The license file (*.lic) supplied by Concurrent Real-Time, along with the customer firmware (*.rpd) file will be supplied to this utility to create a burnable FPGA file (*.cust), that will be supplied to the *ccrtnngfc_flash* utility to burn the firmware on the card.

The user can also supply the '-x' option to additionally create a license only file (*.cust.liconly) file that is associated with the firmware (.rpd). This is useful if you only wish to update the license information of a card that already has the same firmware installed. This is similar to having a (*.cust) file and using the '-L' option when running the *ccrtnngfc_flash* utility.

```

./ccrtnngfc_label -d[OutputDirectory] -c[ChipName] -F -i[InputFile] -K[FpgawbKey]
                  -L[LicenseFile] -m[MemberCode] -o[OutputFile] -S[RunLevelSectorAddress]
                  -t[Tag] -x
-d [OutputDirectory] : Directory to use for Output File
-c [ChipName]        : Chip Name. One of:
                    EPCQ16 EPCQ32 EPCQ64 EPCQ128 EPCQ256 MTQU512
                    (This option is mandatory if not specified in license file)
-F                  : Force overwriting of output file if it exists
-i [InputFile]      : Raw input file. (.rpd extension)
-K [FpgawbKey]     : Fpgawb Key is required if license contains FPGA workbench restriction
-L [LicenseFile]   : License file (.lic extension) to restrict firmware access
                    If '-i' option is not specified, the license file is dumped to stderr
-m [MemberCode]    : Specify Member Code (048 115)
                    (This option is mandatory if not specified in license file)
-o [OutputFile]    : Use output file instead of the default file created by the program
-S [RunLevelSectorAddress] : Run Level Sector Address. (This option is mandatory if not
                    specified in license file)
                    : S0=Base Level, S#=Run Level Number
-t [Tag]           : Insert this tag name in the default file created by the program
-x                : Create an additional License only file (*.liconly)

```

==== Notes ====

- At least one of option '-L' or '-i' must be specified
- If option '-L' is specified and option '-i' is not specified, license file is dumped
- If option '-i' is specified and option '-L' is not specified, input RPD file information

is displayed

- Options 'c', '-m' and '-S' are required if they have not already been defined in LicenseFile
- You cannot specify a Run Level Sector '-S' with Single Level Firmware '-1' option
- Run Level Sector address of zero '-S0' represents the Base Level Firmware in Multi-Firmware support
- If option '-o' is not specified, the created customer FPGA file name will be as follows:
 <OutputDirectory>/<InputFile>_<Tag>_<Function>_<ChipName><MemberCode><RunLevel>.cust
- If the license file contains an FPGAWB restrict key, then the '-K' FpgawbKey is required

e.g. `./ccrtngfc_label -iraw_file.rpd -L LicenseFile.lic` (in its simplest form)
 (output file created is: 'raw_file_<Function>_<ChipName><MemberCode><RunLevel>.cust')

- `./ccrtngfc_label -L LicenseFile.lic` (this will display licensing information)
- `./ccrtngfc_label -iraw_RUN_file.rpd -ooutput_file.cust -S100 -L LicenseFile.lic`
- `./ccrtngfc_label -iraw_SINGLE_file.rpd -L LicenseFile.lic`
- `./ccrtngfc_label -iraw_RUN_file.rpd -ooutput_file.cust -S200 -L LicenseFile.lic`
- `./ccrtngfc_label -iraw_BASE_file.rpd -S0 -L LicenseFile.lic`
 (Will cause firmware to be loaded at start offset Base Run Level)

3.1.10 Flash/ccrtngfc_dump_license

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

Format: `./ccrtngfc_dump_license <Firmware file>`

This utility only dumps the license information from the *.cust or *.liconly files and not the *.lic license file

e.g. `./ccrtngfc_dump_license output_file_048.NGFC_NOLIMIT_MultiFunc_MTQU512048S350.cust`
 `./ccrtngfc_dump_license output_file_048.NGFC_NOLIMIT_MultiFunc_MTQU512048S350.cust.liconly`

3.2 Application Program Interface (API) Access Example Tests

These set of tests are in the `.../test/lib` directory and use the API.

3.2.1 lib/ccrtngfc_acquire_physmem

This is a simple test to acquire physical memory and pause. This is used to test the Cloning region addressing functionality.

```
Usage: ./ccrtngfc_acquire_physmem [-b Board] [-s PhysMemSize]
      -b <board>                (board #, default = 0)
      -s <PhysMemSize>         (Physical Memory Size, default = NONE)
```

Example display:

```
./ccrtngfc_acquire_physmem -s 100
```

```
Physical Memory Information:
  UserPID           =184103
  PhysMemPtr        =0x65d27000
  DriverVirtMemPtr  =0xffff993ee5d27000
  MmappedUserMemPtr =0x7ffff7fdd000
  PhysMemSize       =0x00001000
  PhysMemSizeFreed  =0x00000000
  EntryInTxTbl      =0
  NumOfEntriesUsed  =1
  Flags             =0x0000
....pausing... use <CTRL-C> to terminate
```

3.2.2 lib/ccrtngfc_adc

This test performs validation of the Multi-Function ADC card.

```
Usage: ./ccrtngfc_adc [-A] [-a RollingAve] [-b BoardNo] [-C AdcUpdateClock] [-d Delay]
                  [-e MsgDmaEngine] [-E ExpInpVolt] [-f DataFormat] [-F DebugFile] [-i]
                  [-l LoopCnt] [-m XferMode] [-M SelectAdcModule] [-n NumChans] [-N]
                  [-s InputSignal] [-t Compare] [-T TestBus]
-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..4 or 'n|N')
  -C 0,5,4        (Ch0..3=Clock0, Ch4..7=Clock5, Ch8..11=Clock4 at MAX SPS)
  -C ^2,3         (Ch0..3=Clock2, Ch4..11=Clock3 assignment only) [** Skip Programming
                  Clock **]
  -C 3@20000.0/n  (Ch0..3=Clock3 at 20000 SPS, Ch4..11=No Clock)
  -C 4            (Ch0..11=Clock4 at MAX SPS)
  -C 4@150000.0   (Ch0..11=Clock4) at 150000 SPS)
-d Delay          (Delay between screen refresh -- default is 0 milli-seconds)
-e MsgDmaEngine   (Select MsgDma Engine -- default "=== get free engine ===")
-E <ExpInpVolts>@<Tol> (Expected Input Volts@Tolerance -- default Tol=0.003000)
  +@<Tol>         (Positive Calibration Ref Volt@Tolerance)
  -@<Tol>         (Negative Calibration Ref Volt@Tolerance)
  s@<Tol>         (Requires '-s' input signal option to specify voltage Volt@Tolerance)
                  (valid '-s' arguments are 'g','+', '-', 'f')
-f DataFormat     (select data format, '2' or 'b')
  -f b,2,b        (Ch0..3 & Ch8..11=Offset binary, Ch4..7=Two's complement)
```

```

-f 2/b          (Ch0..3=Two's complement, Ch4..11=Offset binary)
-f b           (Ch0..11=Offset binary)
-F DebugFile   (Debug file with menu display -- default "=== None ===")
#DebugFile     (Debug file without display (only summary) -- default "=== None ===")
@DebugFile     (Debug file without display -- default "=== None ===")
~DebugFile     (For gnuplot, no header or summary -- default "=== None ===")
@, # or ~     (No debug file and no display -- default "=== None ===")
-i            (Enable Interrupts -- default = Disable)
-l LoopCnt     (Loop count -- default is 0)
-m XferMode    (Transfer Mode -- default = 'MSGDMA Channel')
  -mlm        (Library: (Channel Registers) Modular scatter-gather DMA mode)
  -mlx        (Library: (Channel Registers) Clone (ADC->MEM) Modular scatter-gather
              DMA mode)
  -mlz        (Library: (Channel Registers) Clone (ADC->DIAG->MEM) Modular
              scatter-gather DMA mode)
  -mdp        (Driver: (Channel Registers) PIO mode)
  -mlc        (Library: (Channel Registers) program I/O Fast Memory Copy)
  -mlp        (Library: (Channel Registers) PIO)
  -mdP        (Driver: (FIFO) PIO mode)
  -mlP        (Library: (FIFO) PIO mode)
-M SelectAdcModule (Select ADC Module -- default is module 0)
-n NumChans    (Number of channels -- default is 12)
-N            (Open device with O_NONBLOCK flag)
-s InputSignal (select input signal, 'e', 'g', '+', '-', 'f', '0..11')
  -s e,g,e     (Ch0..3 & 8..11=External input, Ch4..7=ground calibration)
  -s +/e       (Ch0..3=Postive calibration, Ch8..11=external reference)
  -s -         (Ch0..11=Negative calibration)
  -s e/e,f     (Ch0..7=external reference, Ch8..11=4 volt calibration)
  -s e/10      (Ch0..3=external reference, Ch4..11=DAC Channel 10)
-t Compare     (Compare two channels for +/- -- default is "=== None ===")
  -t0,10       (Compare channel 0 and 10 for being in sync)
  -t5/7        (Compare channel 5 and 7 for being in sync)
  -t2,11@0.500 (Compare channel 2 and 11 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)

```

```

e.g. ./ccrtnngfc_adc -A -C0@150000.0/1@1234.0 -se/+ (Autocal, ADC0=150000Hz external input,
                                                    ADC1=1234Hz Positive Cal.)
      ./ccrtnngfc_adc -A -C0 -s+ -E+             (Autocal, Max Clock, Positive cal.
                                                    input, validate result)
      ./ccrtnngfc_adc -A -C0 -s- -t0,11 -a100    (Autocal, Max Clock, Negative cal.
                                                    input, compare ch0 and ch11, rolling ave=100)
      ./ccrtnngfc_adc -C0 -s- -Es               (Max Clock, -9.91V input, validate against -9.91V)
      ./ccrtnngfc_adc -C0 -sf -Es               (Max Clock, +4V input, validate against +4V)

```

If you wish to run both ADC and DAC concurrently, you can try the following:

- ```

1) ./ccrtnngfc_clock -C0@4500000 -C1@1000000 -l1 (Program ADC Clock 0 at 4.5 MHz and DAC
 Clock 1 at 1.0 MHz)
2) ./ccrtnngfc_dac -C^1 (Start DAC using 1 MHz Clock 1 in first
 window)
3) ./ccrtnngfc_adc -C^0 (Start ADC using 4.5 MHz Clock 0 in second
 window)

```

### Example display:

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

ADC Information:

```

Flags = 0x00000001
ModuleNumber = 0
Handle = 0x76da40
local_ptr = 0x7ffff7e67000
local_adc_ptr = 0x7ffff7e6b000
local_adc_fifo_ptr = 0x7ffff7e77000
AdcFp = 4
AdcDeviceName = /dev/ccrtnngfc0_adc0

```

local\_ptr=0x7ffff7e67000

Physical Memory Information:

```

UserPID =184544
PhysMemPtr =0x65d27000
DriverVirtMemPtr =0xffff93ee5d27000
MmappedUserMemPtr =0x7ffff7fdd000
PhysMemSize =0x00001000
PhysMemSizeFreed =0x00000000
EntryInTxTbl =0
NumOfEntriesUsed =1
Flags =0x0000

```

Auto Calibration started...done. (0.602 seconds)

```

Board Number [-b]: 0
Update Clock Selected [-C]: Ch00..03 OutputClock=0 (0x7) (150000.000 SPS)
: Ch04..07 OutputClock=1 (0x1) (1234.000 SPS)
: Ch08..11 OutputClock=1 (0x1) (1234.000 SPS)
Delay [-d]: 0 milli-seconds
MSGDMA Engine [-e]: 0
Expected Input Volts [-E]: === Not Specified ===
Data Format [-f]: Ch00..03 Offset binary (0x0)
: Ch04..07 Offset binary (0x0)
: Ch08..11 Offset binary (0x0)
Interrupts [-i]: Disabled
Loop Count [-l]: ***Forever***
Transfer Mode [-m]: Library: (Channels Registers) MODULAR SCATTER-GATHER DMA I/O
Selected ADC Module [-M]: 0
Number of Channels [-n]: 12
Input Signal [-s]: Ch00..03 [0]External Input
: Ch04..07 [1]Calibration Input (0x01: Positive 9.91)
: Ch08..11 [1]Calibration Input (0x01: Positive 9.91)
Scan Count : 33495 (0:00:00:04)
Read Duration (microsecs) : TotalDelta: 3.651 (min= 3.576/max= 9.599/ave= 3.679)

```

##### Raw Data #####

```

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
====
[0] 7fef 8023 803f 801f fedf feda fedc feda fed9 fedb
[1] fed4 fedb

```

##### Volts #####

```

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] -0.0052 +0.0107 +0.0192 +0.0095 +9.9118 +9.9103 +9.9109 +9.9103 +9.9100 +9.9106
[1] +9.9084 +9.9106

```

```

=====
Date: Tue May 16 15:07:20 2023
Expected Input Volts: === Not Specified ===

```



```

Scan Counter: 263107
WorstMinChanVoltsHWM: -0.007324 (Ch00)
WorstMaxChanVoltsHWM: 9.913635 (Ch10)

```

```

=====
<----- (volts) ----->
Chan Min Max Ave TolerExceededCnt
=====
00 -0.0073 -0.0027 -0.0052 -
01 0.0089 0.0131 0.0110 -
02 0.0171 0.0217 0.0193 -
03 0.0070 0.0110 0.0090 -
04 9.9078 9.9133 9.9105 -
05 9.9078 9.9133 9.9106 -
06 9.9078 9.9133 9.9107 -
07 9.9078 9.9133 9.9108 -
08 9.9078 9.9130 9.9105 -
09 9.9078 9.9133 9.9106 -
10 9.9072 9.9136 9.9102 -
11 9.9078 9.9133 9.9104 -
=====

```

### 3.2.3 lib/ccrtngfc\_adc\_calibrate

This test is useful for performing, saving and restoring ADC calibration.

```
Usage: ./ccrtngfc_adc_calibrate [-A] [-b board] [-i inCalFile] [-o outCalFile]
 [-R]
```

```

-A (perform Auto Calibration)
-b <board> (board #, default = 0)
-i <In Cal File> (input calibration file [input->board_reg])
-o <Out Cal File> (output calibration file [board_reg->output])
-R (reset ADC calibration)

```

```

e.g. ./ccrtngfc_adc_calibrate (Dump calibration information to
 stdout)
 ./ccrtngfc_adc_calibrate -A -o Calfile (Perform Auto calibration and dump
 information to 'Calfile')
 ./ccrtngfc_adc_calibrate -i Calfile (Update board calibration with
 supplied 'Calfile')
 ./ccrtngfc_adc_calibrate -R (Reset ADC calibration)

```

#### Example display:

```
./ccrtngfc_adc_calibrate -A
```

```

ADC Information:
 Flags = 0x00000001
 ModuleNumber = 0
 Handle = 0x75a4c0
 local_ptr = 0x7ffff7f55000
 local_adc_ptr = 0x7ffff7f59000
 local_adc_fifo_ptr = 0x7ffff7f65000
 AdcFp = 4
 AdcDeviceName = /dev/ccrtngfc0_adc0

```

```

Device Name : /dev/ccrtngfc0
Board Serial No: 0 (0x00000000)
Auto Calibration started...done. (0.607 seconds)

```

```

==> Dump to 'stdout'
#Date : Tue May 16 15:09:50 2023

```

| #Chan | Negative               | Offset                  | Positive               |
|-------|------------------------|-------------------------|------------------------|
| #==== | =====                  | =====                   | =====                  |
| ch00: | 0.99966217717155814171 | -0.00152587890625000000 | 1.00019804341718554497 |
| ch01: | 0.99926335178315639496 | -0.00610351562500000000 | 1.00064194900915026665 |
| ch02: | 0.99926836881786584854 | -0.00854492187500000000 | 1.00128439348191022873 |
| ch03: | 1.00097584817558526993 | 0.00030517578125000000  | 1.00129837542772293091 |
| ch04: | 1.00159255694597959518 | 0.01129150390625000000  | 1.00023180851712822914 |
| ch05: | 1.00036927126348018646 | -0.00396728515625000000 | 1.00088152382522821426 |
| ch06: | 0.99907627562060952187 | -0.00518798828125000000 | 1.00044342176988720894 |
| ch07: | 0.99927996192127466202 | -0.00366210937500000000 | 1.00012438092380762100 |
| ch08: | 0.99944108212366700172 | -0.01129150390625000000 | 1.00134854065254330635 |
| ch09: | 0.99980337964370846748 | -0.00488281250000000000 | 1.00099764997139573097 |
| ch10: | 0.99926861049607396126 | -0.01037597656250000000 | 1.00179374404251575470 |
| ch11: | 1.00017620576545596123 | 0.00213623046875000000  | 1.00001918524503707886 |

### 3.2.4 lib/ccrtngfc\_adc\_fifo

This test performs validation of the Multi-Function ADC FIFO operation of the card. If two daughter cards are present, both are run concurrently.

```
Usage: ./ccrtngfc_adc_fifo [-A] [-b BoardNo] [-c ChannelSelectMask] [-C AdcUpdateClock]
 [-d Delay] [-e MsgDmaEngine] [-E ExpInpVolt] [-f DataFormat]
 [-F DebugFile] [-i] [-l LoopCnt] [-m XferMode]
 [-M SelectAdcModule] [-N] [-s InputSignal] [-S NumberOfSamples]
 [-T TestBus] [-V] [-W JitterAndOrDebug] [-X]
-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..4 or 'n|N'. If '^' first arg, Skip
 Programming Clock)
 -C 0,3,4 (Ch0..3=Clock0, Ch4..7=Clock3, Ch8..11=Clock4 at MAX SPS)
 -C ^2,3 (Ch0..3=Clock2, Ch4..11=Clock3 assignment only) [** Skip Programming
 Clock **]
 -C 3@20000.0/n (Ch0..3=Clock3 at 20000 SPS, Ch4..11=No Clock)
 -C 4 (Ch0..11=Clock4 at MAX SPS)
 -C 4@150000.0 (Ch0..11=Clock4) at 150000 SPS)
-d Delay (Delay between screen refresh -- default is 1 milli-seconds)
-e MsgDmaEngine (Select MsgDma Engine -- default "=== get free engine ===")
 -e 0,2 (MsgDma Engine 0 for ADC Module 0 and MsgDma Engine 2 for ADC Module 1)
 -e 3,1 (MsgDma Engine 3 for ADC Module 0 and MsgDma Engine 1 for ADC Module 1)
 -e ,3 (MsgDma Engine Default for ADC Module 0 and MsgDma Engine 3 for ADC
 Module 1)
-E <ExpInpVolts>@<Tol> (Expected Input Volts@Tolerance -- default Tol=0.003000)
 +@<Tol> (Positive Calibration Ref Volt@Tolerance)
 -@<Tol> (Negative Calibration Ref Volt@Tolerance)
 s@<Tol> (Requires '-s' input signal option to specify voltage Volt@Tolerance)
 (valid '-s' arguments are 'g','+', '-', 'f', 't')
-f DataFormat (select data format, '2' or 'b')
 -f b,2,b (Ch0..3 & Ch8..11=Offset binary, Ch4..7=Two's complement)
 -f 2/b (Ch0..3=Two's complement, Ch4..11=Offset binary)
 -f b (Ch0..11=Offset binary)
-F DebugFile (Debug file [formatted] -- default "=== None ===")
 /dev/null (No debug output)
 @DebugFile (Debug file [raw])
 #DebugFile (Debug file [raw] followed by creation of 'FORMATTED' file)
-i (Enable Interrupts -- default = Disable)
-l LoopCnt (Loop count -- default is 0)
-m XferMode (Transfer Mode -- default = Library MSGDMA)
 -mdP (Driver: (FIFO) PIO mode)
 -mlM (Library: (FIFO) Modular Scatter-Gather DMA mode)
```

```

 -mlP (Library: (FIFO) PIO mode)
 -M SelectAdcModule (Select ADC Module -- default is module 0)
 -N (Open device with O_NONBLOCK flag for driver operations)
-s InputSignal (select input signal, 'e', 'f', 'g', 'n', 'p', '+', '-', '0..11')
 -s e,g,e (Ch0..3 & 8..11=External input, Ch4..7=ground calibration)
 -s +/e (Ch0..3=Postive calibration, Ch8..11=external reference)
 -s - (Ch0..11=Negative calibration)
 -s e/e,f (Ch0..7=external reference, Ch8..11=4 volt calibration)
 -s e/10 (Ch0..3=external reference, Ch4..11=DAC Channel 10)
-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 (Perform Data Validation, default is not to validate data)
-W JitterAndOrDebug (External A/C input signal - Sine Wave - Valid args are 'j' or 'd')
-W (External A/C input signal - Sine Wave)
-W j (External A/C input signal - Sine Wave, add jitter for low input
 frequencies)
-W d (External A/C input signal - Sine Wave, enable debug)
-W jd (External A/C input signal - Sine Wave, add jitter and enable debug)
-X (Disable Curses Statistics Display)

```

Note: If '^' is specified as first argument to '-C' option, then skip programming clocks  
When specifying the '-W' A/C option, you must specify the -E, -l and -F# options  
When specifying the '-W' A/C option, the expected voltage '-E' supplied is half of P-P  
Low Input A/C frequency correction option '-Wj' must only be used for frequencies <= 10KHz

```

e.g. ./ccrtnghfc_adc_fifo -C0,1@100000 -se/+ (ADC0=4.5MHz external input, ADC1=100000Hz
 Positive Cal.)
 ./ccrtnghfc_adc_fifo -C^0,1 -s- -Es (Max Clock, -9.91V input, validate against
 -9.91V) - [** Skip Programming Clock **]
 ./ccrtnghfc_adc_fifo -C0,1@100000 -sf -Es (Max Clock, +4.030 V input, validate against
 +4.030 V)
 ./ccrtnghfc_adc_fifo -C0 -M0 -V (Max Clock, module 0, validation data)
 ./ccrtnghfc_adc_fifo -C0 -l1000 -F#Debug (Max Clock, create 'Debug.n' raw and
 'Debug.n.FORMATTED' files for both modules)
 ./ccrtnghfc_adc_fifo -l50 -E9 -F#Out -W -c1 (external 9V amplitude A/C input into channel
 0 for 50 loop count)

```

If you wish to test A/C input try the following:

- 1) Connect a 562500 Hz, +/-2 Volts (4.0 Volts P-P) Sine wave to Channel 0 of the ADC
- 2) ./ccrtnghfc\_adc\_fifo -A -l100 -E2@0.35 -F#DEBUG -W -c1 -M0 -se -C0 (At 4.5MSPS, you should get 8 sample points per Sine Wave)

You should see a display similar to below:

```

 Expected Input Sine Wave: 4.00 volts P-P (Tolerance 0.350000 volts)
 Total Tolerance Exceed Count: 0

```

| Chan | MinVolts  | MaxVolts  | MinPpAmp  | MaxPpAmp  | MinError  | MaxError  | dB       | Freq(Hz) | TolerExceedCnt |
|------|-----------|-----------|-----------|-----------|-----------|-----------|----------|----------|----------------|
| 0    | -2.017822 | +2.017212 | +3.704834 | +4.032898 | -0.000061 | -0.295166 | -0.66582 | 562500   | -              |

If you wish to run both ADC and DAC concurrently, you can try the following:

- 1) ./ccrtnghfc\_clock -C0@4500000 -C1@1000000 -l1 (Program ADC Clock 0 at 4.5 MHz and DAC Clock 1 at 1.0 MHz)
- 2) ./ccrtnghfc\_dac\_fifo -C^1 (Start DAC Fifo using 1 MHz Clock 1 in first window)
- 3) ./ccrtnghfc\_adc\_fifo -C^0 (Start ADC Fifo using 4.5 MHz Clock 0 in second window)

Example display:

```

./ccrtnghfc_adc_fifo -C0@300000,1@100000 -se/+ -l1000 -V

```

```

ADC Module 0
ADC Information:
 Flags = 0x00000001
 ModuleNumber = 0
 Handle = 0x76c3a0
 local_ptr = 0x7ffff7f52000
 local_adc_ptr = 0x7ffff7f56000
 local_adc_fifo_ptr = 0x7ffff7f62000
 AdcFp = 4
 AdcDeviceName = /dev/ccrtngfc0_adc0
Physical Memory Information:
 UserPID =184570
 PhysMemPtr =0x65e80000
 DriverVirtMemPtr =0xffff993ee5e80000
 MmappedUserMemPtr =0x7ffff7ed2000
 PhysMemSize =0x00080000
 PhysMemSizeFreed =0x00000000
 EntryInTxTbl =0
 NumOfEntriesUsed =1
 Flags =0x0000
 NumOfChannels=12, FirstChannel=0, LastChannel=11, NumAdc0Chans=4 NumAdc1Chans=4 NumAdc2Chans=4
ADC Module 1
ADC Information:
 Flags = 0x00000001
 ModuleNumber = 1
 Handle = 0x76c3a0
 local_ptr = 0x7ffff7f52000
 local_adc_ptr = 0x7ffff7f58000
 local_adc_fifo_ptr = 0x7ffff7f6a000
 AdcFp = 5
 AdcDeviceName = /dev/ccrtngfc0_adc1
Physical Memory Information:
 UserPID =184570
 PhysMemPtr =0x65f00000
 DriverVirtMemPtr =0xffff993ee5f00000
 MmappedUserMemPtr =0x7ffff7e52000
 PhysMemSize =0x00080000
 PhysMemSizeFreed =0x00000000
 EntryInTxTbl =1
 NumOfEntriesUsed =1
 Flags =0x0000
 NumOfChannels=12, FirstChannel=0, LastChannel=11, NumAdc0Chans=4 NumAdc1Chans=4 NumAdc2Chans=4
===== CpuCountAssignedToThisTask=12
Measuring how long it takes to collect 49152 samples... Measuring how long it takes
to collect 49152 samples...

local_ptr=0x7ffff7f52000

 Number of Samples =49152
 Transfer Mode =Library Modular Scatter-Gather DMA Mode

Board 0: Time in microseconds (TT=Total, FT=Free, FF=FIFO fill, RT=Read, mi=min,
ma=max, av=ave)

ADC0: Eng0: 269: TT=24471.08 FT= 7.81 FF=23013.55 RT= 502.43
(min=501.70/max=566.51/ave=513.58) 391.31 MBytes/Sec (fifo=128156 - 97.78%)
ADC1: Eng1: 269: TT=24558.13 FT= 5.96 FF=23072.54 RT= 503.51
(min=501.59/max=565.34/ave=513.49) 390.47 MBytes/Sec (fifo=128100 - 97.73%)

#####

```

```

ADC Module '0'
#####

=====
 Date: Tue May 16 15:21:03 2023
Expected Input Volts: === Not Specified ===
Scan Counter: ADC0=7372800 ADC1=2457600 ADC2=2457600
Approx. Sample/Second: ADC0=300000 ADC1=100000 ADC2=100000
NumberOfChans: ADC0=4 ADC1=4 ADC2=4
Overflow Count ###: 0 (First Overflow Scan Count: 0)
WorstMinChanVoltsHWM: -0.008240 (Ch00)
WorstMaxChanVoltsHWM: 9.913635 (Ch07)
=====

<----- (volts) ----->
Chan Min Max Ave DetectedCnt TolerExeededCnt
=====
00 -0.0082 -0.0024 -0.0052 7372800 -
01 0.0085 0.0134 0.0112 7372800 -
02 0.0171 0.0220 0.0194 7372800 -
03 0.0070 0.0113 0.0092 7372800 -
04 9.9063 9.9130 9.9098 2457600 -
05 9.9066 9.9130 9.9100 2457600 -
06 9.9063 9.9133 9.9101 2457600 -
07 9.9066 9.9136 9.9101 2457600 -
08 9.9060 9.9127 9.9096 2457600 -
09 9.9069 9.9133 9.9100 2457600 -
10 9.9063 9.9130 9.9097 2457600 -
11 9.9066 9.9130 9.9098 2457600 -
=====

```

```

Below are the statistics for 49152 samples:
Estimated time to collect samples: 24579.621 usecs
Total work time breakdown 24577.287 usecs
Average time to fill FIFO: 23077.097 usecs
Average time to read samples: 502.957 usecs
Average time to process samples: 991.382 usecs
Average time other: 5.851 usecs
Approximate free time available: 23085.282 usecs

```

```

#####
ADC Module '1'
#####

=====
 Date: Tue May 16 15:21:03 2023
Expected Input Volts: === Not Specified ===
Scan Counter: ADC0=7372800 ADC1=2457600 ADC2=2457600
Approx. Sample/Second: ADC0=300000 ADC1=100000 ADC2=100000
NumberOfChans: ADC0=4 ADC1=4 ADC2=4
Overflow Count ###: 0 (First Overflow Scan Count: 0)
WorstMinChanVoltsHWM: -0.001221 (Ch03)
WorstMaxChanVoltsHWM: 9.911804 (Ch07)
=====

<----- (volts) ----->
Chan Min Max Ave DetectedCnt TolerExeededCnt
=====
00 0.0134 0.0253 0.0166 7372800 -
01 0.0064 0.0116 0.0090 7372800 -
02 0.0073 0.0128 0.0101 7372800 -
03 -0.0012 0.0040 0.0012 7372800 -
04 9.8996 9.9069 9.9033 2457600 -

```

|    |        |        |        |         |   |
|----|--------|--------|--------|---------|---|
| 05 | 9.8972 | 9.9036 | 9.9005 | 2457600 | - |
| 06 | 9.8944 | 9.9014 | 9.8982 | 2457600 | - |
| 07 | 9.9051 | 9.9118 | 9.9086 | 2457600 | - |
| 08 | 9.8904 | 9.8978 | 9.8944 | 2457600 | - |
| 09 | 9.8987 | 9.9054 | 9.9021 | 2457600 | - |
| 10 | 9.8941 | 9.9011 | 9.8976 | 2457600 | - |
| 11 | 9.9042 | 9.9109 | 9.9076 | 2457600 | - |

=====

Below are the statistics for 49152 samples:

```

Estimated time to collect samples: 24582.412 usecs
Total work time breakdown 24577.313 usecs
 Average time to fill FIFO: 23076.040 usecs
 Average time to read samples: 503.000 usecs
 Average time to process samples: 992.396 usecs
 Average time other: 5.876 usecs
Approximate free time available: 23087.015 usecs

```

### 3.2.5 lib/ccrtnngfc\_adc\_sps

This is a useful tool to display the sample rate of various channels.

```

Usage: ./ccrtnngfc_adc_sps [-b Board] [-c StartChan,StopChan] [-C AdcUpdateClock]
 [-e MsgDmaEngine] [-E ExpSPS@Tol] [-l LoopCnt]
 [-M SelectAdcModule] [-t TolerancePPT]

-b Board (Board number -- default is 0)
-c StartChan,EndChan (Select start and end channel numners -- default 0,11)
 -c 4,11 (select channels 4 through 11 for processing)
 -c 7 (select channels 7 through 11 for processing)
-C AdcUpdateClock (select ADC update clock, 0..4 or 'n|N')
 -C 0,5,4 (Ch0..3=Clock0, Ch4..7=Clock5, Ch8..11=Clock4 at MAX SPS)
 -C 3@20000.0/n (Ch0..3=Clock3 at 20000 SPS, Ch4..11=No Clock)
 -C 4 (Ch0..11=Clock4 at MAX SPS)
 -C 4@150000.0 (Ch0..11=Clock4) at 150000 SPS)
-e MsgDmaEngine (Select MsgDma Engine -- default "=== get free engine ===")
-E ExpSPS@Tol (specify expected samples/second and tolerance for each ADC)
 -E C (All ADC's to use clock samples/second and default tolerance 0.010%)
 -E c@0.02,30000 (ADC 0 uses clock samples/second and tolerance 0.02%, remaining
 use 30,000 SPS and default tolerance 0.010%)
 -E C@0.02,C (All ADC's to use clock samples/second and default tolerance
 except for ADC 0 tolerance of 0.02%)
 -E 10000,c (ADC 0 to use 10000 SPS, rest of ADCs to use clock samples/second.
 Default tolerance for all ADCs)
 -E 10000,20000,30000 (ADC 0, 1 and 2 to use 10000 SPS, 2000 SPS and 3000 SPS
 respectively. Default tolerance for all ADCs)
-F DebugFile (Debug file with menu display -- default "=== None ===")
 @DebugFile (Debug file without menu display (only summary and rate display)
 -- default "=== None ===")
 @ (No debug file and no menu display (only summary and rate display)
 -- default "=== None ===")
-l LoopCnt (Loop Count -- default is 10000000)
-l 0 (Loop forever)
-M SelectAdcModule (Select ADC Module -- default is module 0)
-t TolerancePPT (Tolerance in Parts/Trillion -- default is 0.020000 PPT)

e.g. ./ccrtnngfc_adc_sps -C0@123456,1@78912 (ADC0 is 123456Hz, ADC1 is 78912Hz)

```

Example display:

```
./ccrtnngfc_adc_sps -C0@123456,1@78912
```

```

ADC Information:
 Flags = 0x00000001
 ModuleNumber = 0
 Handle = 0x76aec0
 local_ptr = 0x7ffff7f52000
 local_adc_ptr = 0x7ffff7f56000
 local_adc_fifo_ptr = 0x7ffff7f62000
 AdcFp = 4
 AdcDeviceName = /dev/ccrtnngfc0_adc0

```

local\_ptr=0x7ffff7f52000

```

Physical Memory Information:
 UserPID =184607
 PhysMemPtr =0x65e58000
 DriverVirtMemPtr =0xffff993ee5e58000
 MmappedUserMemPtr =0x7ffff7f4a000
 PhysMemSize =0x00008000
 PhysMemSizeFreed =0x00000000
 EntryInTxTbl =0
 NumOfEntriesUsed =1
 Flags =0x0000

```

Read: Size 32736, Count 19 (FIFO wait: 7155.4us, Read time/rate: 87.7us/373.2MBPS)

```

===== Samples/Second =====
 [0] [1] [2] [3] [4] [5] [6] [7]
=====
[00 07] 123460 123460 123460 123460 78914 78914 78914 78914
[08 15] 78915 78915 78915 78915

```

==== No overflow occurred (HWM Samples In fifo 8200) ====

```

=====
 <--- (Samples/Second) --->
Chan Min Max Ave
=====
 0 123456 123462 123460
 1 123456 123462 123460
 2 123456 123462 123460
 3 123456 123462 123460
 4 78913 78916 78914
 5 78913 78916 78914
 6 78913 78916 78914
 7 78913 78916 78914
 8 78913 78916 78914
 9 78913 78916 78914
 10 78913 78916 78914
 11 78913 78916 78914

```

### 3.2.6 lib/ccrtnngfc\_check\_bus

This is a simple test to check whether there is interference from other cards that may be sharing the same bus. It simply computes the time it takes to perform hardware reads and computes the jitter. It must be run as *root*.

```

Usage: ./ccrtnngfc_check_bus [-b Board] [-c CPU] [-l LoopCnt] [-o Offset] [-t Tolerance] [-w]
 -b Board (Board number -- default is 0)
 -c CPU (CPU number -- default is 1)
 -l LoopCnt (Loop Count -- default is 1000000)
 -l 0 (Loop forever)

```

```
-o Offset (Hex offset to read/write from -- default offset is 0x00000000)
-t Tolernace (Tolerance -- default is 1.00 micro-seconds)
-w (Use writes instead of reads)
```

Example display:

```
sudo ./ccrtnafc_check_bus
```

```
local_ptr=0x7ffff7fd7000
10000000: usec/read: Cur=1.181 (Min=1.159 Max=1.794 Ave= 1.181335)
 [Bus Jitter (usec): 0.635 ==> LOW]
```

### 3.2.7 lib/ccrtnafc\_clock

This is a useful tool to display information of the various clocks and also program them.

```
Usage: ./ccrtnafc_clock [-b BoardNo] [-C UpdateClock] [-d Delay] [-l LoopCnt] [-R]
 [-S ClockSource] [-t TolerancePPT] [-z]
-b BoardNo (Board number -- default is 0)
-C <Clock>@<Frequency> (set update clock '0..4' & '7..8' 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)
-S ClockSource (Select Clock Source 'g' or 'o')
 -Sg (Select Clock Generator Source)
 -So (Select Clock Oscillator Source)
-t <TolerancePPT> (Tolerance in Parts/Trillion -- default is 0.020000 PPT)
-z (Zap display, default is not to zap display)
```

```
e.g. ./ccrtnafc_clock -C 1@300000
 (Set Clock 1 to 300000 SPS - do not change any other running clocks)
 ./ccrtnafc_clock -R -C0@100000 -C4@12345 -t0.5
 (Reset all clocks and then set Clock 0 to 100000 SPS and Clock 4
 to 12345 SPS and 0.5 PPT)
```

Example display:

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

```
local_ptr=0x7ffff7f52000
Board Number [-b]: 0
Delay [-d]: 10 milli-seconds
Loop Count [-l]: ***Forever***
Scan Count: 54

----- Clock Revision -----
Silicon Revision: A2
Base Part Number: 5341
Device Speed Grade: A
Device Revision: B

----- 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 ***

_____ PLL Clock Status (** Firmware Supports PLL Sync **) _____
 Clock Source Selected: Clock Generator Source
 PLL Unlock Error: PLL Never Unlocked [*** GOOD ***]
 PLL Locked Status: PLL Locked on Frequency
 Clock 0 Status: Clock Generator Present [*** SELECTED CLOCK ***]
 Clock 1 Status: Clock Oscillator Present
 Current Active Clock: Clock Generator is Currently Active

_____ Output Clock Setting _____
 User output clock frequency 0: 100000.000 Samples/Second/Channel
 User output clock frequency 1: *** Not Set ***
 User output clock frequency 2: *** Not Set ***
 User output clock frequency 3: *** Not Set ***
 User output clock frequency 4: 12345.000 Samples/Second/Channel
 Clock Generator output clock frequency 5: 100000000.000 Samples/Second/Channel
 External output clock frequency 6: *** Not Set ***
 HighSpeed DC1 output clock frequency 7: *** Not Set ***
 HighSpeed DC1 & DC2 output clock frequency 8: *** Not Set ***
 Feed-Back output clock frequency 9: 1000000.000 Samples/Second/Channel

```

### 3.2.8 lib/ccrtnngfc\_dac

This test is useful in programming the DAC interface and displaying the DAC registers.

```

Usage: ./ccrtnngfc_dac [-A] [-a RollingAve] [-b BoardNo] [-C DacUpdateClock] [-d Delay]
[-E ExpInpVolt] [-f DataFormat] [-F DebugFile] [-l LoopCnt]
[-M SelectDacModule] [-n NumChans] [-o OutputSelect]
[-s InputSignal] [-v DacVoltage] [-Z]
-A (Perform DAC Auto Calibration first using reference voltage)
-a RollingAve (Rolling average -- default "=== None ===")
-b BoardNo (Board number -- default is 0)
-C DacUpdateClock (select DAC update clock, 0..4 or 'n|N')
 -C 0,3,4 (Ch0..3=Clock0, Ch4..7=Clock3, Ch8..11=Clock4 at MAX SPS)
 -C ^2,3 (Ch0..3=Clock2, Ch4..11=Clock3 assignment only) [** Skip Programming
 Clock **]
 -C 3@20000.0/n (Ch0..3=Clock3 at 20000 SPS, Ch4..11=No Clock)
 -C 4 (Ch0..11=Clock4 at MAX SPS)
 -C 4@150000.0 (Ch0..11=Clock4) at 150000 SPS)
-d Delay (Delay between screen refresh -- default is 0 milli-seconds)
-E <ExpInpVolts>@<Tol> (Expected Input Volts@Tolerance -- default Tol=0.006000)
 +@<Tol> (Positive Calibration Ref Volt@Tolerance)
 -@<Tol> (Negative Calibration Ref Volt@Tolerance)
 c@<Tol> (DAC Channel 0 Volt@Tolerance)
 s@<Tol> (Requires '-s' input signal option to specify voltage Volt@Tolerance)
 (valid '-s' arguments are 'g','+', '-', 'f', 't')
Note: (For differential bipolar, even channels, voltage read is half supplied)
 (For differential bipolar, odd channels, voltage read is neg. half
 supplied)
-f DataFormat (select data format, '2' or 'b')
 -f b,2,b (Ch0..1 & Ch4..11=Offset binary, Ch2..3=Two's complement)
 -f 2/b (Ch0..1=Two's complement, Ch4..11=Offset binary)
 -f b (Ch0..11=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 ===")
-j SpeedMode (DAC Speed Mode: 'h' or 'n' [high/normal])
-l LoopCnt (Loop count -- default is 0)
-M SelectDacModule (Select DAC Module -- default is module 0)
-n NumChans (Number of channels (1..12) -- default is 12)
-o OutputSelect (DAC output select, 's' or 'd')
 -o d,s (Ch0..1=differential, Ch2..11=single_ended)
 -o s/d,s (Ch0..1 & Ch4..11=single_ended, Ch2..3=differential)
 -o d (Ch0..11=differential)
-s InputSignal (ADC select input signal, 'a', 'e', 'g', '+', '-', 'f', '0..11')
 -s a,e (Ch0..3=All DAC Channels 0..7, Ch8..11=External ADC Input)
 -s e,g (Ch0..7=External ADC input, Ch8..11=ground calibration)
 -s +/e (Ch0..7=Postive calibration, Ch8..11=External ADC input)
 -s - (Ch0..11=Negative calibration)
 -s e/f (Ch0..3=External ADC input, Ch4..11=4 volt calibration)
 -s e/11 (Ch0..3=External ADC input, Ch4..11=DAC Channel 11)
-v DacVoltage (DAC Voltage. -10.0 to +10.0)
 -v 1.5,9.9 (Ch0=1.5 volts, Ch1..11= 9.9 volts)
 -v2.5/7.5,9.7 (Ch0=2.5 volts, Ch1=7.5 volts, Ch2..11=9.7 volts)
 -v 9.95 (Ch0..11=9.95 volts)
-Z (Display Calibration Offset & Gain Channels)

e.g. ./ccrtnngfc_dac -os -s7 -v4.5 -E4.5 (Internal Loopback Testing. Generate 4.5V and
 compare)
 ./ccrtnngfc_dac -os -se -v4.5 -E4.5 (External DAC/ADC Loopback Testing. Generate
 4.5V and compare)
 ./ccrtnngfc_dac -od -s2 -v5.0 -E2.5 (Internal Loopback Testing. Generate 5.0V and
 compare diff 2.5V)
 ./ccrtnngfc_dac -od -s3 -v5.0 -E-2.5 (Internal Loopback Testing. Generate 5.0V and
 compare diff -2.5V)
 ./ccrtnngfc_dac -os -sa -v1,2,3,4 -a100 (display all DAC 0..11 channels with rolling
 average of 100)
 ./ccrtnngfc_dac -os -sa -v3.5 -E3.5@0.01 (Internal Loopback Testing. Generate 3.5V and
 compare diff on all chans)
 ./ccrtnngfc_dac -C0 -s- -Es (Max Clock, -9.91V input, validate against -9.91V)

```

If you wish to run both ADC and DAC concurrently, you can try the following:

- ```

-----
1) ./ccrtnngfc_clock -C0@4500000 -C1@1000000 -11 (Program ADC Clock 0 at 4.5 MHz and DAC Clock 1
      at 1.0 MHz)
2) ./ccrtnngfc_dac -C^1 (Start DAC using 1 MHz Clock 1 in first window)
3) ./ccrtnngfc_adc -C^0 (Start ADC using 4.5 MHz Clock 0 in second
      window)

```

Example display:

```
./ccrtnngfc_dac -A -os -s7 -Vb5 -v4.5 -E4.5@0.010 -C0
```

```
local_ptr=0x7ffff7e67000
```

Physical Memory Information:

```

UserPID          =352833
PhysMemPtr       =0x65d26000
DriverVirtMemPtr =0xffff9a30e5d26000
MmappedUserMemPtr =0x7ffff7fdd000
PhysMemSize      =0x00001000
PhysMemSizeFreed =0x00000000
EntryInTxTbl     =0
NumOfEntriesUsed =1
Flags            =0x0000

```

DAC Information:

Flags = 0x00000001
ModuleNumber = 0
Handle = 0x7819a0
local_ptr = 0x7ffff7e67000
local_dac_ptr = 0x7ffff7e6c000
local_dac_fifo_ptr = 0x7ffff7e7b000
DacFp = 4
DacDeviceName = /dev/ccrtnngfc0_dac0

Auto Calibration started...done. (3.019 seconds)

Board Number [-b]: 0
Update Clock Selected [-C]: Ch00..03 OutputClock=0 (0x7) (450000.000 SPS)
Delay [-d]: 0 milli-seconds
Expected Input Volts [-E]: 4.500000 volts (Tolerance 0.100000 volts)
DAC Data Format [-f]: Ch00..01=Obin Ch02..03=Obin Ch04..05=Obin Ch06..07=Obin Ch08..09=Obin
DAC Speed Mode [-j]: Normal Speed
DAC Output Select [-o]: Ch00..01=Sngl Ch02..03=Sngl Ch04..05=Sngl Ch06..07=Sngl Ch08..09=Sngl
ADC Input Signal [-s]: Ch00..03 [1]Calibration Input (0x27: DAC Channel 7)
DAC Voltage [-v]: Ch00=4.50 Ch01=4.50 Ch02=4.50 Ch03=4.50 Ch04=4.50 Ch05=4.50
DAC Voltage Range [-V]: Ch00..01=b10 Ch02..03=b10 Ch04..05=b10 Ch06..07=b10 Ch08..09=b10 Ch10..11=b10
Loop Count [-l]: ***Forever***
SelectedDacModule [-M]: 0
Number of Channels [-n]: 12
Scan Count : 157703 (0:00:00:27)
Tolerance Exceeded Count : 0 (=== Passed ===)
Read Duration (microsecs) : 25.497 (min= 25.333/max= 41.804/ave= 25.716)

Raw Data (DAC Channels)

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
b99a b99a b99a b99a b99a b99a b99a b99a b99a b99a
[1] b99a b99a

Volts (DAC Channels)

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
+4.5001 +4.5001 +4.5001 +4.5001 +4.5001 +4.5001 +4.5001 +4.5001 +4.5001 +4.5001
[1] +4.5001 +4.5001

Raw Data (ADC_Readback, ADC_Readback, ADC_Readback - 4.50v)

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
-- -- -- -- -- -- -- 804f -- --
[1] -- --

Volts (ADC_Readback, ADC_Readback, ADC_Readback - 4.50v)

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
--- --- --- --- --- --- --- +0.0241 --- ---
[1] --- ---

```

=====
Date: Wed Jun 14 08:18:57 2023
Expected Input Volts: 4.500000 volts (Tolerance 0.100000 volts)
Tolerance Exceed Count: 0
Scan Counter: 470894
WorstMinChanVoltsHWM: 0.020264 (Ch07)
WorstMaxChanVoltsHWM: 0.027283 (Ch07)
=====

```

```

<----- (volts) ----->
Chan  Min      Max      Ave      TolerExceededCnt
=====  =====  =====  =====  =====
07     0.0203    0.0273    0.0236         -
=====

```

```
./ccrtnngfc_dac -C0 -Z
```

```
local_ptr=0x7ffff7e67000
```

Physical Memory Information:

```

UserPID           =352843
PhysMemPtr        =0x65d26000
DriverVirtMemPtr  =0xffff9a30e5d26000
MmappedUserMemPtr =0x7ffff7fdd000
PhysMemSize       =0x00001000
PhysMemSizeFreed  =0x00000000
EntryInTxTbl      =0
NumOfEntriesUsed  =1
Flags             =0x0000

```

DAC Information:

```

Flags             = 0x00000001
ModuleNumber      = 0
Handle            = 0x7819a0
local_ptr         = 0x7ffff7e67000
local_dac_ptr     = 0x7ffff7e6c000
local_dac_fifo_ptr = 0x7ffff7e7b000
DacFp             = 4
DacDeviceName     = /dev/ccrtnngfc0_dac0

```

```

Board Number      [-b]: 0
Update Clock Selected [-C]: Ch00..03 OutputClock=0 (0x7) (450000.000 SPS)
                  : Ch04..07 OutputClock=0 (0x7) (450000.000 SPS)
                  : Ch08..11 OutputClock=0 (0x7) (450000.000 SPS)
Delay             [-d]: 0 milli-seconds
Expected Input Volts [-E]: === Not Specified ===
DAC Data Format    [-f]: Ch00..01=Obin Ch02..03=Obin Ch04..05=Obin Ch06..07=Obin Ch08..09=Obin
                  : Ch10..11=Obin
DAC Speed Mode    [-j]: Normal Speed
DAC Output Select [-o]: Ch00..01=Sngl Ch02..03=Sngl Ch04..05=Sngl Ch06..07=Sngl Ch08..09=Sngl
                  : Ch10..11=Sngl
ADC Input Signal  [-s]: Ch00..03 [1]Calibration Input (0x27: DAC Channel 7)
                  : Ch04..07 [1]Calibration Input (0x27: DAC Channel 7)
                  : Ch08..11 [1]Calibration Input (0x27: DAC Channel 7)
DAC Voltage       [-v]: Ch00=99.00 Ch01=99.00 Ch02=99.00 Ch03=99.00 Ch04=99.00 Ch05=99.00
                  : Ch06=99.00 Ch07=99.00 Ch08=99.00 Ch09=99.00 Ch10=99.00 Ch11=99.00
DAC Voltage Range [-V]: Ch00..01=b10 Ch02..03=b10 Ch04..05=b10 Ch06..07=b10 Ch08..09=b10 Ch10..11=b10

Loop Count        [-l]: ***Forever***
SelectedDacModule [-M]: 0
Number of Channels [-n]: 12
Scan Count        : 69521          (0:00:00:21)

```

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) : 25.584 (min= 25.385/max= 38.632/ave= 25.735)

```
##### Raw Data (Offset Calibration DAC Channels) #####
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
====
[0] fff7 ffeb ffed ffe5 ffe2 ffe3 fff2 fff8 ffee ffe9
[1] ffe0 ffe6
```

```
##### Volts (Offset Calibration DAC Channels) #####
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] -0.0027 -0.0064 -0.0058 -0.0082 -0.0092 -0.0089 -0.0043 -0.0024 -0.0055 -0.0070
[1] -0.0098 -0.0079
```

```
##### Raw Data (Negative Calibration DAC Channels) #####
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
====
[0] 5c3a 1ea3 64bd 950b ccfc f7b2 e582 9bf0 c084 f7f6
[1] a517 ff9d
```

```
##### Volts (Negative Calibration DAC Channels) #####
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] +0.9973 +0.9972 +0.9973 +0.9971 +0.9970 +0.9970 +0.9972 +0.9974 +0.9970 +0.9971
[1] +0.9971 +0.9972
```

```
##### Raw Data (Positive Calibration DAC Channels) #####
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
====
[0] a7b3 0bc3 0460 1f36 0943 edff 4957 967c 7b0b f99e
[1] f51d 4af6
```

```
##### Volts (Positive Calibration DAC Channels) #####
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] +0.9983 +0.9983 +0.9984 +0.9982 +0.9981 +0.9982 +0.9983 +0.9983 +0.9981 +0.9983
[1] +0.9982 +0.9984
```

```
##### Raw Data (DAC Channels) #####
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
====
[0] b99a b99a b99a b99a b99a b99a b99a b99a b99a b99a
[1] b99a b99a
```

```
##### Volts (DAC Channels) #####
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] +4.5001 +4.5001 +4.5001 +4.5001 +4.5001 +4.5001 +4.5001 +4.5001 +4.5001 +4.5001
[1] +4.5001 +4.5001
```

```
##### Raw Data (ADC_Readback, ADC_Readback, ADC_Readback) #####
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
====
[0] -- -- -- -- -- -- -- b9eb -- --
[1] -- --
```

```
##### Volts (ADC_Readback, ADC_Readback, ADC_Readback) #####
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] --- --- --- --- --- --- --- +4.5248 --- ---
[1] --- ---
```

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.

```

=====
Date: Wed Jun 14 08:22:45 2023
Expected Input Volts: === Not Specified ===
Scan Counter: 206508
WorstMinChanVoltsHWM: 4.519958 (Ch07)
WorstMaxChanVoltsHWM: 4.526978 (Ch07)
=====
<----- (volts) ----->
Chan  Min      Max      Ave      TolerExceededCnt
=====
07    4.5200    4.5270    4.5235    -
=====

```

3.2.9 lib/ccrtnngfc_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: ./ccrtnngfc_dac_calibrate [-A] [-b board] [-c ChanMask] [-f DataFormat]
      [-i inCalFile] [-M SelectDacModule] [-o outCalFile] [-R]
-A          (perform Auto Calibration)
-b <board>  (board #, default = 0)
-c <ChanMask> (channel selection mask, default = all channels)
-f DataFormat (select data format, '2' or 'b')
  -f b,2,b   (Ch0..1 & Ch4..11=Offset binary, Ch2..3=Two's complement)
  -f 2/b     (Ch0..1=Two's complement, Ch4..11=Offset binary)
  -f b       (Ch0..11=Offset binary)
-i <In Cal File> (input calibration file [input->board_reg])
-M SelectDacModule (Select DAC Module -- default is module 0)
-o <Out Cal File> (output calibration file [board_reg->output])
-R          (reset DAC calibration)

```

```

e.g. ./ccrtnngfc_dac_calibrate          (Dump calibration information to stdout)
     ./ccrtnngfc_dac_calibrate -A -o Calfile (Perform Auto calibration and dump information
     to 'Calfile')
     ./ccrtnngfc_dac_calibrate -i Calfile  (Update board calibration with supplied
     'Calfile')

```

Example display:

```

./ccrtnngfc_dac_calibrate -A -oOutputCal

DAC Information:
  Flags          = 0x00000001
  ModuleNumber   = 0
  Handle         = 0x76c5a0
  local_ptr      = 0x7ffff7f55000
  local_dac_ptr  = 0x7ffff7f5a000
  local_dac_fifo_ptr = 0x7ffff7f69000
  DacFp         = 4
  DacDeviceName  = /dev/ccrtnngfc0_dac0

Device Name      : /dev/ccrtnngfc0
Board Serial No: 706503 (0x000ac7c7)
Auto Calibration started...done. (3.026 seconds)

==> Dump of 'OutputCal' file
#Date           : Wed Jun 14 08:26:33 2023

#Chan  Negative          Offset          Positive

```

```

#====  =====  =====  =====
ch00:  0.99730931594967842102 -0.00213623046875000000  0.99823938077315688133
ch01:  0.99721193406730890274 -0.00640869140625000000  0.99833755288273096085
ch02:  0.99734734417870640755 -0.00549316406250000000  0.99839489161968231201
ch03:  0.99713751301169395447 -0.00793457031250000000  0.99821293447166681290
ch04:  0.99702146509662270546 -0.00885009765625000000  0.99811642197892069817
ch05:  0.99705506581813097000 -0.00854492187500000000  0.99814362032338976860
ch06:  0.99724035896360874176 -0.00396728515625000000  0.99825175246223807335
ch07:  0.99738377984613180161 -0.00213623046875000000  0.99833340803161263466
ch08:  0.99702031910419464111 -0.00518798828125000000  0.99806922534480690956
ch09:  0.99714408721774816513 -0.00671386718750000000  0.99827769445255398750
ch10:  0.99705672170966863632 -0.00976562500000000000  0.99821040173992514610
ch11:  0.99724986264482140541 -0.00762939453125000000  0.99835348222404718399

```

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

3.2.10 lib/ccrtngfc_dac_fifo

This test performs validation of the Multi-Function DAC FIFO operation of the card. If two daughter cards are present, both are run concurrently.

```

Usage: ./ccrtngfc_dac_fifo [-b board] [-c ChannelSelectMask] [-C DacUpdateClock]
      [-e MsgDmaEngine] [-f format] [-j SpeedMode] [-l LoopCnt]
      [-M SelectDacModule] [-N] [-o OutputSelect] [-S NumSamples]
      [-u UpdateMode] [-v OutputVolts] [-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..4 for each module. If '^' first arg,
  Skip Programming Clock)
  -C 2@20000.0 (Ch0..11=Clock2 at 20000 SPS for both modules)
  -C 4         (Ch0..11=Clock4 at MAX SPS for both modules)
  -C ^3       (Ch0..11=Clock3 both modules assignment only) [** Skip Programming
  Clock **]
  -C 1@100000,2@200000 (Ch0..11=Clock1 at 100000 SPS for module 1 and Clock 2 @200000 SPS
  for module 2)
-e MsgDmaEngine (Select MsgDma Engine -- default "=== get free engine ===")
  -e 0,2        (MsgDma Engine 0 for DAC Module 0 and MsgDma Engine 2 for DAC Module 1)
  -e 3,1        (MsgDma Engine 3 for DAC Module 0 and MsgDma Engine 1 for DAC Module 1)
  -e ,3        (MsgDma Engine Default for DAC Module 0 and MsgDma Engine 3 for DAC
  Module 1)
-f DataFormat  (select data format, '2' or 'b')
  -f b,2,b     (Ch0..1 & Ch4..11=Offset binary, Ch2..3=Two's complement)
  -f 2/b       (Ch0..1=Two's complement, Ch4..11=Offset binary)
  -f b         (Ch0..11=Offset binary)
-j SpeedMode   (DAC Speed Mode: 'h' or 'n' [high/normal])
-l LoopCnt     (Loop count -- default is 0)
-M SelectDacModule (Select DAC Module -- default is all available modules)
-o OutputSelect (DAC output select, 's' or 'd')
  -o d,s       (Ch0..1=differential, Ch2..11=single_ended)
  -o s/d,s     (Ch0..1 & Ch4..11=single_ended, Ch2..3=differential)
  -o d         (Ch0..11=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 +10.0)
  -v 1.5,9.9  (Ch0=1.5 volts, Ch1..11= 9.9 volts)
  -v 2.5/7.5,9.7 (Ch0=2.5 volts, Ch1=7.5 volts, Ch2..11=9.7 volts)
  -v 9.95     (Ch0..11=9.95 volts)
-w <WaveType> (default = 's' Sine Wave)

```

```

-wu          (Saw Wave [up])
-wd          (Saw Wave [down])
-ws          (Sine Wave)
-wx          (Square Wave)
-wX          (Square Wave - Alternate Sample)
-wy          (Step Wave [down])
-wz          (Step Wave [up])
-wt          (Triangle Wave)
-ww          (All Wave [Sine/Square/StepUp/Triangle/StepDown])

```

Note: If high speed mode is enabled, only even channels are used
 If high speed mode is enabled, maximum clock speed is 2000000.0 SPS, otherwise it is 1000000.0 SPS
 If '^' is specified as first argument to '-C' option, then skip programming clocks

```

e.g. ./ccrtnngfc_dac_fifo -od,s,d,s -v8,10,5,1 -wx          (dac0&2 differential, dac1&3 single_ended)
     ./ccrtnngfc_dac_fifo -ws -ui -od                      (sine wave, immediate)

```

If you wish to run both ADC and DAC concurrently, you can try the following:

- 1) ./ccrtnngfc_clock -C0@4500000 -C1@1000000 -l1 (Program ADC Clock 0 at 4.5 MHz and DAC Clock 1 at 1.0 MHz)
- 2) ./ccrtnngfc_dac_fifo -C^1 (Start DAC Fifo using Clock at 1 in first window)
- 3) ./ccrtnngfc_adc_fifo -C^0 (Start ADC Fifo using Clock at 0 in second window)

Example display:

```
./ccrtnngfc_dac_fifo -C0
```

```
MyHandle=0xa37f700
```

```
DacModule0: Programming Clock0: Desired Frequency 1000000.000, Tolerance=0.020000 PPT
```

```
DacModule1: Programming Clock0: Desired Frequency 1000000.000, Tolerance=0.020000 PPT
```

```
StartChannelNumber=0
```

```
EndChannelNumber =11
```

```

DAC Information:
  Flags          = 0x00000001
  ModuleNumber   = 0
  Handle         = 0xa37f700
  local_ptr      = 0x7ffff7f52000
  local_dac_ptr  = 0x7ffff7f57000
  local_dac_fifo_ptr = 0x7ffff7f66000
  DacFp         = 4
  DacDeviceName = /dev/ccrtnngfc0_dac0

```

```
#### Output Clock 0 Frequency = 1000000.000 Samples/Second/Channel
```

```
DAC 0....
```

```

  State = 0x0 (Idle)
  Update Mode = 0x0 (Immediate)
  Data Format = 0x0 (Offset Binary)
  Output Select = 0x0 (Single-Ended)
  Output Range = 10

```

```
DAC 1....
```

```

  State = 0x0 (Idle)
  Update Mode = 0x0 (Immediate)
  Data Format = 0x0 (Offset Binary)
  Output Select = 0x0 (Single-Ended)

```



```

    Output Range = 10
DAC 2....
    State = 0x0 (Idle)
    Update Mode = 0x0 (Immediate)
    Data Format = 0x0 (Offset Binary)
    Output Select = 0x0 (Single-Ended)
    Output Range = 10
DAC 3....
    State = 0x0 (Idle)
    Update Mode = 0x0 (Immediate)
    Data Format = 0x0 (Offset Binary)
    Output Select = 0x0 (Single-Ended)
    Output Range = 10
DAC 4....
    State = 0x0 (Idle)
    Update Mode = 0x0 (Immediate)
    Data Format = 0x0 (Offset Binary)
    Output Select = 0x0 (Single-Ended)
    Output Range = 10
DAC 5....
    State = 0x0 (Idle)
    Update Mode = 0x0 (Immediate)
    Data Format = 0x0 (Offset Binary)
    Output Select = 0x0 (Single-Ended)
    Output Range = 10
DacModule0: DAC Update Selection (7): DacModule0: Clock Generator 0 Selected (Desired
Frequency=1000000.00)

StartChannelNumber=0
EndChannelNumber  =11

```

```

DAC Information:
    Flags           = 0x00000001
    ModuleNumber    = 1
    Handle          = 0xa37f700
    local_ptr       = 0x7ffff7f52000
    local_dac_ptr   = 0x7ffff7f59000
    local_dac_fifo_ptr = 0x7ffff7f6e000
    DacFp          = 5
    DacDeviceName   = /dev/ccrtngfc0_dac1

```

```

#### Output Clock 0 Frequency = 1000000.000 Samples/Second/Channel
DAC 0....
    State = 0x0 (Idle)
    Update Mode = 0x0 (Immediate)
    Data Format = 0x0 (Offset Binary)
    Output Select = 0x0 (Single-Ended)
    Output Range = 10
DAC 1....
    State = 0x0 (Idle)
    Update Mode = 0x0 (Immediate)
    Data Format = 0x0 (Offset Binary)
    Output Select = 0x0 (Single-Ended)
    Output Range = 10
DAC 2....
    State = 0x0 (Idle)
    Update Mode = 0x0 (Immediate)
    Data Format = 0x0 (Offset Binary)
    Output Select = 0x0 (Single-Ended)
    Output Range = 10

```

```

DAC 3....
    State = 0x0 (Idle)
    Update Mode = 0x0 (Immediate)
    Data Format = 0x0 (Offset Binary)
    Output Select = 0x0 (Single-Ended)
    Output Range = 10
DAC 4....
    State = 0x0 (Idle)
    Update Mode = 0x0 (Immediate)
    Data Format = 0x0 (Offset Binary)
    Output Select = 0x0 (Single-Ended)
    Output Range = 10
DAC 5....
    State = 0x0 (Idle)
    Update Mode = 0x0 (Immediate)
    Data Format = 0x0 (Offset Binary)
    Output Select = 0x0 (Single-Ended)
    Output Range = 10
DacModule1: DAC Update Selection (7): DacModule1: Clock Generator 0 Selected (Desired
Frequency=1000000.00)
===== CpuCountAssignedToThisTask=12

Device Name      : /dev/ccrtngfc0
Physical Memory Information:
  UserPID          =354912
  PhysMemPtr       =0x65e78000
  DriverVirtMemPtr =0xffff9a30e5e78000
  MmappedUserMemPtr =0x7ffff7f4c000
  PhysMemSize      =0x00006000
  PhysMemSizeFreed =0x00000000
  EntryInTxTbl    =0
  NumOfEntriesUsed =1
  Flags           =0x0000

#### Output Clock 0 Frequency = 1000000.000 Samples/Second/Channel
#### Expected Transfer Rate   =      4.000 MBytes/Second
#### Expected Wave Period     =      0.512 msec
#### Expected Wave Frequency   =    1953.125 Hz
    Speed Mode: 0 (Normal speed)
    Channel Mask: 0xfff
    Samples/Write: 512
Voltage Selection: Ch00=10.00 Ch01=10.00 Ch02=10.00 Ch03=10.00 Ch04=10.00 Ch05=10.00
                  : Ch06=10.00 Ch07=10.00 Ch08=10.00 Ch09=10.00 Ch10=10.00 Ch11=10.00
Generating a continuous Sine Wave on selected channels: <CTRL-C> to abort

### Board 0: All Selected Channels Output: (Raw=0xffffe Volts= 9.999)

DAC0: Eng0: 512.266 usec/write: 3.998 MBytes/sec, 511.998 usec period, 1.953 KHz (fifo=129696 - 98.95%)
DAC1: Eng1: 512.987 usec/write: 3.992 MBytes/sec, 512.006 usec period, 1.953 KHz (fifo=129660 - 98.92%)

#### DISABLING CLOCK OUTPUT ####
DAC0: DaughterCardThreadId=140737333200640
DACHandle=0xa737140
##### DacStartCount=1 thread=140737333200640
DAC1: DaughterCardThreadId=140737324807936
DACHandle=0xa73b198
##### DacStartCount=2 thread=140737324807936
#### ENABLING CLOCK OUTPUT ####
#### DISABLING CLOCK OUTPUT ####

```

3.2.11 lib/ccrtngfc_dac_setchan

This test generates voltages on various Analog Output channels.

```
Usage: ./ccrtngfc_dac_setchan [-b board] [-c ChannelSelectMask] [-C DacUpdateClock]
      [-e MsgDmaEngine] [-f format] [-i] [-j SpeedMode] [-l LoopCnt]
      [-m WriteMode] [-M SelectDacModule] [-N] [-o OutputSelect]
      [-S NumSamples] [-u UpdateMode] [-v OutputVolts] [-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..4 or 's|S')
  -C s      (Ch0..11=Software Update)
  -C ^2    (Ch0..11=Clock2 assignment only) [** Skip Programming Clock **]
  -C 3@20000.0 (Ch0..11=Clock3 at 20000 SPS)
  -C 4      (Ch0..11=Clock4 at MAX SPS)
-e MsgDmaEngine (Select MsgDma Engine -- default "=== get free engine ===")
-f DataFormat (select data format, '2' or 'b')
  -f b,2,b  (Ch0..1 & Ch4..11=Offset binary, Ch2..3=Two's complement)
  -f 2/b    (Ch0..1=Two's complement, Ch4..11=Offset binary)
  -f b      (Ch0..11=Offset binary)
-i          (Enable Interrupts -- default = Disable)
-j SpeedMode (DAC Speed Mode: 'h' or 'n' [high/normal])
-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)
  -mlp      (Library: [Channel Registers] PIO mode)
  -mlm      (Library: [Channel Registers] Modular scatter-gather DMA mode)
  -mlx      (Library: [Channel Registers] Clone [MEM->DAC] Modular
            scatter-gather DMA mode)
  -mup      (User: [Channel Registers] PIO mode)
  -mdP      (Driver: [FIFO] PIO mode)
  -mlM      (Library: [FIFO] Modular scatter-gather DMA mode)
  -mlP      (Library: [FIFO] PIO mode)
-M SelectDacModule (Select DAC Module -- default is module 0)
-N          (Open device with O_NONBLOCK flag for driver operations)
-o OutputSelect (DAC output select, 's' or 'd')
  -o d,s    (Ch0..1=differential, Ch2..11=single_ended)
  -o s/d,s  (Ch0..1 & Ch4..11=single_ended, Ch2..3=differential)
  -o d      (Ch0..11=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 +10.0)
  -v 1.5,9.9 (Ch0=1.5 volts, Ch1..11= 9.9 volts)
  -v 2.5/7.5,9.7 (Ch0=2.5 volts, Ch1=7.5 volts, Ch2..11=9.7 volts)
  -v 9.95    (Ch0..11=9.95 volts)
-w <WaveType> (default = 'c' Constant Voltage)
  -wc      (Constant Voltage)
  -wu      (Saw Wave [up])
  -wd      (Saw Wave [down])
  -ws      (Sine Wave)
  -wx      (Square Wave)
  -wX      (Square Wave - Alternate Sample)
  -wy      (Step Wave [down])
  -wz      (Step Wave [up])
  -wt      (Triangle Wave)
  -ww      (All Wave [Sine/Square/StepUp/Triangle/StepDown])
```

Note: If high speed mode is enabled, only even channels are used

If high speed mode is enabled, maximum clock speed is 2000000.0 SPS, otherwise it is 1000000.0 SPS

```
e.g. ./ccrtnngfc_dac_setchan -od,s,d,s -v8,10,5,1 -wx -mdp (dac0&2 differential, dac1&3
                                         single_ended)
      ./ccrtnngfc_dac_setchan -ws -ui -od                (sine wave, immediate)
```

If you wish to run both ADC and DAC concurrently, you can try the following:

- ```

1) ./ccrtnngfc_clock -C0@4500000 -C1@1000000 -l1 (Program ADC Clock 0 at 4.5 MHz and DAC
 Clock 1 at 1.0 MHz)
2) ./ccrtnngfc_dac_setchan -C^1 -ws -mlM (Start DAC Chan using 1 MHz Clock 1 in
 first window)
3) ./ccrtnngfc_adc -C^0 -mlP (Start ADC using 4.5 MHz Clock 0 in second
 window)
```

Example display:

```
./ccrtnngfc_dac_setchan -ws -ui -od
```

```
Device Name : /dev/ccrtnngfc0
```

```
StartChannelNumber=0
EndChannelNumber =11
```

Physical Memory Information:

```
UserPID =355245
PhysMemPtr =0x65e78000
DriverVirtMemPtr =0xffff9a30e5e78000
MmappedUserMemPtr =0x7ffff7f4c000
PhysMemSize =0x00006000
PhysMemSizeFreed =0x00000000
EntryInTxTbl =0
NumOfEntriesUsed =1
Flags =0x0000
```

DAC Information:

```
Flags = 0x00000001
ModuleNumber = 0
Handle = 0xa37ec60
local_ptr = 0x7ffff7f52000
local_dac_ptr = 0x7ffff7f57000
local_dac_fifo_ptr = 0x7ffff7f66000
DacFp = 4
DacDeviceName = /dev/ccrtnngfc0_dac0
```

```
Output Clock 0 Frequency = 1000000.000 Samples/Second/Channel
DAC 0....
```

```
State = 0x0 (Idle)
Update Mode = 0x0 (Immediate)
Data Format = 0x0 (Offset Binary)
Output Select = 0x1 (Differential)
Output Range = 20
```

```
DAC 1....
```

```
State = 0x0 (Idle)
Update Mode = 0x0 (Immediate)
Data Format = 0x0 (Offset Binary)
Output Select = 0x1 (Differential)
Output Range = 20
```

```
DAC 2....
```

```
State = 0x0 (Idle)
```

```

 Update Mode = 0x0 (Immediate)
 Data Format = 0x0 (Offset Binary)
 Output Select = 0x1 (Differential)
 Output Range = 20
DAC 3....
 State = 0x0 (Idle)
 Update Mode = 0x0 (Immediate)
 Data Format = 0x0 (Offset Binary)
 Output Select = 0x1 (Differential)
 Output Range = 20
DAC 4....
 State = 0x0 (Idle)
 Update Mode = 0x0 (Immediate)
 Data Format = 0x0 (Offset Binary)
 Output Select = 0x1 (Differential)
 Output Range = 20
DAC 5....
 State = 0x0 (Idle)
 Update Mode = 0x0 (Immediate)
 Data Format = 0x0 (Offset Binary)
 Output Select = 0x1 (Differential)
 Output Range = 20

 Write Mode: -mlp: Library: (CHANNEL) PIO Mode
 Speed Mode: 0 (Normal speed)
 Channel Mask: 0xfff
 Samples/Write: 512 (per active channel)
Voltage Selection: Ch00=20.00 Ch01=20.00 Ch02=20.00 Ch03=20.00 Ch04=20.00 Ch05=20.00
 : Ch06=99.00 Ch07=99.00 Ch08=99.00 Ch09=99.00 Ch10=99.00 Ch11=99.00
Generating a continuous Sine Wave on selected channels: <CTRL-C> to abort

5795531: 4.307 usec/write: 928.721 KBytes/sec, 2.309 msec period, 433.007 Hz

```

### 3.2.12 lib/ccrtnngfc\_daughtercard\_info

This test returns information that is located in the EEPROMs on the daughter cards.

```

Usage: ./ccrtnngfc_daughtercard_info [-b Board] [-d DaughterCard]
-b Board (Board number -- default is 0)
-d DaughterCard (Daughter Card Selection -- default is 0)

```

#### Example display:

```

./ccrtnngfc_daughtercard_info

===== Daughter Card 0 =====

(0.128 seconds)
#####
 Board Number: 0
 Daughter Card: 0
 Daughter Card Id: 1
 Daughter Card Assembly: 1579321-901
 Revision: 'D'
 Serial Number: 729514
 Date: 02/21/2023 (mm/dd/yyyy)
 Description: 'High Speed Analog Daughter Card'
 Notes: 'Calibrated 2/21/2023'
#####

===== Daughter Card 1 =====

```

```
(0.128 seconds)
#####
Board Number: 0
Daughter Card: 0
Loaded Firmware Function: 1 (Installed Daughter Card Supported)
Daughter Card Id: 1
Daughter Card Assembly: 1579321-901
Revision: 'D'
Serial Number: 729514
Date: 02/21/2023 (mm/dd/yyyy)
Description: 'High Speed Analog Daughter Card'
Notes: 'Calibrated 2/21/2023'
#####

===== Daughter Card 1 =====
```

```
(0.128 seconds)
#####
Board Number: 0
Daughter Card: 1
Loaded Firmware Function: 1 (Installed Daughter Card Supported)
Daughter Card Id: 1
Daughter Card Assembly: 1579321-901
Revision: 'D'
Serial Number: 729512
Date: 02/21/2023 (mm/dd/yyyy)
Description: 'High Speed Analog Daughter Card'
Notes: 'Calibrated 2/21/2023'
#####
```

### 3.2.13 lib/ccrtnngfc\_dio

This test generates, views and tests various digital channels.

```
Usage: ./ccrtnngfc_dio [-b BoardNo] [-d Delay] [-F DebugFile] [-l LoopCnt]
 [-M SelectLDioModule] [-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 SelectLDioModule (Select LDIO Module -- default is DIO module 0)
-n NumChans (number of channels -- default is 32)
-p PatternSelect (LDIO 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 4)
 -rd (Digital Isolators test)
 -rD (Fast [no cursors] Digital Isolators test)
 -re (External Loopback DIO test - Loopback breakout box required)
 -rE (Fast [no cursors] external loopback DIO test - Loopback breakout
 box required)
 -ri (Read DIO input channels)
 -rI (Internal Loopback DIO test - No inputs/outputs must be connected)
 -rL (Fast [no cursors] internal loopback DIO test - No inputs/outputs
 must be connected)
 -ro (Write pattern to DIO output channels)
```

```

-rt (Digital Terminators test)
-rT (Fast Digital Terminators test)
-s SkipChannelsMask (Skip channels mask -- default is @0x00000000)
-sXXXXXXXX (LDIO Channels 31..00=XXXXXXXX in Hex)
e.g. ./ccrtnngfc_dio -rl -sf0 (Internal Loopback Testing. Skip LDIO Channels 04-07)
 ./ccrtnngfc_dio -rE -sf0 (External Loopback w/o Curses Testing. Skip LIO
 Channels 04-07)

```

Example display:

```
./ccrtnngfc_dio -rl -sF
```

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
Output Sync Mode [-c]: 1 [SYNC]
Delay [-d]: 100 milli-seconds
LDIO Module [-g]: 0
Loop Count [-l]: ***Forever***
Number of Channels [-n]: 32
Pattern Selection [-p]: 2 (Adding Bit)
Run Option [-r]: 5 (Internal Loopback Test)
Skip Channels Mask [-s]: 0x0000000F (31..00)

Channel Mismatch Count : 0 (=== Passed ===)
DIO Direction : 0xFFFFFFFF (All Output)
DIO Channels Terminator : 0x00000000 (All Off)
LDIO Enable : 0x00000001 (Enable)
Input Snapshot : 0x00000001 (Snapshot)
Scan Count : 970

Write Duration (microsecs) : 1.953 (min= 1.844/max= 6.046/ave= 1.968)
Read Duration (microsecs) : 3.928 (min= 3.874/max= 30.171/ave= 3.992)

Channels Output Input Expected
===== ===== ===== =====
30..00 [0]: 000001F0 000001F0 000001F0

 <----- Input Channels ----->
 [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
 === === === === === === === === === ===
[0] skip skip skip skip + + + + + .
[1]
[2]
[3] . .

AvalonPtr=0x7ffff7f52000

Abort Received

=== Test Passed ===

```

### 3.2.14 lib/ccrtnngfc\_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: ./ccrtnngfc_disp [-b Board] [-d Delay] [-e MsgDmaEngine] [-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)
-e MsgDmaEngine (Select MsgDma Engine -- default "=== get free engine ===")
-H (Enable Hyper-Drive Mode -- default "=== Disabled ===")
-i (Enable Interrupts -- default = Disable)
-l LoopCnt (Loop Count - default = 0)
-m XferMode (Transfer Mode -- default = MSGDMA)
 -mm (Avalon Memory: Modular Scatter-Gather DMA mode)
 -mp (Avalon 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:

```
./ccrtnngfc_disp
```

```
local_ptr=0x7ffff7f52000
```

```
Physical Memory Information:
UserPID =186992
PhysMemPtr =0x65d28000
DriverVirtMemPtr =0xffff993ee5d28000
MmappedUserMemPtr =0x7ffff7fdd000
PhysMemSize =0x00001000
PhysMemSizeFreed =0x00000000
EntryInTxTbl =0
NumOfEntriesUsed =1
Flags =0x0000
```

```
Board Number [-b]: 0
Delay [-d]: 0 milli-seconds
MSGDMA Engine [-e]: 0
Hyper-Drive [-H]: Disabled
Interrupts [-i]: Disabled
Loop Count [-l]: ***Forever***
Transfer Mode [-m]: Modular Scatter-Gather DMA I/O (Avalon Memory)
Offset [-o]: 0x00000000
Transfer Size [-s]: 0x00001000 (4096) bytes (146.952 MBytes/Second)
Display Size [-S]: 0x00000200 (512) bytes
```

```
ScanCount : 192177
Read Duration (microsecs) : 27.873 (min= 26.258/max= 40.341/ave= 27.914)
```

```

 00 04 08 0C 10 14 18 1C
=====
000000 93200101 08252023 00010000 00120000 00000000 00000000 00000020 00010001
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

```

### 3.2.15 lib/ccrtngfc\_dma

This test transfers data from physical memory to the Local register area and back. There are three modes of operation. One is regular DMA, the second is Modular Scatter-Gather DMA and the third is programmed I/O. Depending on the number of DMA engines supported by the card, the user can select one of them to perform the DMA. Additionally, if the card supports Modular Scatter-Gather DMA, then they can also select that. Area select is one of three areas the user can specify. They represent the area in physical memory and local register where the transfer is to occur. The test automatically switches to a different area corresponding to the regular DMA engine supplied. If multiple copies of this application is run on the same card using the same DMA engine, then the user needs to manually select a different area '-A' so the data mismatch does not occur due to using the same area region.

```
Usage: ./ccrtngfc_dma [-A Area2Select] [-b Board] [-e MsgDmaEngine] [-i] [-l LoopCnt]
 [-m XferMode] [-s Size] [-v VerboseNo]
```

```

-A Area2Select (Area to select -- default = -1)
-b Board (Board number -- default = 0)
-e MsgDmaEngine (Select MsgDma Engine -- default "=== get free engine ===")
-i (Enable Interrupts -- default = Disable)
-l LoopCnt (Loop Count - default = 1000)
-m XferMode (Transfer Mode -- default = MSGDMA)
 -mm (MsgDma 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. ./ccrtngfc_dma -A1 (perform dma using MsgDMA on area 1)
 ./ccrtngfc_dma -i -mm (perform dma using MsgDMA with interrupts on area 0)
 ./ccrtngfc_dma -mp (perform Programmed I/O on area 0)

```

#### Example display:

```
./ccrtngfc_dma
```

```
Device Name: /dev/ccrtngfc0
```

```
local_ptr=0x7ffff7f55000
```

#### Physical Memory Information:

```

UserPID =186997
PhysMemPtr =0x65800000
DriverVirtMemPtr =0xffff993ee5800000
MmappedUserMemPtr =0x7ffff7065000
PhysMemSize =0x00200000
PhysMemSizeFreed =0x00000000
EntryInTxTbl =0
NumOfEntriesUsed =2
Flags =0x0000

```

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

```
MSGDMA Address[A0]: 0x00100400 - 0x00103400
```

```
Transfer Size: 12288 (0x00003000) bytes (MsgDMA without Interrupts: MsgDma Engine)
```

```
###
```

```
1000: A2P: Total: 36.262us (338.87 MB/s): first=0xface0000 last=0xface0bff
```

| (micro-seconds) |     |     | (MBytes/second) |     |     |
|-----------------|-----|-----|-----------------|-----|-----|
| Min             | Max | Ave | Min             | Max | Ave |

```

P2A: 37.83 43.73 38.97 280.99 324.84 315.34
A2P: 36.15 42.53 36.33 288.95 339.91 338.22

```

### 3.2.16 lib/ccrtngfc\_example

This test provides a simple example of programming ADC, DAC and DIO.

```

Usage: ./ccrtngfc_example [-b Board] [-M SelectModule]
 -b Board (Board number -- default is 0)
 -M SelectModule (Select Module -- default is module 0)

```

#### Example display:

```
./ccrtngfc_example
```

```

DAC Information:
 Flags = 0x00000001
 ModuleNumber = 0
 Handle = 0x48d8c0
 local_ptr = 0x7ffff7800000
 local_dac_ptr = 0x7ffff7805000
 local_dac_fifo_ptr = 0x7ffff7814000
 DacFp = 4
 DacDeviceName = /dev/ccrtngfc0_dac0

```

```

ADC Information:
 Flags = 0x00000001
 ModuleNumber = 0
 Handle = 0x48d8c0
 local_ptr = 0x7ffff7800000
 local_adc_ptr = 0x7ffff7804000
 local_adc_fifo_ptr = 0x7ffff7810000
 AdcFp = 5
 AdcDeviceName = /dev/ccrtngfc0_adc0

```

```
local_ptr=0x7ffff7800000
```

```

Physical Memory Information:
 UserPID =4416
 PhysMemPtr =0x6c1d2000
 DriverVirtMemPtr =0xffff8a782c1d2000
 MmappedUserMemPtr =0x7ffff7fb6000
 PhysMemSize =0x00001000
 PhysMemSizeFreed =0x00000000
 EntryInTxTbl =1
 NumOfEntriesUsed =1
 Flags =0x0000

```

```

Configuring ADC
- Activate ADC
- Configure ADC
- Set Calibration to Positive Reference Voltage
- Calibrate ADC
Configuring DAC
- Activate DAC
- Select Software Update
- Configure DAC
- Write 0 to DAC outputs
- Make DAC operational
- Calibrate DAC
Programming Clocks

```

```

Reading ADC Channels Using Transfer API
- First Seize MsgDma Engine before multiple ADC transfers
- Perform ADC transfers
- Finally Release MsgDma Engine when done
- Dump ADC_channel data

==== ADC Channels - Using ccrtNGFC_Transfer_Data() ==== (length=48)
+DMP+ 0 00007eda 00007ed8 00007edb 00007edb *~..~..~..~..~..~.*
+DMP+ 0x10 00007ee0 00007edc 00007eda 00007ed7 *~..~..~..~..~..~.*
+DMP+ 0x20 00007edb 00007edb 00007edd 00007edd *~..~..~..~..~..~.*

Writing DAC Channels Using Transfer API
- First Seize MsgDma Engine before multiple ADC transfers
- Generate DAC data
DacCh00: 0x80000000 (0.000000 volts)
DacCh01: 0x80000000 (0.000000 volts)
DacCh02: 0x80000000 (0.000000 volts)
DacCh03: 0x80000000 (0.000000 volts)
DacCh04: 0x80000000 (0.000000 volts)
DacCh05: 0x80000000 (0.000000 volts)
DacCh06: 0x80000000 (0.000000 volts)
DacCh07: 0x80000000 (0.000000 volts)
DacCh08: 0x80000000 (0.000000 volts)
DacCh09: 0x80000000 (0.000000 volts)
DacCh10: 0x80000000 (0.000000 volts)
DacCh11: 0x80000000 (0.000000 volts)
- Perform DAC transfers
- Finally Release MsgDma Engine when done

Configuring LIO
- Activate LVDS I/O Module
- Set LDIO output sync mode to SYNC
- Set LDIO input snapshot mode
- Set LIO ports direction

Reading LIO Channels 00..15
- CCRTNGFC_LIO_CHAN_00_15=0x00000000

Writing LIO Channels 16..31
- CCRTNGFC_LIO_CHAN_16_31=0xbabe0000

Configuring DIO
- Activate DIO Module
- Set LDIO output sync mode to SYNC
- Set LDIO input snapshot mode
- Set DIO ports direction

Reading DIO Channels 00..15
- CCRTNGFC_DIO_CHAN_00_15=0x00002000

Writing DIO Channels 16..31
- CCRTNGFC_DIO_CHAN_16_31=0xbabe0000

Single (one descriptor) Modular Scatter-Gather DMA
- Allocating memory and seeding with pattern
- Seizing MSGDMA
- Configure Single MSGDMA (PCIe ==> Avalon)
- Fire Single MSGDMA: Xfer 0x8000 bytes: PCIe ==> Avalon (@0x8000)
- Validating data
- Configure Single MSGDMA (Avalon ==> PCIe)
- Fire Single MSGDMA: Xfer 0x8000 bytes: Avalon (@0x8000) ==> PCIe
- Validating data
- Releasing MSGDMA

Multi (four descriptor) Modular Scatter-Gather DMA (Single-Shot)
- Allocating memory and seeding with pattern
- Seizing MSGDMA
- Configure multi MSGDMA (PCIe ==> Avalon ==> PCIe ==> Avalon ==> PCIe)
- Setup Multi MSGDMA
- Fire Multi MSGDMA (Single-Shot)

```

- Validating data
- Releasing MSGDMA
- ### Multi (four descriptor) Modular Scatter-Gather DMA (Clone) ###
- Allocating memory and seeding with pattern
- Seizing MSGDMA
- Configure multi MSGDMA (PCIe ==> Avalon ==> PCIe ==> Avalon ==> PCIe)
- Setup Multi MSGDMA
- Stop and Initialize Multi MSGDMA (Clone)
- Fire Multi MSGDMA and wait one cycle (Clone: Once cycle wait)
- Validating data
- Releasing MSGDMA

### 3.2.17 lib/ccrtngfc\_expires

This test is useful in displaying board expires information.

```
Usage: ./ccrtngfc_expires -[b Board] -[s]
 -b <board> (board #, default = 0)
 -s (short display, default = verbose)
```

Example display:

```
./ccrtngfc_expires (for card that has no restrictions)
```

```
Device Name: /dev/ccrtngfc0
Board Serial No: 98765 (0x000181cd)
```

```
#####
###
UNRESTRICTED FIRMWARE
###
#####
```

```
./ccrtngfc_expires (for restricted card that has NO expiration date)
```

```
Device Name: /dev/ccrtngfc0
Board Serial No: 98765 (0x000181cd)
```

```
#####
###
RESTRICTED FIRMWARE
###
#####
```

```
=====
=== No Expiration Date ===
=====
```

```
./ccrtngfc_expires (for restricted card that has expiration date)
```

```
Device Name: /dev/ccrtngfc0
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
=====
./ccrtnngfc_expires -s (for card that has no restrictions)

Unrestricted

./ccrtnngfc_expires -s (for restricted card that has NO expiration date)

Restricted: No expiration date

./ccrtnngfc_expires -s (for restricted card that has expiration date)

Restricted: Expire in 10550462 seconds

```

### 3.2.18 lib/ccrtnngfc\_identify

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

```

Usage: ./ccrtnngfc_identify -[absx]
-a (Identify all cards through a light sequence)
-b <board> (board #, default = 0)
-s <seconds> (Identify Board: ENABLED for number of seconds,
 default = 10)
-s 0 (Identify Board: DISABLED)
-s <negative value> (Identify Board: ENABLED forever)
-x (silent)

```

If the '-a' option is selected, all other options are ignored. This option will sequence through all the cards found in turn as follows:

- 1) The first device number will flash its LED for 10 seconds
- 2) The remaining devices numbers will be selected sequentially and flash their LEDs for 3 seconds

#### Example display:

```

./ccrtnngfc_identify

Device Name : /dev/ccrtnngfc0
Board ID : 9320
Board Type : 01
Board Function : 01
Board Serial No : 706503 (0x000ac7c7)
Firmware Revision : 1.0 (Major.Minor)
LDIO Module 0 : DIO
LDIO Module 1 : LIO
MsgDma Support : 31 descriptors (Yes)
Cloning Support : 3 (Yes) Cloning is supported. Region addressing allowed for any user

```

```

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

```

```
./ccrtnngfc_identify -a
```

```

TotalBoardCount=1
DNum IRQ MSI Bu:Sl:Fn VnID:Sub BdID:Ty:Fu:Sub FMaj.Min(mm:dd:yy hh:mm:ss) MC FmFlvCod
FwbRev IPCores Temp:C/F SerialNo RLS# Func
0 0 162 Y b9:00:00 1542:1542 9320.01.01:0100 1.0(08/25/23 12:00:00) 048 00000000
00000000 0 41.6/106.8 706503 350 MultiFunc

```

```
Device Numbers: (enter <CTRL-C> to terminate)
=====
0*
```

### 3.2.19 lib/ccrtngfc\_info

This test is useful in getting information for all the *ccrtngfc* devices in the system.

```
Usage: ./ccrtngfc_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:

```
./ccrtngfc_info

IRQ MSI Bu:Sl:Fn VnID:Sub BdID:Ty:Fu:Sub FMaj.Min(mm:dd:yy hh:mm:ss) MC FmFlvCod FwbRev IPCores Temp:C/F
SerialNo RLS# Func
 0 162 Y b9:00:00 1542:1542 9320.01.01:0100 0001.000(08/25/23 12:00:00) 048 00000000 00000000 0
41.6/106.8 706503 350 MultiFunc
```

```
./ccrtngfc_info -l
```

```
Board 0
Version: 2023.6.0
Build: Wed May 17 09:19:29 EDT 2023
Module: ccrtngfc
Board Index: 0 (PCIe-CCUR_FPGA_NGFC)
Bus: 0xb9
Slot: 0x00
Func: 0x00
Vendor ID: 0x1542
Sub-Vendor ID: 0x1542
Board Info: 0x93200101 (id=9320, type=0x01, func=0x01 (MultiFunc))
Member Code: 1 (048)
Sub-Device ID: 0x0100
Firmware Date/Time: 0x08252023 0x00120000 (08/25/2023 12:00:00)
Firmware Revision: 0x00010000 (1.0)
Fpgawb Revision: 0x00000000 (0000.00-00) (Not Supported)
Firmware Flavor Code: 0x00000000 (0) (****)
Number of Advanced IP Cores: 0x00000000 (0)
Board Serial Number: 0x000ac7c7 (706503)
FPGA Chip Temperature: 0x1c4 (40.9 degree C, 105.6 degree F)
FPGA Chip Voltage (1.8VS): 0x01e (1.76 Volts)
FPGA Chip Voltage (3.3VS): 0x038 (3.28 Volts)
FPGA Chip Voltage (VCC): 0x02e (0.90 Volts)
FPGA Chip Voltage (VCCP): 0x02e (0.90 Volts)
FPGA Chip Voltage (VCCPT): 0x02e (1.80 Volts)
FPGA Chip Voltage (VCCERAM): 0x02e (0.90 Volts)
FPGA Chip Voltage (VCCL_HPS): 0x000 (0.00 Volts)
FPGA Chip Voltage (ADCGND): 0x000 (0.00 Volts)
Run Level Sector Number: 0x15e (350)
Multi-Firmware Support: 0x1 (Yes)
MSI Support: Enabled
Scatter-Gather DMA Support: Yes
Number of MSG DMA Descriptors: 31
Double-Word Support: Yes
IRQ Level: 162
Maximum Link Width: 4
Negotiated Link Width: 4
Cloning Support: 3 (Cloning is supported. Region addressing allowed for any user)
Calibration Reference: 9.91 Volts
```

./crrtngfc\_info -l -v

```
Board 0
Version: 2023.6.0
Build: Wed May 17 09:19:29 EDT 2023
Module: crrtngfc
Board Index: 0 (PCIe-CCUR_FPGA_NGFC)
Bus: 0xb9
Slot: 0x00
Func: 0x00
Vendor ID: 0x1542
Sub-Vendor ID: 0x1542
Board Info: 0x93200101 (id=9320, type=0x01, func=0x01 (MultiFunc))
Member Code: 1 (048)
Sub-Device ID: 0x0100
Firmware Date/Time: 0x08252023 0x00120000 (08/25/2023 12:00:00)
Firmware Revision: 0x00010000 (1.0)
Fpgawb Revision: 0x00000000 (0000.00-00) (Not Supported)
Firmware Flavor Code: 0x00000000 (0) (****)
Number of Advanced IP Cores: 0x00000000 (0)
Board Serial Number: 0x000ac7c7 (706503)
FPGA Chip Temperature: 0x1c4 (40.9 degree C, 105.6 degree F)
FPGA Chip Voltage (1.8VS): 0x01e (1.76 Volts)
FPGA Chip Voltage (3.3VS): 0x038 (3.28 Volts)
FPGA Chip Voltage (VCC): 0x02e (0.90 Volts)
FPGA Chip Voltage (VCCP): 0x02e (0.90 Volts)
FPGA Chip Voltage (VCCPT): 0x02e (1.80 Volts)
FPGA Chip Voltage (VCCERAM): 0x02e (0.90 Volts)
FPGA Chip Voltage (VCCL_HPS): 0x000 (0.00 Volts)
FPGA Chip Voltage (ADCGND): 0x000 (0.00 Volts)
Run Level Sector Number: 0x15e (350)
Multi-Firmware Support: 0x1 (Yes)
MSI Support: Enabled
Scatter-Gather DMA Support: Yes
Number of MSG DMA Descriptors: 31
Double-Word Support: Yes
IRQ Level: 162
Maximum Link Width: 4
Negotiated Link Width: 4
Cloning Support: 3 (Cloning is supported. Region addressing allowed for any user)
Calibration Reference: 9.91 Volts

---ADC Information---
Maximum Voltage Range: 10 Volts
Number of ADCs: 3
Number of ADC Channels: 12
Number of ADC Resolution: 16 Bits
All ADC Channels Mask: 0x0000ffff
Maximum ADC Fifo Threshold: 0x00020000

---DAC Information---
Maximum Single-Ended Voltage Range: 10 Volts
Maximum Differential Voltage Range: 20 Volts
Number of DACs: 6
Number of DAC Channels: 12
Number of DAC Resolution: 16 Bits
All DAC Channels Mask: 0x0000ffff

---LDIO Module0 (DIO) Information---
Number of Channels: 32
Number of Ports: 32
Number of Channels/Port: 1
Number of Registers: 1
Number of Channels/Register: 32

---LDIO Module0 (DIO) COS Interrupt Information---
Module COS Count: 0
Module COS OVFL Count: 0
COS Status: 0x00000000
```

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.

```

COS Ovfl Status: 0x00000000

---LDIO Module1 (LIO) Information---
 Number of Channels: 32
 Number of Ports: 8
 Number of Channels/Port: 4
 Number of Registers: 1
Number of Channels/Register: 32

---LDIO Module1 (LIO) COS Interrupt Information---
 Module COS Count: 0
 Module COS OVFL Count: 0
 COS Status: 0x00000000
 COS Ovfl Status: 0x00000000

---Physical Memory Information---
 Num of Trans Tbl Entries: 16
Num of Physical Memory Entries: 512
 Avalon Page Bits: 20
 Avalon Page Size: 1048576
 TX Interface Base: 16777216

---Analog/MsgDMA Interrupt Information---
 Interrupt Count: 0
 MSG DMA %d Count: 0
 MSG DMA %d Count: 0
 MSG DMA %d Count: 0
 MSG DMA %d Count: 0
 MSG DMA %d Count: 0
 MSG DMA %d Count: 0
Interrupts Occurred Mask: 0x00000000
 Wakeup Interrupt Mask: 0x00000000
 Timeout Seconds: 0
Interrupts Occurred Mask: 0x00000000
 Wakeup Interrupt Mask: 0x00000000

---Memory Regions Information---
 Region 0: Addr=0xfbe80000 Size=32768 (0x8000)
 Region 2: Addr=0xfbe00000 Size=524288 (0x80000)

```

### 3.2.20 lib/ccrtngfc\_ldio\_intr

This test allows the user to check out the Digital I/O and LVDS I/O components of the cards using interrupts.

```

Usage: ./ccrtngfc_ldio_intr [-b Board] [-d Delay] [-F FallCh] [-l LoopCnt] [-L LevelCh]
 [-M SelectLDioModule_First] [-R RiseCh] [-X DeleteCh]
-b <board> (board #, default = 0)
-d Delay (Delay between screen refresh -- default is 100 milli-seconds)
-F FallCh (Falling Edge Channel_List)
-l LoopCnt (Loop count -- default is 0)
-L LevelCh (Level State Channel_List)
-M SelectLDioModule_First (Select LDIO Module -- default module is 'module 0')
-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 LDIO 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 of running LIO and DIO concurrently:

- 1) Connect 17 KHz square wave to DIO (M0) channel 0 with a voltage of 0 to +4 volts and load of High-Z
- 2) Connect 15 KHz square wave to LIO (M1) channel 3 with a voltage of +/- 200 milli-volts and load of 50 Ohms
- 3) shield -a 2,4-5 (shield processor 2, 4 & 5)



- 4) ./ccrtnngfc\_smp\_affinity -c4 (force driver to CPU 2)
- 5) run -b4-5 ./ccrtnngfc\_ldio\_intr -M0 -M1 (run DIO & LIO test on CPU 4 & 5)
- or --
- 6) run -b4-5 ./ccrtnngfc\_ldio\_intr -M0 -M1 -F0 -R3 (run DIO & LIO test on CPU 4 & 5, Rise Ch3, Fall Ch0)

Example display:

```
Connect 20KHz square wave to DIO channel 0 with a voltage of 0 to +4 volts and load of High-Z
shield -a 2, 4-5
./ccrtnngfc_smp_affinity -c4
run -b4-5 ./ccrtnngfc_ldio_intr -M0
```

```
Rising Edge[-R]: ### No Channels Selected ###
Falling Edge[-F]: ### No Channels Selected ###
Level State[-L]: Number of Channels = 32
 Module 0: 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
Disable COS[-X]: ### No Channels Selected ###
```

```
===== [DIO Module 0] =====
Scan Count: 417 (ddd:hh:mm:ss = 000:00:00:41)
COS Interrupt Handler: (Active)
COS Interrupt Duration (usec): 24.69 min=16.51 max=51.73 run_ave=24.96 (40056.86 Hz)
Driver Interrupt Response Time (usec): 12.24 min=11.03 max=18.99 run_ave=12.19
Driver Interrupts Occurred Mask: 0x00000300
Driver Wakeup Interrupt Mask: 0x00000100
Driver Spurious Interrupt Count: 0
Driver Repeat on New Interrupt Count: 0
Driver User Wakeup Count: 1673552
User Callback Count: 1673552
Missed User Callback Count: 0
```

```
[DIO Module 0]
COS Enable (Ch31..00): 0xffffffff
COS Mode (Ch31..00): 0x00000000
COS Edge Sense (Ch31..00): 0x00000000
COS Status (Ch31..00): 0x00000001
COS OVERFLOW Status (Ch31..00): 0x00000000
Module COS Counts: 1673554
Module COS Rate (Hz): 40000.19
Module Queued COS Counts: 2
Module Hardware Overflow COS Counts: 0
```

```
<----- [DIO Module 0] Channels COS Status Count ('+' Rise, '-' Fall, '.' Level, 'NC' No COS) ----->
 [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] 1673554. 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.
```

```
<----- [DIO Module 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.
```

```
Connect 20KHz square wave to LIO channel 3 with a voltage of +/- 200 milli-volts and load of 50 ohms
shield -a 2, 4-5
./ccrtnngfc_smp_affinity -c4
```

```
run -b4-5 ./crtngfc_ldio_intr -M1
```

```
Rising Edge[-R]: ### No Channels Selected ###
Falling Edge[-F]: ### No Channels Selected ###
Level State[-L]: Number of Channels = 32
Module 1: 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
Disable COS[-X]: ### No Channels Selected ###
```

```
===== [LIO Module 1] =====
Scan Count: 757 (ddd:hh:mm:ss = 000:00:01:16)
COS Interrupt Handler: (Active)
COS Interrupt Duration (usec): 24.63 min=15.17 max=58.51 run_ave=24.96 (40056.92 Hz)
Driver Interrupt Response Time (usec): 13.36 min=12.09 max=23.49 run_ave=13.48
Driver Interrupts Occurred Mask: 0x00000200
Driver Wakeup Interrupt Mask: 0x00000200
Driver Spurious Interrupt Count: 0
Driver Repeat on New Interrupt Count: 0
Driver User Wakeup Count: 3044519
User Callback Count: 3044519
Missed User Callback Count: 0
```

```
[LIO Module 1]
COS Enable (Ch31..00): 0xffffffff
COS Mode (Ch31..00): 0x00000000
COS Edge Sense (Ch31..00): 0x00000000
COS Status (Ch31..00): 0x00000008
COS OVERFLOW Status (Ch31..00): 0x00000000
Module COS Counts: 3044556
Module COS Rate (Hz): 39999.96
Module Queued COS Counts: 37
Module Hardware Overflow COS Counts: 0
```

```
<----- [LIO Module 1] Channels COS Status Count ('+' Rise, '-' Fall, '.' Level, 'NC' No COS) ----->
 [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] 0. 0. 0. 3044556. 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.
```

```
<----- [LIO Module 1] 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.
```

```
Connect 15KHz square wave to LIO channe 3 with a voltage of +/- 200 milli-volts and load of 50 ohms
Connect 17KHz square wave to DIO channel 0 with a voltage of 0 to +4 volts and load of High-Z
shield -a 2, 4-5
./crtngfc_smp_affinity -c4
run -b4-5 ./crtngfc_ldio_intr -M0 -M1
```

```
Rising Edge[-R]: ### No Channels Selected ###
Falling Edge[-F]: ### No Channels Selected ###
Level State[-L]: Number of Channels = 64
Module 0: 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
Module 1: 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
Disable COS[-X]: ### No Channels Selected ###
```

```
===== [DIO Module 0] [LIO Module 1] =====
Scan Count: 537 (ddd:hh:mm:ss = 000:00:00:54)
```

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.

```

COS Interrupt Handler: (Active)
COS Interrupt Duration (usec): 28.86 min=13.84 max=70.14 run_ave=26.40 (37873.55 Hz)
Driver Interrupt Response Time (usec): 19.24 min=10.78 max=25.28 run_ave=15.45
Driver Interrupts Occurred Mask: 0x00000300
Driver Wakeup Interrupt Mask: 0x00000300
Driver Spurious Interrupt Count: 0
Driver Repeat on New Interrupt Count: 868468
Driver User Wakeup Count: 2052781
User Callback Count: 2052781
Missed User Callback Count: 0

```

```

[DIO Module 0] [LIO Module 1]
COS Enable (Ch31..00): 0xffffffff 0xffffffff
COS Mode (Ch31..00): 0x00000000 0x00000000
COS Edge Sense (Ch31..00): 0x00000000 0x00000000
COS Status (Ch31..00): 0x00000000 0x00000008
COS OVERFLOW Status (Ch31..00): 0x00000000 0x00000000
Module COS Counts: 1845314 1625180
Module COS Rate (Hz): 34000.80 30000.68
Module Queued COS Counts: 115691 2
Module Hardware Overflow COS Counts: 0 0

```

```

<----- [DIO Module 0] Channels COS Status Count ('+' Rise, '-' Fall, '.' Level, 'NC' No COS) ----->
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] 1841872. 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.

```

```

<----- [DIO Module 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.

```

```

<----- [LIO Module 1] Channels COS Status Count ('+' Rise, '-' Fall, '.' Level, 'NC' No COS) ----->
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] 0. 0. 0. 1625180. 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.

```

```

<----- [LIO Module 1] 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.

```

```

Connect 15KHz square wave to LIO channe 3 with a voltage of +/- 200 milli-volts and load of 50 ohms
Connect 17KHz square wave to DIO channel 0 with a voltage of 0 to +4 volts and load of High-Z
shield -a 2, 4-5
./cctrngfc_smp_affinity -c4
run -b4-5 ./cctrngfc_ldio_intr -M0 -M1 -F0 -R3

```

```

Rising Edge[-R]: Number of Channels = 2
Module 0: 3
Module 1: 3
Falling Edge[-F]: Number of Channels = 2
Module 0: 0
Module 1: 0

```

```

Level State[-L]: Number of Channels = 60
Module 0: 1 2 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
Module 1: 1 2 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
Disable COS[-X]: ### No Channels Selected ###

```

```

===== [DIO Module 0] [LIO Module 1] =====
Scan Count: 8018 (ddd:hh:mm:ss = 000:00:13:24)
COS Interrupt Handler: (Active)
COS Interrupt Duration (usec): 63.01 min=17.02 max=89.75 run_ave=41.59 (24042.27 Hz)
Driver Interrupt Response Time (usec): 18.25 min=0.07 max=37.26 run_ave=15.32
Driver Interrupts Occurred Mask: 0x00000300
Driver Wakeup Interrupt Mask: 0x00000300
Driver Spurious Interrupt Count: 0
Driver Repeat on New Interrupt Count: 3218271
Driver User Wakeup Count: 19320260
User Callback Count: 19320260
Missed User Callback Count: 0

```

```

[DIO Module 0] [LIO Module 1]
COS Enable (Ch31..00): 0xffffffff 0xffffffff
COS Mode (Ch31..00): 0x00000009 0x00000009
COS Edge Sense (Ch31..00): 0x00000008 0x00000008
COS Status (Ch31..00): 0x00000001 0x00000008
COS OVERFLOW Status (Ch31..00): 0x00000000 0x00000000
Module COS Counts: 13673976 12065273
Module COS Rate (Hz): 17000.00 15000.00
Module Queued COS Counts: 803864 15
Module Hardware Overflow COS Counts: 0 0

```

```

<----- [DIO Module 0] Channels COS Status Count ('+' Rise, '-' Fall, '.' Level, 'NC' No COS) ----->
 [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] 13673976- 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.

```

```

<----- [DIO Module 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.

```

```

<----- [LIO Module 1] Channels COS Status Count ('+' Rise, '-' Fall, '.' Level, 'NC' No COS) ----->
 [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
=====
[0] 0- 0. 0. 12065273+ 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.

```

```

<----- [LIO Module 1] 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.

```

### 3.2.21 lib/ccrtngfc\_lio

This test validates the proper operation of the LVDS I/O. It can be run with various run options to test the component.

```

Usage: ./ccrtnngfc_lio [-b BoardNo] [-c] [-d Delay] [-F DebugFile] [-l LoopCnt]
 [-M SelectLDioModule] [-n NumChans] [-p PatternSelect]
 [-r RunOption] [-s SkipChannelsMask]
-b BoardNo (Board number -- default is 0)
-c (Switch to Continuous mode -- default is SYNC)
-d Delay (Delay between screen refresh -- default is 100)
-F DebugFile (Debug file -- default "=== None ===")
-l LoopCnt (Loop count -- default is 0)
-M SelectLDioModule (Select LDIO Module -- default is LIO module 1)
-n NumChans (number of channels -- default is 32)
-p PatternSelect (LDIO 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 Output Isolation Test)
 -rD (Fast [no curses] Digital Output Isolation Test)
 -re (External Loopback LIO test - Loopback breakout box required)
 -rE (Fast [no curses] external loopback LIO test - Loopback breakout
 box required)
 -ri (Read LIO input channels)
 -rL (Internal Loopback LIO test - No inputs/outputs must be connected)
 -rL (Fast [no curses] internal loopback LIO test - No inputs/outputs
 must be connected)
 -ro (Write pattern to LIO output channels)
-s SkipChannelsMask (Skip channels mask -- default is @0x00000000)
 -sXXXXXXXX (LDIO Channels 31..00=XXXXXXXX in Hex)
e.g. ./ccrtnngfc_lio -rL -sf0 (Internal Loopback Testing. Skip LDIO Channels 04-07)
 ./ccrtnngfc_lio -rE -sf0 (External Loopback w/o Curses Testing. Skip LIO
 Channels 04-07)

```

Example display:

```
./ccrtnngfc_lio
```

```
AvalonPtr=0x7ffff7f52000
```

```

Board Number [-b]: 0
Output Sync Mode [-c]: 1 [SYNC]
Delay [-d]: 100 milli-seconds
LDIO Module [-g]: 1
Loop Count [-l]: ***Forever***
Number of Channels [-n]: 32
Run Option [-r]: 0 (Digital Input)
Skip Channels Mask [-s]: 0x00000000 (31..00)

```

```

LIO Direction : 0x00 (All Input)
LDIO Enable : 0x00000001 (Enable)
Input Snapshot : 0x00000001 (Snapshot)
Scan Count : 239

```

```
Read Duration (microsecs) : 3.824 (min= 3.749/max= 13.590/ave= 3.905)
```

```

Channels Inputs
=====
31..00 [1]: FFFFFFFF

```

```
<----- Input Channels ----->
```

```

 [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
 === === === === === === === === === ===
[0] + + + + + + + + + +
[1] + + + + + + + + + +
[2] + + + + + + + + + +
[3] + +

```

./ccrtnngfc\_lio -rl

AvalonPtr=0x7ffff7f52000

```

Board Number [-b]: 0
Output Sync Mode [-c]: 1 [SYNC]
Delay [-d]: 100 milli-seconds
LDIO Module [-g]: 1
Loop Count [-l]: ***Forever***
Number of Channels [-n]: 32
Pattern Selection [-p]: 0 (Rolling Ones)
Run Option [-r]: 2 (Internal Loopback Test)
Skip Channels Mask [-s]: 0x00000000 (31..00)

```

```

Channel Mismatch Count : 0 (=== Passed ===)
LIO Direction : 0xFF (All Output)
LDIO Enable : 0x00000001 (Enable)
Input Snapshot : 0x00000001 (Snapshot)
Scan Count : 97

```

```

Write Duration (microsecs) : 1.863 (min= 1.833/max= 2.215/ave= 1.898)
Read Duration (microsecs) : 198.889 (min= 198.861/max= 216.188/ave= 200.349)

```

```

Channels Output Input Expected
===== ===== ===== =====
31..00 [1]: 00000001 00000001 00000001

```

```

<----- Input Channels ----->
 [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
 === === === === === === === === === ===
[0] +
[1]
[2]
[3] . .

```

### 3.2.22 lib/ccrtnngfc\_msgdma

This test performs a modular scatter-gather DMA test on boards that support it. Additionally, it displays performance information for each mode of operation.

```

Usage: ./ccrtnngfc_msgdma [-a AddrOff,ToAddrOff] [-b Board] [-C] [-d NumDesc] [-e MsgDmaEngine]
 [-f Input,Output] [-i] [-l LoopCnt] [-m Mode] [-s TotalXferSize] [-v]
 [-X] [-z]
-a <AddrOff,ToAddrOff> (First Avalon Address Offset, default DiagRam offset)
 (Second 'ToAddrOff' only for Avalon2Avalon mode)
-b <Board> (board #, default = 0)
-C (Perform Clone mode scatter-gather instead of single-shot)
-d <NumDesc> (Number of Descriptors, default = 1)

```

```

-e MsgDmaEngine (Select MsgDma Engine -- default "=== get free engine ===")
-f <Input>,<#Output> (Use input file as input data. default None)
 (Use Output file to write 'to' data. default None)
 (Prepend with '#' to remove comments and address)
-i (Use interrupts, default is poll)
-l <LoopCnt> (Loop Count, default = 1000)
-m <Mode> (Mode of Operation, default = all)
 'a2p' (Avalon memory address to Pci memory address)
 'p2a' (Pci memory address to Avalon memory address)
 'p2p' (Pci memory address to Pci memory address)
 'a2a' (Avalon memory address to Avalon memory address)
 'all' (All above modes with only memory addresses)

 'A2p' (Avalon FIFO address to Pci memory address - specify FIFO address '-a')
 'p2A' (Pci memory address to Avalon FIFO address - specify FIFO address '-a')
 'A2A' (Avalon FIFO address to Avalon FIFO address - specify FIFO address '-a')
-s <TotalXferSize> (Total Transfer Size in bytes, default size of DiagRam)
 (Maximum transfer size is 0x3FFFF)
-v (Verbose operation. default is quiet)
-X (Skip Data Validation, default is to validate)
-z (Zap messages, default is not to zap messages)

```

Notes:

- 1) For modes 'p2a' or 'a2p' only the first address 'AddrOff' is used in option '-a'
- 2) For modes 'a2a' the first address 'AddrOff' is "FROM" and second address 'ToAddrOff' is "TO"
- 3) If Input file is specified in the '-f' option, its contents is used to seed input
- 4) If '-X' option is specified, no pattern is written to input, unless '-f Input' option is specified
- 5) Multiple '-m' options can be specified on a single command line
- 6) When address '-a' option is not specified, DiagRam offset is used for Analog input/output
- 7) Normal running process if no arguments specified is as follows:
  - a) Incrementing pattern written to the input using programmed I/O and readback validated
  - b) Output written with 'baadbeef' pattern using programmed I/O
  - c) Scatter-Gather DMA performed from Input to Output
  - d) Data is read back from both Input and Output using programmed I/O and compared
- 8) An upper case 'A' in the -m option represents an Avalon FIFO address, while a lower case 'a' in the -m option represents a regular Avalon memory address
- 9) If a regular memory Avalon address is specified as an Avalon FIFO address and vice-versa results will be unpredictable
- 10) When either input or output Avalon address is pointing to a FIFO, then data validation is skipped
- 11) If a size is specified for a memory or FIFO address that is greater than it can handle, the result will be unpredictable. You will need to reset the firmware to restore proper operation

```

e.g. ./ccrtngfc_msgdma -mall (Run all transfer modes with validation)
./ccrtngfc_msgdma -a0x8000 -s0x100 (Run all modes with Avalon Address 0x8000 and size 0x100)
./ccrtngfc_msgdma -a0xA000 -s0x200 -ma2a (Run a2a with Avalon Address 0xA000 and size 0x200)
./ccrtngfc_msgdma -mp2a -l1 -d1 -fHexFile_16K -a0x10004 -X
 (Transfer Input file to Avalon memory at 0x10004)
./ccrtngfc_msgdma -ma2p -l1 -d1 -f,OutFile -s0x4000 -a0x10004 -X
 (Transfer Avalon memory at 0x10004 to output file 'OutFile')
./ccrtngfc_msgdma -mA2p -l10000 -s0x20000 -d16 -a0x18010
 (Transfer Avalon FIFO at 0x18010 to PCI memory with 16 descriptors
 where each descriptor has a transfer size of 0x2000 bytes.
 No validation will be performed)

```

Example display:

```
./ccrtngfc_msgdma
```

```
TotalXferSize = 0x00008000, individual descriptor length=0x008000
```

```
driver_lib_ptr: 0x7ffff7fde000
```

```
Eng0: 1000: P2P Total: Size 0x8000, Fire= 43.31us/ 756.56MB/s
```

```

(mi/ma/av: 730.25/ 889.30/ 819.46 MB/s, 36.85/ 44.87/ 39.99 us)
LastWord=0x007cffff
Eng0: 1000: A2A Total: Size 0x4000, Fire= 84.70us/ 193.42MB/s
(mi/ma/av: 183.64/ 194.42/ 193.21 MB/s, 84.27/ 89.22/ 84.80 us)
LastWord=0x003e7fff
Eng0: 1000: P2A Total: Size 0x8000, Fire= 86.95us/ 376.87MB/s
(mi/ma/av: 351.63/ 377.08/ 371.98 MB/s, 86.90/ 93.19/ 88.09 us)
LastWord=0x007cffff
Eng0: 1000: A2P Total: Size 0x8000, Fire= 86.50us/ 378.84MB/s
(mi/ma/av: 360.20/ 383.21/ 378.55 MB/s, 85.51/ 90.97/ 86.56 us)
LastWord=0x007cffff

```

./ccrtngfc\_msgdma -e0 -C (Cloning Option)

### Cloning Option Selected ###

### TotalXferSize = 0x00008000, individual descriptor length=0x008000 ###

driver\_lib\_ptr: 0x7ffff7fde000

```

Eng0: 1000: P2P Total: Size 0x8000, Fire= 35.44us/ 924.68MB/s
(mi/ma/av: 814.17/ 926.02/ 924.43 MB/s, 35.39/ 40.25/ 35.45 us)
LastWord=0x007cffff
Eng0: 1000: A2A Total: Size 0x4000, Fire= 90.39us/ 181.25MB/s
(mi/ma/av: 180.58/ 188.83/ 188.69 MB/s, 86.77/ 90.73/ 86.83 us)
LastWord=0x003e7fff
Eng0: 1000: P2A Total: Size 0x8000, Fire= 83.71us/ 391.43MB/s
(mi/ma/av: 369.81/ 391.53/ 391.31 MB/s, 83.69/ 88.61/ 83.74 us)
LastWord=0x007cffff
Eng0: 1000: A2P Total: Size 0x8000, Fire= 88.34us/ 370.92MB/s
(mi/ma/av: 370.87/ 374.50/ 374.28 MB/s, 87.50/ 88.35/ 87.55 us)
LastWord=0x007cffff

```

./ccrtngfc\_msgdma -e4

### TotalXferSize = 0x00008000, individual descriptor length=0x008000 ###

driver\_lib\_ptr: 0x7ffff7fde000

```

Eng4: 1000: P2P Total: Size 0x8000, Fire= 107.70us/ 304.25MB/s
(mi/ma/av: 290.85/ 306.83/ 304.46 MB/s, 106.80/ 112.66/ 107.63 us)
LastWord=0x007cffff
Eng4: 1000: A2A Total: Size 0x4000, Fire= 84.90us/ 192.97MB/s
(mi/ma/av: 183.67/ 194.40/ 192.75 MB/s, 84.28/ 89.20/ 85.00 us)
LastWord=0x003e7fff
Eng4: 1000: P2A Total: Size 0x8000, Fire= 107.35us/ 305.24MB/s
(mi/ma/av: 291.14/ 305.59/ 305.05 MB/s, 107.23/ 112.55/ 107.42 us)
LastWord=0x007cffff
Eng4: 1000: A2P Total: Size 0x8000, Fire= 86.36us/ 379.43MB/s
(mi/ma/av: 362.43/ 386.62/ 379.78 MB/s, 84.75/ 90.41/ 86.28 us)
LastWord=0x007cffff

```

### 3.2.23 lib/ccrtngfc\_msgdma\_clone

Cloning is an optional feature of this card that can be purchased separately. The basic cloning option is an extremely powerful tool that gives the user the ability to continuously transfer the contents of a region on the card to a physical memory entirely under hardware control, once cloning has commenced. This continuous transfer is performed at MsgDma speeds. A more advanced cloning option that can be purchased is known as Region Addressing. This option allows the board to Clone any MsgDma location to another MsgDma location, i.e. the source and destination locations can be any valid MsgDma able physical address in the system.

Only one Cloning or MsgDma operation can be active at a given time. Additionally, it is meaningless to perform Cloning on a FIFO region for two reasons. Firstly, each data in a FIFO is synchronous, however, the Cloned region



is accessed asynchronously. Secondly, when the FIFO runs empty (*underflow*) or cannot accept more data (*overflow*) the results are unpredictable.



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

This test show the capabilities of this new cloning option.

```
Usage: ./ccrtnngfc_msgdma_clone [-a FromAddr,ToAddr] [-b Board] [-d Delay] [-e MsgDmaEngine]
 [-F DebugFile] [-l LoopCnt] [-P] [-q] [-s XferSize]
 [-S DisplaySize] [-v Delay] [-X] [-Z]
 -a <FromAddr,ToAddr> (Clone address space From/To address in Hex)
 (If address less than board size, board offset used)
 -b <Board> (board #, default = 0)
 -d <Delay> (Delay between screen refresh -- default is 0 milli-seconds)
 -e MsgDmaEngine (Select MsgDma Engine -- default "=== get free engine ===")
 -F DebugFile (Debug file -- default "=== None ===")
 @DebugFile (Debug file and no display)
 @ (No Debug file and no display)
 -l LoopCnt (Loop count -- default is 0)
 -P (Program Board)
 -q (Quite (non-interactive) mode)
 -s <XferSize> (Transfer size in bytes)
 -S <DisplaySize> (Display size in bytes)
 -v Delay (Verify data. Add additional one cycle delay in micro-seconds if
 specified)
 -X (Disable region protection)
 -Z (Disable address cache - default is enabled)
```

e.g.

```
./ccrtnngfc_msgdma_clone (Clone DiagRam to physical memory created by
this program)
./ccrtnngfc_msgdma_clone -a,-1 (Clone DiagRam to physical memory created by
this program)
./ccrtnngfc_msgdma_clone -a-1,-1 -v (Clone physical memory to physical memory
created by this program and verify)
./ccrtnngfc_msgdma_clone -a,c000 -s0x1000 -v (Clone DiagRam at 0x8000 to board DiagRam at
0xc000)
./ccrtnngfc_msgdma_clone -a8000,c000 -s0x4000 -v (Clone DiagRam at 0x8000 to DiagRam at 0xC000
and verify)
./ccrtnngfc_msgdma_clone -a8000,-1 -s0x4000 -v (Clone DiagRam at 0x8000 to physical memory
created and verify)
./ccrtnngfc_msgdma_clone -a-1,8000 -s0x4000 -v (Clone physical memory created to DiagRam at
0x8000 and verify)
./ccrtnngfc_msgdma_clone -a,0xbd308000 -X (Clone DiagRam to some other board at
specified physical address)
```

#### Example display:

```
./ccrtnngfc_msgdma_clone -b0
```

```
Device Name : /dev/ccrtnngfc0
Board ID : 9320
Board Type : 01
Board Function : 01
Board Serial No : 706503 (0x000ac7c7)
Number of MsgDMA Descriptors: 31
```

Local Region (BAR2) Size : 0x00080000  
Local Region (BAR2) Address: 0xfbe00000  
Config Region (BAR0) Size : 0x00008000  
Config Region (BAR0) Address: 0xfbe80000

>>>##### Processing 'From' Address (0x0008000) #####<<<

From\_\_\_\_\_

|                             |                  |
|-----------------------------|------------------|
| TranslationRequired         | = 0              |
| UserSuppliedPhysicalAddress | = 0x0008000      |
| AvalonEquivalentAddress     | = 0x0008000      |
| PhysicalMemoryToAttach      | = 0xfbe0000      |
| PhysicalMemorySize          | = 0x0004000      |
| VirtualUserAddress          | = 0x7ffff7f5a000 |
| Flags                       | = 0x0000         |

>>>##### Processing 'To' Address (0xffffffffffffffc) #####<<<

Physical Memory Information:

|                   |                     |
|-------------------|---------------------|
| UserPID           | =209014             |
| PhysMemPtr        | =0x65e58000         |
| DriverVirtMemPtr  | =0xffff993ee5e58000 |
| MmappedUserMemPtr | =0x7ffff7fda000     |
| PhysMemSize       | =0x0004000          |
| PhysMemSizeFreed  | =0x00000000         |
| EntryInTxTbl      | =0                  |
| NumOfEntriesUsed  | =1                  |
| Flags             | =0x0000             |

To\_\_\_\_\_

|                             |                  |
|-----------------------------|------------------|
| TranslationRequired         | = 1              |
| UserSuppliedPhysicalAddress | = 0x65e58000     |
| AvalonEquivalentAddress     | = 0x01058000     |
| PhysicalMemoryToAttach      | = 0x65e58000     |
| PhysicalMemorySize          | = 0x0004000      |
| VirtualUserAddress          | = 0x7ffff7fda000 |
| Flags                       | = 0x0000         |

|                   |                                       |
|-------------------|---------------------------------------|
| Physical Address  | [-a]: 0xFBE08000/0x65E58000 (From/To) |
| Board Number      | [-b]: 0                               |
| Delay             | [-d]: 0 (milli-seconds)               |
| MSGDMA Engine     | [-e]: 0                               |
| Loop Count        | [-l]: 0 (forever)                     |
| Program Board     | [-P]: 0 (no)                          |
| Quiet Mode        | [-q]: 0 (interactive)                 |
| Transfer Size     | [-s]: 16384 (bytes)                   |
| Display Size      | [-S]: 256 (bytes)                     |
| Verify Data       | [-v]: no                              |
| Region Protection | [-X]: 1 (enabled)                     |
| Address Cache     | [-Z]: 0 (enabled)                     |
| One Cycle Time    | : 42.953 (micro-seconds)              |

| From_____                            | To_____                               |
|--------------------------------------|---------------------------------------|
| TranslationRequired = 0              | TranslationRequired = 1               |
| UserSuppliedPhysicalAddr = 0x0008000 | UserSuppliedPhysicalAddr = 0x65e58000 |
| AvalonEquivalentAddress = 0x0008000  | AvalonEquivalentAddress = 0x01058000  |
| PhysicalMemoryToAttach = 0xfbe0000   | PhysicalMemoryToAttach = 0x65e58000   |
| PhysicalMemorySize = 0x0004000       | PhysicalMemorySize = 0x0004000        |
| VirtualUserAddress = 0x7ffff7f5a000  | VirtualUserAddress = 0x7ffff7fda000   |
| Flags = 0x0000                       | Flags = 0x0000                        |

ScanCount=3361 (XferSize=16384, DisplaySize=256) (CopyTime: From 10638.176 usecs,  
To 3.454 usecs)

| From       | 00       | 04       | 08       | 0C       | 10       | 14       | 18       | 1C       |
|------------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0x00008000 | 007d0000 | 007d0001 | 007d0002 | 007d0003 | 007d0004 | 007d0005 | 007d0006 | 007d0007 |
| 0x00008020 | 007d0008 | 007d0009 | 007d000a | 007d000b | 007d000c | 007d000d | 007d000e | 007d000f |
| 0x00008040 | 007d0010 | 007d0011 | 007d0012 | 007d0013 | 007d0014 | 007d0015 | 007d0016 | 007d0017 |
| 0x00008060 | 007d0018 | 007d0019 | 007d001a | 007d001b | 007d001c | 007d001d | 007d001e | 007d001f |
| 0x00008080 | 007d0020 | 007d0021 | 007d0022 | 007d0023 | 007d0024 | 007d0025 | 007d0026 | 007d0027 |
| 0x000080a0 | 007d0028 | 007d0029 | 007d002a | 007d002b | 007d002c | 007d002d | 007d002e | 007d002f |
| 0x000080c0 | 007d0030 | 007d0031 | 007d0032 | 007d0033 | 007d0034 | 007d0035 | 007d0036 | 007d0037 |
| 0x000080e0 | 007d0038 | 007d0039 | 007d003a | 007d003b | 007d003c | 007d003d | 007d003e | 007d003f |

| To         | 00       | 04       | 08       | 0C       | 10       | 14       | 18       | 1C       |
|------------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0x65e58000 | 007d0000 | 007d0001 | 007d0002 | 007d0003 | 007d0004 | 007d0005 | 007d0006 | 007d0007 |
| 0x65e58020 | 007d0008 | 007d0009 | 007d000a | 007d000b | 007d000c | 007d000d | 007d000e | 007d000f |
| 0x65e58040 | 007d0010 | 007d0011 | 007d0012 | 007d0013 | 007d0014 | 007d0015 | 007d0016 | 007d0017 |
| 0x65e58060 | 007d0018 | 007d0019 | 007d001a | 007d001b | 007d001c | 007d001d | 007d001e | 007d001f |
| 0x65e58080 | 007d0020 | 007d0021 | 007d0022 | 007d0023 | 007d0024 | 007d0025 | 007d0026 | 007d0027 |
| 0x65e580a0 | 007d0028 | 007d0029 | 007d002a | 007d002b | 007d002c | 007d002d | 007d002e | 007d002f |
| 0x65e580c0 | 007d0030 | 007d0031 | 007d0032 | 007d0033 | 007d0034 | 007d0035 | 007d0036 | 007d0037 |
| 0x65e580e0 | 007d0038 | 007d0039 | 007d003a | 007d003b | 007d003c | 007d003d | 007d003e | 007d003f |

./ccrtngfc\_msgdma\_clone -e5

Device Name : /dev/ccrtngfc0  
Board ID : 9320  
Board Type : 01  
Board Function : 01  
Board Serial No : 706503 (0x000ac7c7)  
Number of MsgDMA Descriptors: 31

Local Region (BAR2) Size : 0x00080000  
Local Region (BAR2) Address: 0xfbe00000  
Config Region (BAR0) Size : 0x00080000  
Config Region (BAR0) Address: 0xfbe80000

>>>##### Processing 'From' Address (0x00008000) #####<<<

From \_\_\_\_\_  
TranslationRequired = 0  
UserSuppliedPhysicalAddress = 0x00008000  
AvalonEquivalentAddress = 0x00008000  
PhysicalMemoryToAttach = 0xfbe08000  
PhysicalMemorySize = 0x00004000  
VirtualUserAddress = 0x7ffff7f5a000  
Flags = 0x0000

>>>##### Processing 'To' Address (0xfffffffffffffc) #####<<<

Physical Memory Information:  
UserPID =209761  
PhysMemPtr =0x65e58000  
DriverVirtMemPtr =0xffff993ee5e58000  
MmappedUserMemPtr =0x7ffff7fda000  
PhysMemSize =0x00004000  
PhysMemSizeFreed =0x00000000  
EntryInTxTbl =0  
NumOfEntriesUsed =1  
Flags =0x0000

To \_\_\_\_\_

```

TranslationRequired = 1
UserSuppliedPhysicalAddress = 0x65e58000
AvalonEquivalentAddress = 0x01058000
PhysicalMemoryToAttach = 0x65e58000
PhysicalMemorySize = 0x00004000
VirtualUserAddress = 0x7ffff7fda000
Flags = 0x0000

```

```

Physical Address [-a]: 0xFBE08000/0x65E58000 (From/To)
Board Number [-b]: 0
Delay [-d]: 0 (milli-seconds)
MSGDMA Engine [-e]: 5
Loop Count [-l]: 0 (forever)
Program Board [-P]: 0 (no)
Quiet Mode [-q]: 0 (interactive)
Transfer Size [-s]: 16384 (bytes)
Display Size [-S]: 256 (bytes)
Verify Data [-v]: no
Region Protection [-X]: 1 (enabled)
Address Cache [-Z]: 0 (enabled)
One Cycle Time : 40.358 (micro-seconds)

```

| From                                  | To                                    |
|---------------------------------------|---------------------------------------|
| TranslationRequired = 0               | TranslationRequired = 1               |
| UserSuppliedPhysicalAddr = 0x00008000 | UserSuppliedPhysicalAddr = 0x65e58000 |
| AvalonEquivalentAddress = 0x00008000  | AvalonEquivalentAddress = 0x01058000  |
| PhysicalMemoryToAttach = 0xfbe08000   | PhysicalMemoryToAttach = 0x65e58000   |
| PhysicalMemorySize = 0x00004000       | PhysicalMemorySize = 0x00004000       |
| VirtualUserAddress = 0x7ffff7f5a000   | VirtualUserAddress = 0x7ffff7fda000   |
| Flags = 0x0000                        | Flags = 0x0000                        |

```

ScanCount=1171 (XferSize=16384, DisplaySize=256) (CopyTime: From 4692.005 usecs,
To 3.461 usecs)

```

| From       | 00       | 04       | 08       | 0C       | 10       | 14       | 18       | 1C       |
|------------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0x00008000 | 007d0000 | 007d0001 | 007d0002 | 007d0003 | 007d0004 | 007d0005 | 007d0006 | 007d0007 |
| 0x00008020 | 007d0008 | 007d0009 | 007d000a | 007d000b | 007d000c | 007d000d | 007d000e | 007d000f |
| 0x00008040 | 007d0010 | 007d0011 | 007d0012 | 007d0013 | 007d0014 | 007d0015 | 007d0016 | 007d0017 |
| 0x00008060 | 007d0018 | 007d0019 | 007d001a | 007d001b | 007d001c | 007d001d | 007d001e | 007d001f |
| 0x00008080 | 007d0020 | 007d0021 | 007d0022 | 007d0023 | 007d0024 | 007d0025 | 007d0026 | 007d0027 |
| 0x000080a0 | 007d0028 | 007d0029 | 007d002a | 007d002b | 007d002c | 007d002d | 007d002e | 007d002f |
| 0x000080c0 | 007d0030 | 007d0031 | 007d0032 | 007d0033 | 007d0034 | 007d0035 | 007d0036 | 007d0037 |
| 0x000080e0 | 007d0038 | 007d0039 | 007d003a | 007d003b | 007d003c | 007d003d | 007d003e | 007d003f |

| To         | 00       | 04       | 08       | 0C       | 10       | 14       | 18       | 1C       |
|------------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0x65e58000 | 007d0000 | 007d0001 | 007d0002 | 007d0003 | 007d0004 | 007d0005 | 007d0006 | 007d0007 |
| 0x65e58020 | 007d0008 | 007d0009 | 007d000a | 007d000b | 007d000c | 007d000d | 007d000e | 007d000f |
| 0x65e58040 | 007d0010 | 007d0011 | 007d0012 | 007d0013 | 007d0014 | 007d0015 | 007d0016 | 007d0017 |
| 0x65e58060 | 007d0018 | 007d0019 | 007d001a | 007d001b | 007d001c | 007d001d | 007d001e | 007d001f |
| 0x65e58080 | 007d0020 | 007d0021 | 007d0022 | 007d0023 | 007d0024 | 007d0025 | 007d0026 | 007d0027 |
| 0x65e580a0 | 007d0028 | 007d0029 | 007d002a | 007d002b | 007d002c | 007d002d | 007d002e | 007d002f |
| 0x65e580c0 | 007d0030 | 007d0031 | 007d0032 | 007d0033 | 007d0034 | 007d0035 | 007d0036 | 007d0037 |
| 0x65e580e0 | 007d0038 | 007d0039 | 007d003a | 007d003b | 007d003c | 007d003d | 007d003e | 007d003f |

```
./ccrtngfc_msgdma_clone -a8000,-1 -s0x4000 -v
```

```

Device Name : /dev/ccrtngfc0
Board ID : 9320
Board Type : 01
Board Function : 01
Board Serial No : 706503 (0x000ac7c7)

```

Number of MsgDMA Descriptors: 31

Local Region (BAR2) Size : 0x0008000  
Local Region (BAR2) Address: 0xfbe0000  
Config Region (BAR0) Size : 0x0008000  
Config Region (BAR0) Address: 0xfbe8000

>>##### Processing 'From' Address (0x0008000) #####<<<

From\_\_\_\_\_

|                             |                  |
|-----------------------------|------------------|
| TranslationRequired         | = 0              |
| UserSuppliedPhysicalAddress | = 0x0008000      |
| AvalonEquivalentAddress     | = 0x0008000      |
| PhysicalMemoryToAttach      | = 0xfbe0800      |
| PhysicalMemorySize          | = 0x0004000      |
| VirtualUserAddress          | = 0x7ffff7f5a000 |
| Flags                       | = 0x0000         |

>>##### Processing 'To' Address (0xffffffffffffffc) #####<<<

Physical Memory Information:

|                   |                     |
|-------------------|---------------------|
| UserPID           | =209765             |
| PhysMemPtr        | =0x65e5800          |
| DriverVirtMemPtr  | =0xffff993ee5e58000 |
| MmappedUserMemPtr | =0x7ffff7fda000     |
| PhysMemSize       | =0x0004000          |
| PhysMemSizeFreed  | =0x0000000          |
| EntryInTxTbl      | =0                  |
| NumOfEntriesUsed  | =1                  |
| Flags             | =0x0000             |

To\_\_\_\_\_

|                             |                  |
|-----------------------------|------------------|
| TranslationRequired         | = 1              |
| UserSuppliedPhysicalAddress | = 0x65e5800      |
| AvalonEquivalentAddress     | = 0x0105800      |
| PhysicalMemoryToAttach      | = 0x65e5800      |
| PhysicalMemorySize          | = 0x0004000      |
| VirtualUserAddress          | = 0x7ffff7fda000 |
| Flags                       | = 0x0000         |

Additional One Cycle Delay= 0.000 micro-seconds

|                   |                                                           |
|-------------------|-----------------------------------------------------------|
| Physical Address  | [-a]: 0xFBE08000/0x65E58000 (From/To)                     |
| Board Number      | [-b]: 0                                                   |
| Delay             | [-d]: 0 (milli-seconds)                                   |
| MSGDMA Engine     | [-e]: 0                                                   |
| Loop Count        | [-l]: 0 (forever)                                         |
| Program Board     | [-P]: 0 (no)                                              |
| Quiet Mode        | [-q]: 0 (interactive)                                     |
| Transfer Size     | [-s]: 16384 (bytes)                                       |
| Display Size      | [-S]: 256 (bytes)                                         |
| Verify Data       | [-v]: 0.000 (Additional One Cycle Delay in micro-seconds) |
| Region Protection | [-X]: 1 (enabled)                                         |
| Address Cache     | [-Z]: 0 (enabled)                                         |
| One Cycle Time    | : 42.865 (micro-seconds)                                  |

|                                      |                                      |
|--------------------------------------|--------------------------------------|
| From_____                            | To_____                              |
| TranslationRequired = 0              | TranslationRequired = 1              |
| UserSuppliedPhysicalAddr = 0x0008000 | UserSuppliedPhysicalAddr = 0x65e5800 |
| AvalonEquivalentAddress = 0x0008000  | AvalonEquivalentAddress = 0x0105800  |
| PhysicalMemoryToAttach = 0xfbe0800   | PhysicalMemoryToAttach = 0x65e5800   |
| PhysicalMemorySize = 0x0004000       | PhysicalMemorySize = 0x0004000       |

```

VirtualUserAddress = 0x7ffff7f5a000 VirtualUserAddress = 0x7ffff7fda000
Flags = 0x0000 Flags = 0x0000

FailCount=0 (==== passed ====)
ScanCount=1784 (XferSize=16384, DisplaySize=256) (WaitAfterPatternWrite: 42.925 usecs)

```

```

____From____ ____00____ ____04____ ____08____ ____0C____ ____10____ ____14____ ____18____ ____1C____
0x00008000 006f7000 006f7001 006f7002 006f7003 006f7004 006f7005 006f7006 006f7007
0x00008020 006f7008 006f7009 006f700a 006f700b 006f700c 006f700d 006f700e 006f700f
0x00008040 006f7010 006f7011 006f7012 006f7013 006f7014 006f7015 006f7016 006f7017
0x00008060 006f7018 006f7019 006f701a 006f701b 006f701c 006f701d 006f701e 006f701f
0x00008080 006f7020 006f7021 006f7022 006f7023 006f7024 006f7025 006f7026 006f7027
0x000080a0 006f7028 006f7029 006f702a 006f702b 006f702c 006f702d 006f702e 006f702f
0x000080c0 006f7030 006f7031 006f7032 006f7033 006f7034 006f7035 006f7036 006f7037
0x000080e0 006f7038 006f7039 006f703a 006f703b 006f703c 006f703d 006f703e 006f703f

```

```

____To____ ____00____ ____04____ ____08____ ____0C____ ____10____ ____14____ ____18____ ____1C____
0x65e58000 006f7000 006f7001 006f7002 006f7003 006f7004 006f7005 006f7006 006f7007
0x65e58020 006f7008 006f7009 006f700a 006f700b 006f700c 006f700d 006f700e 006f700f
0x65e58040 006f7010 006f7011 006f7012 006f7013 006f7014 006f7015 006f7016 006f7017
0x65e58060 006f7018 006f7019 006f701a 006f701b 006f701c 006f701d 006f701e 006f701f
0x65e58080 006f7020 006f7021 006f7022 006f7023 006f7024 006f7025 006f7026 006f7027
0x65e580a0 006f7028 006f7029 006f702a 006f702b 006f702c 006f702d 006f702e 006f702f
0x65e580c0 006f7030 006f7031 006f7032 006f7033 006f7034 006f7035 006f7036 006f7037
0x65e580e0 006f7038 006f7039 006f703a 006f703b 006f703c 006f703d 006f703e 006f703f

```

### 3.2.24 lib/ccrtngfc\_msgdma\_info

This test provides useful modular scatter-gather DMA information for cards that support it.

```

Usage: ./ccrtngfc_msgdma_info [-b Board] [-e MsgDmaEngine] [-l] [-s]
 -b <Board> (board #, default = 0)
 -e MsgDmaEngine (Select MsgDma Engine -- default -1)
 -l (long format)
 -s (Usage Status)

```

Note! You cannot use both '-s' and '-l' options at the same time

#### Example display:

```

./ccrtngfc_msgdma_info -s
MsgDma Engine 0: ### IN-USE ###
 Pid: 210565
 Tid: 140737353964416
MsgDma Engine 1: === FREE ===
MsgDma Engine 2: === FREE ===
MsgDma Engine 3: === FREE ===
MsgDma Engine 4: === FREE ===
MsgDma Engine 5: === FREE ===

```

```
./ccrtngfc_msgdma_info
```

```
driver_lib_ptr: 0x7ffff7fde000
```

```

===== MsgDma Engine 0 =====
MsgDmaOwnerPid = 0 === FREE ===
MsgDmaOwnerTid = 0 === FREE ===
MsgDmaExtDesOneCycleDelay (nanosecs) = 0
MsgDmaDescriptorBaseOffset = 0x00005000
MsgDmaTerminatingDescriptorOffset = 0x000057c0

```

```

Driver MsgDmaDispatcherCsrPtr = 0xfffffaeac5274000
Driver MsgDmaPrefetcherCsrPtr = 0xfffffaeac5274020
Driver MsgDmaExtendedDescriptorPtr = 0xfffffaeac5275000
Driver MsgDmaTerminatingDescriptorPtr = 0xfffffaeac52757c0

```

```

Library MsgDmaDispatcherCsrPtr = 0x7ffff7ff3000
Library MsgDmaPrefetcherCsrPtr = 0x7ffff7ff3020
Library MsgDmaExtendedDescriptorPtr = 0x7ffff7ff4000
Library MsgDmaTerminatingDescriptorPtr = 0x7ffff7ff47c0

```

===== Dispatcher (Addr: 4000) =====

```

Status = 0x0000000a
Control = 0x0000000c
ReadFillLevel = 0x00000000
WriteFillLevel = 0x00000000
ResponseFillLevel = 0x00000000
ReadSequenceNumber = 0x00000001
WriteSequenceNumber = 0x00000001

```

===== Prefetcher (Addr: 4020) =====

```

Status = 0x00000000
Control = 0x00000000
NextDescriptorPointer = 0xbaadbeef00005000 (### Descriptor ID 1 ###)
DescriptorPollingFrequency = 0x00000000

```

===== Descriptors =====

| ID | Addr    | ReadAddr | WritAddr | Length | Stat | Control  | SeqN | Rbct | Wbct | Rstr | Wstr | ActBytXfr | NextPtr               |
|----|---------|----------|----------|--------|------|----------|------|------|------|------|------|-----------|-----------------------|
| 1  | (5000): | 00008000 | 0000c000 | 004000 | 0000 | 80000000 | 0001 | 00   | 00   | 0000 | 0000 | 00000000  | 000057c0 (Terminator) |
| 2  | (5040): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 3  | (5080): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 4  | (50c0): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 5  | (5100): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 6  | (5140): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 7  | (5180): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 8  | (51c0): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 9  | (5200): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 10 | (5240): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 11 | (5280): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 12 | (52c0): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 13 | (5300): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 14 | (5340): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 15 | (5380): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 16 | (53c0): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 17 | (5400): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 18 | (5440): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 19 | (5480): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 20 | (54c0): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 21 | (5500): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 22 | (5540): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 23 | (5580): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 24 | (55c0): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 25 | (5600): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 26 | (5640): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 27 | (5680): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 28 | (56c0): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 29 | (5700): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 30 | (5740): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |
| 31 | (5780): | 00000000 | 00000000 | 000000 | 0000 | 00000000 | 0000 | 00   | 00   | 0000 | 0000 | 00000000  | 00000000              |

./crtngfc\_msgdma\_info -l

driver\_lib\_ptr: 0x7ffff7fde000

```

===== MsgDma Engine 0 =====
MsgDmaOwnerPid = 0 === FREE ===
MsgDmaOwnerTid = 0 === FREE ===
MsgDmaExtDesOneCycleDelay (nanosecs) = 0
MsgDmaDescriptorBaseOffset = 0x00005000
MsgDmaTerminatingDescriptorOffset = 0x000057c0

Driver MsgDmaDispatcherCsrPtr = 0xfffffaeac5274000
Driver MsgDmaPrefetcherCsrPtr = 0xfffffaeac5274020
Driver MsgDmaExtendedDescriptorPtr = 0xfffffaeac5275000
Driver MsgDmaTerminatingDescriptorPtr = 0xfffffaeac52757c0

Library MsgDmaDispatcherCsrPtr = 0x7ffff7ff3000
Library MsgDmaPrefetcherCsrPtr = 0x7ffff7ff3020
Library MsgDmaExtendedDescriptorPtr = 0x7ffff7ff4000
Library MsgDmaTerminatingDescriptorPtr = 0x7ffff7ff47c0

===== Dispatcher (Addr: 4000) =====
Status = 0x0000000a
Control = 0x0000000c
ReadFillLevel = 0x00000000
WriteFillLevel = 0x00000000
ResponseFillLevel = 0x00000000
ReadSequenceNumber = 0x00000001
WriteSequenceNumber = 0x00000001

===== Prefetcher (Addr: 4020) =====
Status = 0x00000000
Control = 0x00000000
NextDescriptorPointer = 0xbaadbeef00005000 (### Descriptor ID 1 ###)
DescriptorPollingFrequency = 0x00000000

===== Descriptor ID 1 (address: 0x5000) =====
ReadAddress = 0x00000000000008000
WriteAddress = 0x0000000000000c000
NextDescriptorPointer = 0x00000000000057c0 (### Terminator ###)
Status = 0x0000
Control = 0x80000000
Length = 0x00004000 (16384)
SequenceNumber = 0x0001 (1)
ReadBurstCount = 0x00 (0)
WriteBurstCount = 0x00 (0)
ReadStride = 0x0000 (0)
WriteStride = 0x0000 (0)
ActualBytesTransferred = 0x00000000 (0)

===== Descriptor ID 2 (address: 0x5040) =====
ReadAddress = 0x00000000000000000
WriteAddress = 0x00000000000000000
NextDescriptorPointer = 0x00000000000000000
Status = 0x0000
Control = 0x00000000
Length = 0x00000000 (0)
SequenceNumber = 0x0000 (0)
ReadBurstCount = 0x00 (0)
WriteBurstCount = 0x00 (0)
ReadStride = 0x0000 (0)
WriteStride = 0x0000 (0)
ActualBytesTransferred = 0x00000000 (0)

.
.

```



```

===== Descriptor ID 30 (address: 0x5740) =====
ReadAddress = 0x0000000000000000
WriteAddress = 0x0000000000000000
NextDescriptorPointer = 0x0000000000000000
Status = 0x0000
Control = 0x00000000
Length = 0x00000000 (0)
SequenceNumber = 0x0000 (0)
ReadBurstCount = 0x00 (0)
WriteBurstCount = 0x00 (0)
ReadStride = 0x0000 (0)
WriteStride = 0x0000 (0)
ActualBytesTransferred = 0x00000000 (0)

===== Descriptor ID 31 (address: 0x5780) =====
ReadAddress = 0x0000000000000000
WriteAddress = 0x0000000000000000
NextDescriptorPointer = 0x0000000000000000
Status = 0x0000
Control = 0x00000000
Length = 0x00000000 (0)
SequenceNumber = 0x0000 (0)
ReadBurstCount = 0x00 (0)
WriteBurstCount = 0x00 (0)
ReadStride = 0x0000 (0)
WriteStride = 0x0000 (0)
ActualBytesTransferred = 0x00000000 (0)

```

### 3.2.25 lib/ccrtngfc\_msgdma\_multi\_clone

This test is a more powerful version of the *ccrtngfc\_msgdma\_clone* test above. It allows the users to specify multiple source and destination addresses during the cloning operation. There is a limit on the number of physical memory that can be created or mapped. When that limit is reached, the tests fail to run. Additionally, there is a limit to the number of *MsgDm* descriptors that can be specified. Once again, if that limit is exceeded the test will fail to run.



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

```

Usage: ./ccrtngfc_msgdma_multi_clone [-a FromAddr,ToAddr,Size] [-b Board] [-d Delay]
 [-e MsgDmaEngine] [-F DebugFile] [-l LoopCnt]
 [-M SelectAdcModule] [-P] [-q] [-s XferSize]
 [-S DisplaySize] [-t|T FromAddr,Size,ClockFreq] [-X] [-Z]
-a <FromAddr,ToAddr,Size> (Clone address space From/To address (hex) and size (bytes))
 (If address less than board size, board offset used)
-b <Board> (board #, default = 0)
-d <Delay> (Delay between screen refresh -- default is 5 milli-seconds)
-e MsgDmaEngine (Select MsgDma Engine -- default "=== get free engine ===")
-F DebugFile (Debug file -- default "=== None ===")
 @DebugFile (Debug file and no curses display)
 @ (No Debug file and no curses display)
-l LoopCnt (Loop count -- default is 0)
-M SelectAdcModule (Select ADC Module -- default is module 0)
-P (Program Board)
-q (Quiet (non-interactive) mode)
-s <XferSize> (Transfer size in bytes)

```

```

-S <DisplaySize> (Display size in bytes)
-t <FromAddr,Size,ClockFreq> (Perform debug firmware testing - From address (hex), size
 (samples) and clock freq. - non-verbose)
-T <FromAddr,Size,ClockFreq> (Perform debug firmware testing - From address (hex), size
 (samples) and clock freq. - verbose)
-X (Disable region protection)
-Z (Disable address cache - default is enabled)

```

Note: If the size is not specified in the '-a' option, then the default size or that specified in the '-s' option is used

If debug firmware is installed, you can use the '-t|T' option

The debug firmware uses Clock 3 and its value is programmed by the 't|T' option

e.g.

```

./ccrtngfc_msgdma_multi_clone (Clone DiagRam to physical memory created
 by this program)
./ccrtngfc_msgdma_multi_clone -a,-1 (Clone DiagRam to physical memory created
 by this program)
./ccrtngfc_msgdma_multi_clone -a-1,-1 (Clone physical memory to physical memory
 created by this program)
./ccrtngfc_msgdma_multi_clone -a,c000 -s0x1000 (Clone DiagRam at 0x8000 to board DiagRam
 at 0xc000)
./ccrtngfc_msgdma_multi_clone -a8000,c000 -s0x4000 (Clone DiagRam at 0x8000 to DiagRam at
 0xc000)
./ccrtngfc_msgdma_multi_clone -a,0xbd308000 -X (Clone DiagRam to some other board at
 specified physical address)
./ccrtngfc_msgdma_multi_clone -a8000,8800 -a8800,9000 -a9000,9800 -a9800,a000 -aa000,a800
 -aa800,b000 -ab000,b800 -ab800,c000 -ac000,c800 -s256
 (Clone through all 9 descriptors)
./ccrtngfc_msgdma_multi_clone -a8000,8800 -a8800,9000 -a9000,9800 -a9800,a000,128 -aa000,a800
 -aa800,b000 -ab000,b800 -ab800,c000 -ac000,c800,288 -s256 -S288
 (Clone through all 9 descriptors with
 different sizes)
./ccrtngfc_msgdma_multi_clone -t (Perform non-verbose Debug Firmware
 testing using default address 0x6000
 and size 64 samples (256 bytes) with
 default Clock Frequency)
./ccrtngfc_msgdma_multi_clone -T,256 -F/tmp/LOG (Perform verbose Debug Firmware testing
 using default address, 256 bytes and
 send output to /tmp/LOG with default
 Clock Frequency)
./ccrtngfc_msgdma_multi_clone -T,,450000 -F/tmp/LOG (Perform verbose Debug Firmware testing
 using default address and size and send
 output to /tmp/LOG with 450000 SPS
 Clock)

```

#### Example display:

```
./ccrtngfc_msgdma_multi_clone
```

```

Device Name : /dev/ccrtngfc0
Board ID : 9320
Board Type : 01
Board Function : 01
Board Serial No : 706503 (0x000ac7c7)
Number of MsgDMA Descriptors: 31

```

```

Local Region (BAR2) Size : 0x00080000
Local Region (BAR2) Address: 0xfbe00000
Config Region (BAR0) Size : 0x00008000
Config Region (BAR0) Address: 0xfbe80000

```

>>>##### Processing 'From' Address (0x00008000) #####<<<

From\_\_\_\_\_
TranslationRequired = 0
UserSuppliedPhysicalAddress = 0x00008000
UserSuppliedSize = 0x00004000
AvalonEquivalentAddress = 0x00008000
PhysicalMemoryToAttach = 0xfbe08000
PhysicalMemorySize = 0x00004000
VirtualUserAddress = 0x7ffff7f5a000

>>>##### Creating Physical Memory of size (0x00004000) for 'To' Address #####<<<

Physical Memory Information:
UserPID =228731
PhysMemPtr =0x65e58000
DriverVirtMemPtr =0xffff993ee5e58000
MmappedUserMemPtr =0x7ffff7fda000
PhysMemSize =0x00004000
PhysMemSizeFreed =0x00000000
EntryInTxTbl =0
NumOfEntriesUsed =1
Flags =0x0000

To\_\_\_\_\_
TranslationRequired = 1
UserSuppliedPhysicalAddress = 0x65e58000
UserSuppliedSize = 0x00004000
AvalonEquivalentAddress = 0x01058000
PhysicalMemoryToAttach = 0x65e58000
PhysicalMemorySize = 0x00004000
VirtualUserAddress = 0x7ffff7fda000

Physical Address [-a]: 0xFBE08000/0x65E58000 (From/To)
Board Number [-b]: 0
Delay [-d]: 5 (milli-seconds)
MSGDMA Engine [-e]: 0
Loop Count [-l]: 0 (forever)
Program Board [-P]: 0 (no)
Quiet Mode [-q]: 0 (interactive)
Transfer Size [-s]: 16384 (bytes)
Display Size [-S]: 256 (bytes)
Region Protection [-X]: 1 (enabled)
Address Cache [-Z]: 0 (enabled)
One Cycle Time : 42.944 (micro-seconds)
Current Descriptor : 0

From\_\_\_\_\_ To\_\_\_\_\_
TranslationRequired = 0 TranslationRequired = 1
UserSuppliedPhysicalAddr = 0x00008000 UserSuppliedPhysicalAddr = 0x65e58000
UserSuppliedSize = 0x00004000 UserSuppliedSize = 0x00004000
AvalonEquivalentAddress = 0x00008000 AvalonEquivalentAddress = 0x01058000
PhysicalMemoryToAttach = 0xfbe08000 PhysicalMemoryToAttach = 0x65e58000
PhysicalMemorySize = 0x00004000 PhysicalMemorySize = 0x00004000
VirtualUserAddress = 0x7ffff7f5a000 VirtualUserAddress = 0x7ffff7fda000
Flags = 0x0000 Flags = 0x0000

ScanCount=534 (XferSize=16384, DisplaySize=256) (CopyTime: From 10639.312 usec,
To 3.478 usec)

\_\_\_\_From\_\_\_\_ \_00\_\_\_\_ \_04\_\_\_\_ \_08\_\_\_\_ \_0C\_\_\_\_ \_10\_\_\_\_ \_14\_\_\_\_ \_18\_\_\_\_ \_1C\_\_\_\_

```

0x00008000 00049000 00049001 00049002 00049003 00049004 00049005 00049006 00049007
0x00008020 00049008 00049009 0004900a 0004900b 0004900c 0004900d 0004900e 0004900f
0x00008040 00049010 00049011 00049012 00049013 00049014 00049015 00049016 00049017
0x00008060 00049018 00049019 0004901a 0004901b 0004901c 0004901d 0004901e 0004901f
0x00008080 00049020 00049021 00049022 00049023 00049024 00049025 00049026 00049027
0x000080a0 00049028 00049029 0004902a 0004902b 0004902c 0004902d 0004902e 0004902f
0x000080c0 00049030 00049031 00049032 00049033 00049034 00049035 00049036 00049037
0x000080e0 00049038 00049039 0004903a 0004903b 0004903c 0004903d 0004903e 0004903f

```

```

____ To _____ 00 _____ 04 _____ 08 _____ 0C _____ 10 _____ 14 _____ 18 _____ 1C _____
0x65e58000 00049000 00049001 00049002 00049003 00049004 00049005 00049006 00049007
0x65e58020 00049008 00049009 0004900a 0004900b 0004900c 0004900d 0004900e 0004900f
0x65e58040 00049010 00049011 00049012 00049013 00049014 00049015 00049016 00049017
0x65e58060 00049018 00049019 0004901a 0004901b 0004901c 0004901d 0004901e 0004901f
0x65e58080 00049020 00049021 00049022 00049023 00049024 00049025 00049026 00049027
0x65e580a0 00049028 00049029 0004902a 0004902b 0004902c 0004902d 0004902e 0004902f
0x65e580c0 00049030 00049031 00049032 00049033 00049034 00049035 00049036 00049037
0x65e580e0 00049038 00049039 0004903a 0004903b 0004903c 0004903d 0004903e 0004903f

```

```

Enter 'c|C' to clear the pattern
 'w|W' toggle pattern write (Pattern Write Disabled)
 'q|Q' ->

```

### 3.2.26 lib/ccrtnngfc\_sensors

This test displays the various Power Module sensors and also allows to clear the faults and reset peak values.

```

Usage: ./ccrtnngfc_sensors [-b Board] [-c Command] [-d Delay]
 -b <board> (board #, default = 0)
 -c Command (Commands [f,p])
 f (Clear Faults)
 p (Clear Peak Values)
 -d Delay (Delay between screen refresh -- default is 1000)

```

#### Example display:

```
./ccrtnngfc_sensor
```

```

Board Number (-b): 0
Delay (-d): 1000 milli-seconds

Board Serial Number: 706503
FPGA Chip Temperature: 40.9 degree C (105.6 degree F)
FPGA Chip Voltage (1.8VS): 1.76 Volts
FPGA Chip Voltage (3.3VS): 3.28 Volts
FPGA Chip Voltage (VCC): 0.90 Volts
FPGA Chip Voltage (VCCP): 0.90 Volts
FPGA Chip Voltage (VCCPT): 1.80 Volts
FPGA Chip Voltage (VCCERAM): 0.90 Volts
FPGA Chip Voltage (VCCL_HPS): 0.00 Volts
FPGA Chip Voltage (ADCGND): 0.00 Volts

Power Module Information PM Module 0 PM Module 1
===== =====
Input Supply Voltage: 11.70312 Volts 11.70312 Volts
Output Voltage: 0.89990 Volts 0.90015 Volts
Output Current: 1.60547 Amps 0.60352 Amps
Output Temperature: 36.31250 Degree C 36.00000 Degree C
DIE Temperature: 44.87500 Degree C 44.87500 Degree C
PWM Frequency: 352.00000 KHz 352.00000 KHz
Output Power: 1.44531 Watts 0.54395 Watts

```

Peak Output Current: 1.76367 Amps

0.73633 Amps

### 3.2.27 lib/ccrtngfc\_smp\_affinity

This test provides a useful mechanism to display or set the IRQ to specific set of CPUs. This is useful when we want to make sure that the driver interrupts are not being interfered with other CPU activity.

Usage: ./ccrtngfc\_smp\_affinity [-b Board] [-c CpuMask]  
-b Board (Board number -- default is 0)  
-c CpuMask (CPU mask in HEX -- default is none)

e.g. ./ccrtngfc\_smp\_affinity (display IRQ CPU mask for selected board)  
./ccrtngfc\_smp\_affinity -c2 (set IRQ CPU for cpu 1)  
./ccrtngfc\_smp\_affinity -c4 (set IRQ CPU for cpu 2)  
./ccrtngfc\_smp\_affinity -c0x8 (set IRQ CPU for cpu 3)  
./ccrtngfc\_smp\_affinity -cE2 (set IRQ CPU for cpu 1,5,6,7)

#### Example display:

```
./ccrtngfc_smp_affinity
(IRQ57) fc user f8 actual
```

```
./ccrtngfc_smp_affinity -b1 -c8
(IRQ57) 08 user 08 actual
```

### 3.2.28 lib/ccrtngfc\_transfer

This test performs various DMA and Programmed I/O transfers between the board components and the PCI memory.

Usage: ./ccrtngfc\_transfer [-b Board] [-c CaseNum] [-e MsgDmaEngine] [-i] [-l LoopCnt]  
[-M SelectAdcModule] [-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)  
-e MsgDmaEngine (Select MsgDma Engine -- default "=== get free engine ===")  
-i (Enable Interrupts -- default = Disable)  
-l LoopCnt (Loop Count -- default is 100)  
-M SelectAdcModule (Select ADC Module -- default is module 0)  
-s XferSize (Avalon Ram Xfer Size in bytes -- default is 32768)

#### Example display:

```
./ccrtngfc_transfer (for cards with modular scatter-gather DMA support)
```

```
local_ptr=0x7ffff7f55000
```

```
Size of Avalon RAM = 32768 (0x00008000)
```

```
Physical Memory Information:
```

```
UserPID =324135
PhysMemPtr =0x65e78000
DriverVirtMemPtr =0xffff993ee5e78000
MmappedUserMemPtr =0x7ffff7f4d000
PhysMemSize =0x00008000
PhysMemSizeFreed =0x00000000
EntryInTxTbl =0
NumOfEntriesUsed =1
Flags =0x0000
```

```

1: Memory -> Avalon RAM (MSGDMA) (Size=0x8000): 100 (89.74 us, 365.15 MBytes/Sec)
2: Memory -> Avalon RAM (PIO) (Size=0x8000): 100 (478.34 us, 68.50 MBytes/Sec)
3: Avalon RAM -> Memory (MSGDMA) (Size=0x8000): 100 (88.50 us, 370.25 MBytes/Sec)
4: Avalon RAM -> Memory (PIO) (Size=0x8000): 100 (6566.20 us, 4.99 MBytes/Sec)
5: Memory -> Avalon ADC Calibration (PIO) (Size=0x30): 10000 (0.07 us, 727.27 MBytes/Sec)
6: Avalon ADC Calibration -> Memory (PIO) (Size=0x30): 10000 (12.72 us, 3.77 MBytes/Sec)
**** Test Passed ****

```

### 3.2.29 lib/ccrtngfc\_tst\_lib

This is an interactive test that accesses the various supported API calls.

```

Usage: ./ccrtngfc_tst_lib [-b Board] [-M SelectModule]
-b Board (Board number -- default board is 0)
-M SelectModule (Select Module -- default is all available modules)

```

#### Example display:

```
./ccrtngfc_tst_lib
```

```
Device Name: /dev/ccrtngfc0
```

```
===== Module Information =====
Module Number 0: ADC Id= 1579321.901: SN#= 729514 (High Speed Analog Daughter Card)
=====
```

```
===== Module Information =====
Module Number 1: ADC Id= 1579321.901: SN#= 729512 (High Speed Analog Daughter Card)
=====
```

```
===== Module Information =====
Module Number 0: DAC Id= 1579321.901: SN#= 729514 (High Speed Analog Daughter Card)
=====
```

```
===== Module Information =====
Module Number 1: DAC Id= 1579321.901: SN#= 729512 (High Speed Analog Daughter Card)
=====
```

```

01 = Clear Driver Error 02 = Clear Library Error
03 = Display BOARD Registers 04 = Display CONFIG Registers
05 = Dump Physical Memory List 06 = Get All Boards Driver Information
07 = Get Board CSR 08 = Get Board Information
09 = Get Driver Error 10 = Get Driver Information
11 = Get Library Error 12 = Get Mapped Config Pointer
13 = Get Mapped Driver/Library Pointer 14 = Get Mapped Local Pointer
15 = Get Physical Memory 16 = Get Power Module Information
17 = Get Value 18 = Initialize Board
19 = MMap Physical Memory 20 = Munmap Physical Memory
21 = Reload Firmware 22 = Reset Board
23 = Set Board CSR 24 = Set Value
25 = ### ADC MENU ### 26 = ### CALIBRATION MENU ###
27 = ### CLOCK GENERATOR MENU ### 28 = ### DAC MENU ###
29 = ### INTERRUPT MENU ### 30 = ### IP CORE MENU ###
31 = ### LDIO MENU ###

```

```
Main Selection ('h'=display menu, 'q'=quit)->
```

```

Main Selection ('h'=display menu, 'q'=quit)-> 25
```

```

Command: ADC_menu()
01 = -- ADC Activate 02 = -- ADC Clear FIFO Interrupt Status
03 = -- ADC Close 04 = -- ADC Disable
05 = -- ADC Driver Read Operation 06 = -- ADC Get Calibration Bus CSR
07 = -- ADC Get CSR 08 = -- ADC Get Driver Read Mode

```

|                                           |                                                |
|-------------------------------------------|------------------------------------------------|
| 09 = -- ADC Get FIFO Channel Select       | 10 = -- ADC Get FIFO Information               |
| 11 = -- ADC Get FIFO Threshold            | 12 = -- ADC Get Information                    |
| 13 = -- ADC Get FIFO Interrupt Status     | 14 = -- ADC Get Interrupt Timeout Seconds      |
| 15 = -- ADC Get Power Up Status           | 16 = -- ADC Get Test Bus Control               |
| 17 = -- ADC Get Value                     | 18 = -- ADC Open                               |
| 19 = -- ADC Read Channels                 | 20 = -- ADC Reset (disable followed by enable) |
| 21 = -- ADC Reset FIFO                    | 22 = -- ADC Set Calibration Bus CSR            |
| 23 = -- ADC Set CSR                       | 24 = -- ADC Set Driver Read Mode               |
| 25 = -- ADC Set FIFO Channel Select       | 26 = -- ADC Set FIFO Threshold                 |
| 27 = -- ADC Set Interrupt Timeout Seconds | 28 = -- ADC Set Test Bus Control               |
| 29 = -- ADC Set Value                     |                                                |

ADC Selection ('h'=display menu, 'q'=quit)->

Main Selection ('h'=display menu, 'q'=quit)-> 26

Command: calibration\_menu()

|                                               |                                             |
|-----------------------------------------------|---------------------------------------------|
| 01 = -- ADC: Get Negative Calibration         | 02 = -- ADC: Get Offset Calibration         |
| 03 = -- ADC: Get Positive Calibration         | 04 = -- ADC: Perform Auto Calibration       |
| 05 = -- ADC: Perform External Negative Calib. | 06 = -- ADC: Perform External Offset Calib. |
| 07 = -- ADC: Perform External Positive Calib. | 08 = -- ADC: Perform Negative Calibration   |
| 09 = -- ADC: Perform Offset Calibration       | 10 = -- ADC: Perform Positive Calibration   |
| 11 = -- ADC: Read Channels Calibration        | 12 = -- ADC: Reset Calibration              |
| 13 = -- ADC: Write Channels Calibration       | 14 = -- DAC: Get Negative Calibration       |
| 15 = -- DAC: Get Offset Calibration           | 16 = -- DAC: Get Positive Calibration       |
| 17 = -- DAC: Perform Auto Calibration         | 18 = -- DAC: Perform Negative Calibration   |
| 19 = -- DAC: Perform Offset Calibration       | 20 = -- DAC: Perform Positive Calibration   |
| 21 = -- DAC: Read Channels Calibration        | 22 = -- DAC: Reset Calibration              |
| 23 = -- DAC: Write Channels Calibration       |                                             |

Calibration Selection ('h'=display menu, 'q'=quit)->

Main Selection ('h'=display menu, 'q'=quit)-> 27

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 Get PLL CSR                      |
| 17 = Clock Set Generator CSR                | 18 = Clock Set Generator Dividers           |
| 19 = Clock Set Generator Input Clock Enable | 20 = Clock Set Generator Input Clock Select |
| 21 = Clock Set Generator Output Config      | 22 = Clock Set Generator Output Format      |
| 23 = Clock Set Generator Output Mode        | 24 = Clock Set Generator Output Mux         |
| 25 = Clock Set Generator P-Divider Enable   | 26 = Clock Set Generator Value              |
| 27 = Clock Set Generator Voltage Select     | 28 = Clock Set Generator Zero Delay         |
| 29 = Clock Set PLL CSR                      | 30 = Compute All Output Clocks              |
| 31 = Program All Output Clocks              | 32 = Read Clock Registers                   |
| 33 = Reset Clock (Hardware)                 | 34 = Soft Reset                             |
| 35 = Update Clock Generator Divider         | 36 = Write Clock Registers                  |

Clock Generator Selection ('h'=display menu, 'q'=quit)->

Main Selection ('h'=display menu, 'q'=quit)-> 28

Command: DAC\_menu()

|                                       |                                           |
|---------------------------------------|-------------------------------------------|
| 01 = -- DAC Activate                  | 02 = -- DAC Clear FIFO Interrupt Status   |
| 03 = -- DAC Close                     | 04 = -- DAC Disable                       |
| 05 = -- DAC Driver Write Operation    | 06 = -- DAC Get CSR                       |
| 07 = -- DAC Get Driver Write Mode     | 08 = -- DAC Get FIFO Channel Select       |
| 09 = -- DAC Get FIFO Information      | 10 = -- DAC Get FIFO Threshold            |
| 11 = -- DAC Get FIFO Write Count      | 12 = -- DAC Get Information               |
| 13 = -- DAC Get FIFO Interrupt Status | 14 = -- DAC Get Interrupt Timeout Seconds |

|                                           |                                                |
|-------------------------------------------|------------------------------------------------|
| 15 = -- DAC Get Power Up Status           | 16 = -- DAC Get Speed Mode                     |
| 17 = -- DAC Get Update Source Select      | 18 = -- DAC Get Value                          |
| 19 = -- DAC Open                          | 20 = -- DAC ReadBack Channels                  |
| 21 = -- DAC Read Channels                 | 22 = -- DAC Reset (disable followed by enable) |
| 23 = -- DAC Reset FIFO                    | 24 = -- DAC Set CSR                            |
| 25 = -- DAC Set Driver Write Mode         | 26 = -- DAC Set FIFO Channel Select            |
| 27 = -- DAC Set FIFO Threshold            | 28 = -- DAC Set FIFO Write Count               |
| 29 = -- DAC Set Interrupt Timeout Seconds | 30 = -- DAC Set Speed Mode                     |
| 31 = -- DAC Set Update Source Select      | 32 = -- DAC Set Value                          |
| 33 = -- DAC Write Channels                |                                                |

DAC Selection ('h'=display menu, 'q'=quit)->

---

Main Selection ('h'=display menu, 'q'=quit)-> 29

Command: interrupt\_menu()

|                             |                             |
|-----------------------------|-----------------------------|
| 01 = Add Irq                | 02 = Clear Interrupt Status |
| 03 = Disable Pci Interrupts | 04 = Enable Pci Interrupts  |
| 05 = Get Interrupt Status   | 06 = Get Interrupt Timeout  |
| 07 = Remove Irq             | 08 = Set Interrupt Timeout  |

Interrupt Selection ('h'=display menu, 'q'=quit)->

---

Main Selection ('h'=display menu, 'q'=quit)-> 30

Command: IPCORE\_menu()

|                                |                                   |
|--------------------------------|-----------------------------------|
| 01 = IpCore Get Ip Information | 02 = IpCore Get Ip Mapped Pointer |
|--------------------------------|-----------------------------------|

IP Core Selection ('h'=display menu, 'q'=quit)->

---

Main Selection ('h'=display menu, 'q'=quit)-> 31

Command: LDIO\_menu()

|                                              |                                          |
|----------------------------------------------|------------------------------------------|
| 01 = LDIO Get Channels Terminator            | 02 = LDIO Get Ports Direction            |
| 03 = LDIO Set Channels Terminator            | 04 = LDIO Set Channels Terminator to Off |
| 05 = LDIO Set Channels Terminator to On      | 06 = LDIO Set Ports Direction            |
| 07 = LDIO Set Ports Direction to Inputs      | 08 = LDIO Set Ports Direction to Outputs |
| 09 = LDIO Activate                           | 10 = LDIO Disable                        |
| 11 = LDIO Get Channels Polarity              | 12 = LDIO Get COS Channels Edge Sense    |
| 13 = LDIO Get COS Channels Enable            | 14 = LDIO Get COS Channels Mode          |
| 15 = LDIO Get COS Channels Overflow          | 16 = LDIO Get COS Channels Status        |
| 17 = LDIO Get Input Channels Filter          | 18 = LDIO Get Input Snapshot             |
| 19 = LDIO Get Output Sync                    | 20 = LDIO Information                    |
| 21 = LDIO Read Input Channels                | 22 = LDIO Read Output Channels           |
| 23 = LDIO Reset (disable followed by enable) | 24 = LDIO Set Channels Polarity          |
| 25 = LDIO Set COS Channels Edge Sense        | 26 = LDIO Set COS Channels Enable        |
| 27 = LDIO Set COS Channels Mode              | 28 = LDIO Set Input Channels Filter      |
| 29 = LDIO Set Input Snapshot                 | 30 = LDIO Set Output Sync                |
| 31 = LDIO Write Output Channels              | 32 = LDIO Write Output Channels High     |
| 33 = LDIO Write Output Channels Low          | 34 = LIO Get Ports Direction             |
| 35 = LIO Module Off                          | 36 = LIO Module On                       |
| 37 = LIO Set Ports Direction                 | 38 = LIO Set Ports Direction to Inputs   |
| 39 = LIO Set Ports Direction to Outputs      |                                          |

LDIO Selection ('h'=display menu, 'q'=quit)->

---



*This page intentionally left blank*