DAC Software Interface
CCRTNGFC (WC-CP-FIO2)

PCIe Next Generation FPGA I/O Card (NGFC)

Driver	ccrtngfc (WC-CP-FIO2)	
OS	RedHawk (CentOS/Rocky or Ubuntu based)	
Vendor	Concurrent Real-Time	
Hardware	PCIe Programmable Multi-Function Card (CP-FPGA-4 & 5)	
Author	Darius Dubash	
Date	November 13th, 2023	Rev 2023.1

This page intentionally left blank

Table of Contents

1. INTRODUCTION 6

1.1 Related Documents 6

2. SOFTWARE SUPPORT 6

2.1 Application Program Interface (API) Access 6

2.1.1 ccrtNGFC_DC_DAC_Activate() 7
2.1.2 ccrtNGFC_DC_DAC_Clear_Interrupt_Status() 7
2.1.3 ccrtNGFC_DC_DAC_Close() 8
2.1.4 ccrtNGFC_DC_DAC_Get_CSR() 8
2.1.5 ccrtNGFC_DC_DAC_Get_Driver_Write_Mode() 9
2.1.6 ccrtNGFC_DC_DAC_Get_Fifo_Channel_Select() 9
2.1.7 ccrtNGFC_DC_DAC_Get_Fifo_Info() 10
2.1.8 ccrtNGFC_DC_DAC_Get_Fifo_Threshold() 10
2.1.9 ccrtNGFC_DC_DAC_Get_Fifo_Write_Count() 11
2.1.10 ccrtNGFC_DC_DAC_Get_Info() 11
2.1.11 ccrtNGFC_DC_DAC_Get_Interrupt_Status() 12
2.1.12 ccrtNGFC_DC_DAC_Get_Interrupt_Timeout_Seconds() 12
2.1.13 ccrtNGFC_DC_DAC_Get_Negative_Cal() 12
2.1.14 ccrtNGFC_DC_DAC_Get_Offset_Cal() 13
2.1.15 ccrtNGFC_DC_DAC_Get_Positive_Cal() 14
2.1.16 ccrtNGFC_DC_DAC_Get_Power_Up_Status() 14
2.1.17 ccrtNGFC_DC_DAC_Get_Speed_Mode() 15
2.1.18 ccrtNGFC_DC_DAC_Get_Update_Source_Select() 16
2.1.19 ccrtNGFC_DC_DAC_Get_Value() 16
2.1.20 ccrtNGFC_DC_DAC_MsgDma_Configure_Channel() 17
2.1.21 ccrtNGFC_DC_DAC_MsgDma_Configure_Fifo() 18
2.1.22 ccrtNGFC_DC_DAC_MsgDma_Fire_Channel() 19
2.1.23 ccrtNGFC_DC_DAC_MsgDma_Fire_Fifo() 19
2.1.24 ccrtNGFC_DC_DAC_Open() 20
2.1.25 ccrtNGFC_DC_DAC_Perform_Auto_Calibration() 21
2.1.26 ccrtNGFC_DC_DAC_Perform_Negative_Calibration() 21
2.1.27 ccrtNGFC_DC_DAC_Perform_Offset_Calibration() 22
2.1.28 ccrtNGFC_DC_DAC_Perform_Positive_Calibration() 23
2.1.29 ccrtNGFC_DC_DAC_Presence() 23
2.1.30 ccrtNGFC_DC_DAC_ReadBack_Channels() 24
2.1.31 ccrtNGFC_DC_DAC_Read_Channels() 25
2.1.32 ccrtNGFC_DC_DAC_Read_Channels_Calibration() 26
2.1.33 ccrtNGFC_DC_DAC_Reset_Calibration() 26
2.1.34 ccrtNGFC_DC_DAC_Reset_Fifo() 27
2.1.35 ccrtNGFC_DC_DAC_Set_CSR() 27
2.1.36 ccrtNGFC_DC_DAC_Set_Driver_Write_Mode() 28
2.1.37 ccrtNGFC_DC_DAC_Set_Fifo_Channel_Select() 28
2.1.38 ccrtNGFC_DC_DAC_Set_Fifo_Threshold() 29
2.1.39 ccrtNGFC_DC_DAC_Set_Fifo_Write_Count() 30
2.1.40 ccrtNGFC_DC_DAC_Set_Interrupt_Timeout_Seconds() 30
2.1.41 ccrtNGFC_DC_DAC_Set_Negative_Cal() 31
2.1.42 ccrtNGFC_DC_DAC_Set_Offset_Cal() 31
2.1.43 ccrtNGFC_DC_DAC_Set_Positive_Cal() 32
2.1.44 ccrtNGFC_DC_DAC_Set_Speed_Mode() 33
2.1.45 ccrtNGFC_DC_DAC_Set_Update_Source_Select() 33
2.1.46 ccrtNGFC_DC_DAC_Set_Value() 34

2.1.47 ccrtNGFC_DC_DAC_Wait_For_Channel_Idle() 34
2.1.48 ccrtNGFC_DC_DAC_Wait_For_Fifo_To_Drain() 35
2.1.49 ccrtNGFC_DC_DAC_Write() 35
2.1.50 ccrtNGFC_DC_DAC_Write_Channels() 36
2.1.51 ccrtNGFC_DC_DAC_Write_Channels_Calibration() 37

This page intentionally left blank

1. Introduction

This document provides the software interface to the ***ccrtngfc*** driver DAC Daughter Card which communicates with the Concurrent Real-Time PCI Express Next Generation FPGA I/O Card (NGFC). This DAC daughter card is an optional interface that needs to be physically installed in one of two FMC slots located on the mother board. For additional information for low-level programming is contained in the *Concurrent Real-Time PCIe Next Generation FPGA I/O Cards (NGFC) Design Specification* (No. *0610111*) document.

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

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

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

1.1 Related Documents

- PCIe Next Generaton 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.

2. Software Support

2.1 Application Program Interface (API) Access

Before any of the DAC Daughter Card APIs can be invoked, the user first needs to issue the *ccrtNFGC_DC_DAC_Open()* call and supply the board handle that was returned with the *ccrtNGFC_Open()* call. Once the DAC daughter card is successfully opened, the call will return a DAC Handle which will be supplied as the first argument to all the remaining DAC APIs.

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 section describes the calls that are available to access this daughter card.

2.1.1 ccrtNGFC_DC_DAC_Activate()

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

```
/*************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Activate (void                       *DacHandle,
                             _ccrtngfc_dac_all_enable_t  activate,
                             _ccrtngfc_dac_all_enable_t  *current_state)

   Description: Activate/DeActivate Daughter Card DAC module

   Input:   void                                 *DacHandle      (DAC handle pointer)
            _ccrtngfc_dac_all_enable_t           activate        (activate/deactivate)
               # CCRTNGFC_DAC_ALL_DISABLE
               # CCRTNGFC_DAC_ALL_ENABLE
               # CCRTNGFC_DAC_ALL_RESET          (disable followed by enable)
               # CCRTNGFC_DAC_ALL_ENABLE_DO_NOT_CHANGE
   Output:  _ccrtngfc_dac_all_enable_t           *current_state  (active/deactive)
               # CCRTNGFC_DAC_ALL_DISABLE
               # CCRTNGFC_DAC_ALL_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)
 *************************************************************************/
```

2.1.2 ccrtNGFC_DC_DAC_Clear_Interrupt_Status()

This call clears the interrupt status for the DAC

```
/*************************************************************************
  _ccrtngfc_lib_error_number_t
  ccrtNGFC_DC_DAC_Clear_Interrupt_Status (void                    *DacHandle,
                                          _ccrtngfc_intsta_dac_t  dac_fifo_intr)

   Description: Clear Daughter Card DAC Interrupt Status

   Input:   void                    *DacHandle    (DAC handle pointer)
            _ccrtngfc_intsta_dac_t  dac_fifo_intr (pointer to DAC FIFO interrupt status)
                # CCRTNGFC_INT_DAC_FIFO_THRESHOLD_NONE
                # CCRTNGFC_INT_DAC_FIFO_THRESHOLD_RESET
                # CCRTNGFC_INT_DAC_FIFO_THRESHOLD_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)
 *************************************************************************/
```

2.1.3 ccrtNGFC_DC_DAC_Close()

This call is to be used when you no longer want to access the DAC daughter card that was opened with the *ccrtNGFC_DC_DAC_Open()* call.

```
/*****************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Close (void *DacHandle)

   Description: Close a Daughter Card DAC module.

   Input:   void                          *DacHandle  (DAC handle pointer)
   Output:  None
   Return:  _ccrtngfc_lib_error_number_t
               # CCRTNGFC_LIB_NO_ERROR               (successful)
               # CCRTNGFC_LIB_INVALID_ARG            (invalid argument)
               # CCRTNGFC_LIB_INVALID_MODULE         (invalid module)
 *****************************************************************************/
```

2.1.4 ccrtNGFC_DC_DAC_Get_CSR()

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

```
/*****************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_CSR (void                  *DacHandle,
                            _ccrtngfc_dac_mask_t  dac_mask,
                            _ccrtngfc_dac_csr_t   *dac_csr)

   Description: Get Daughter Card DAC Control and Status information

   Input:   void                                     *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_mask_t                     dac_mask    (selected DAC mask)
               # CCRTNGFC_DAC_MASK_0_1
               # CCRTNGFC_DAC_MASK_2_3
               # CCRTNGFC_DAC_MASK_4_5
               # CCRTNGFC_DAC_MASK_6_7
               # CCRTNGFC_DAC_MASK_8_9
               # CCRTNGFC_DAC_MASK_10_11
   Output:  _ccrtngfc_dac_csr_t                      *dac_csr    (pointer to DAC csr)
               _ccrtngfc_daccsr_busy_t               dac_interface_busy
                  # CCRTNGFC_DAC_IDLE
                  # CCRTNGFC_DAC_BUSY
               _ccrtngfc_daccsr_updmode_t            dac_update_mode
                  # CCRTNGFC_DAC_MODE_IMMEDIATE
                  # CCRTNGFC_DAC_MODE_SYNCHRONIZED
               _ccrtngfc_daccsr_dataformat_t         dac_data_format
                  # CCRTNGFC_DAC_OFFSET_BINARY
                  # CCRTNGFC_DAC_TWOS_COMPLEMENT
               _ccrtngfc_daccsr_output_select_t      dac_output_select
                  # CCRTNGFC_DAC_SINGLE_ENDED
                  # CCRTNGFC_DAC_DIFFERENTIAL
   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_DAC_IS_NOT_ACTIVE      (DAC is not active)
 *****************************************************************************/
```

2.1.5 ccrtNGFC_DC_DAC_Get_Driver_Write_Mode()

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

```
/*************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Driver_Write_Mode (void                              *DacHandle,
                                          _ccrtngfc_driver_DAC_write_mode_t *mode)

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

   Input:   void                                *DacHandle  (DAC handle pointer)
   Output:  _ccrtngfc_driver_DAC_write_mode_t   *mode       (selected DAC write mode)
               # CCRTNGFC_DAC_PIO_CHANNEL
               # CCRTNGFC_DAC_PIO_FIFO
   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_IOCTL_FAILED            (driver ioctl call failed)
               # CCRTNGFC_LIB_INVALID_ARG             (invalid argument)
 *************************************************************************/
```

2.1.6 ccrtNGFC_DC_DAC_Get_Fifo_Channel_Select()

This call returns the current Fifo Channel selection mask. Only samples for these selected channels should be placed in the FIFO.

```
/*************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Fifo_Channel_Select (void                         *DacHandle,
                                            _ccrtngfc_dac_channel_mask_t
                                                          *dac_fifo_channel_select_mask)

   Description: Get Daughter Card DAC Fifo Channel Selection

   Input:   void                         *DacHandle                    (DAC handle pointer)
   Output:  _ccrtngfc_dac_channel_mask_t *dac_fifo_channel_select_mask  (channel select mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_MASK
   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_DAC_IS_NOT_ACTIVE (DAC is not active)
```

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

2.1.7 ccrtNGFC_DC_DAC_Get_Fifo_Info()

This call returns DAC FIFO information to the user.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Fifo_Info (void                     *DacHandle,
                                  ccrtngfc_dac_fifo_info_t *dac_fifo)

   Description: Get Daughter Card DAC Fifo control and Status information

   Input:   void                                *DacHandle   (DAC handle pointer)
   Output:  ccrtngfc_dac_fifo_info_t            *dac_fifo    (pointer to DAC fifo struct)
               _ccrtngfc_dac_fifo_reset_t       reset;
                  # CCRTNGFC_DAC_FIFO_ACTIVE
                  # CCRTNGFC_DAC_FIFO_RESET
               _ccrtngfc_dac_fifo_overflow_t    overflow;
                  # CCRTNGFC_DAC_FIFO_NO_OVERFLOW
                  # CCRTNGFC_DAC_FIFO_OVERFLOW
               _ccrtngfc_dac_fifo_underflow_t   underflow;
                  # CCRTNGFC_DAC_FIFO_NO_UNDERFLOW
                  # CCRTNGFC_DAC_FIFO_UNDERFLOW
               _ccrtngfc_dac_fifo_full_t        full;
                  # CCRTNGFC_DAC_FIFO_NOT_FULL
                  # CCRTNGFC_DAC_FIFO_FULL
               _ccrtngfc_dac_fifo_threshold_t   threshold_exceeded;
                  # CCRTNGFC_DAC_FIFO_THRESHOLD_NOT_EXCEEDED
                  # CCRTNGFC_DAC_FIFO_THRESHOLD_EXCEEDED
               _ccrtngfc_dac_fifo_empty_t       empty;
                  # CCRTNGFC_DAC_FIFO_NOT_EMPTY
                  # CCRTNGFC_DAC_FIFO_EMPTY
               _ccrtngfc_dac_high_speed_mode_t high_speed_mode;
                  # CCRTNGFC_DAC_NORMAL_SPEED_MODE
                  # CCRTNGFC_DAC_HIGH_SPEED_MODE
               uint                             data_counter;
               uint                             threshold;
               uint                             max_threshold;
               uint                             driver_threshold;
               uint                             write_count;
               dac_fifo_channel_select_mask     channel_select_mask;
   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_DAC_IS_NOT_ACTIVE (DAC is not active)
***************************************************************************/
```

2.1.8 ccrtNGFC_DC_DAC_Get_Fifo_Threshold()

This call returns the DAC Fifo threshold information.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Fifo_Threshold (void *DacHandle,
                                       uint *dac_threshold)

   Description: Get Daughter Card DAC Fifo Threshold
```

```
   Input:   void                    *DacHandle      (DAC handle pointer)
   Output:  uint                    *dac_threshold  (DAC fifo threshold)
   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_DAC_IS_NOT_ACTIVE     (DAC is not active)
 ***************************************************************************/
```

2.1.9 ccrtNGFC_DC_DAC_Get_Fifo_Write_Count()

This call returns the count of number of samples in the DAC FIFO.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Fifo_Write_Count (void *DacHandle,
                                         uint *dac_write_count)

   Description: Get Daughter Card DAC Fifo Write Count

   Input:   void                    *DacHandle       (DAC handle pointer)
   Output:  uint                    *dac_write_count (DAC fifo write count)
   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_DAC_IS_NOT_ACTIVE     (DAC is not active)
 ***************************************************************************/
```

2.1.10 ccrtNGFC_DC_DAC_Get_Info()

This call returns some useful DAC information.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Info (void                *DacHandle,
                             ccrtngfc_dac_info_t *DacInfo)

   Description: Get Daughter Card DAC Information

   Input:    void                    *DacHandle    (DAC handle pointer)
   Output:   ccrtngfc_dac_info_t     *DacInfo      (pointer to returned DacInfo)
               # int                           Flags                // DAC flags
               # _ccrtngfc_dac_module_t        ModuleNumber         // DAC module number
               # ccrtngfc_handle_t             *Handle              // Main Device library Handle
               # ccrtngfc_local_ctrl_data_t    *local_ptr           // Main Local register pointer
               # ccrtngfc_1579321_dac_t        *local_dac_ptr       // DAC Local register pointer
               # u_int32_t                     *local_dac_fifo_ptr  // DAC Local FIFO register
                                                                        pointer
               # int                           DacFp                // DAC file descriptor pointer
               # char                          DacDeviceName[25]    // DAC Device Name
               # ccrtngfc_module_info_t        DacModuleInfo_ptr    // DAC Module Information
                                                                        pointer
   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_DAC_IS_NOT_ACTIVE  (DAC is not active)
 ************************************************************************/
```

2.1.11 ccrtNGFC_DC_DAC_Get_Interrupt_Status()

This call returns the DAC interrupt status

```
/************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Interrupt_Status (void                   *DacHandle,
                                         _ccrtngfc_intsta_dac_t *dac_fifo_intr)

   Description: Get Daughter Card DAC Interrupt Status information

   Input:   void                    *DacHandle      (DAC handle pointer)
   Output:  _ccrtngfc_intsta_dac_t  *dac_fifo_intr  (pointer to DAC FIFO interrupt status)
               # CCRTNGFC_INT_DAC_FIFO_THRESHOLD_NONE
               # CCRTNGFC_INT_DAC_FIFO_THRESHOLD_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)
 ************************************************************************/
```

2.1.12 ccrtNGFC_DC_DAC_Get_Interrupt_Timeout_Seconds()

This call returns the write time out maintained by the driver. It is the time to wait for the DAC Fifo to drain so as to have enough space for the samples to be written before the write operation commences. The device should have been opened in the block mode (*O_NONBLOCK* not set) for write to wait for the operation to complete.

```
/************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Interrupt_Timeout_Seconds (void    *DacHandle,
                                                  int     *timeout_secs)

   Description: Get Daughter Card DAC Interrupt Timeout Seconds

   Input:   void            *DacHandle           (DAC Handle pointer)
   Output:  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)
 ************************************************************************/
```

2.1.13 ccrtNGFC_DC_DAC_Get_Negative_Cal()

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

```
/************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Negative_Cal (void                              *DacHandle,
```

```
                                    _ccrtngfc_dac_channel_mask_t   ChanMask,
                                    ccrtngfc_dac_cal_t             *cal)

   Description: Get Daughter Card DAC Negative Calibration data.

   Input:   void                          *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_MASK
   Output:  ccrtngfc_dac_cal_t           *cal        (pointer to board cal)
               uint    Raw[CCRTNGFC_MAX_DAC_CHANNELS]
               double  Float[CCRTNGFC_MAX_DAC_CHANNELS]
   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_DAC_IS_NOT_ACTIVE      (DAC is not active)
 ***************************************************************************/
```

2.1.14 ccrtNGFC_DC_DAC_Get_Offset_Cal()

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

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Offset_Cal (void                          *DacHandle,
                                   _ccrtngfc_dac_channel_mask_t ChanMask,
                                   ccrtngfc_dac_cal_t           *cal)

   Description: Get Daughter Card DAC Offset Calibration data.

   Input:   void                          *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t  ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_MASK
   Output:  ccrtngfc_dac_cal_t            *cal        (pointer to board cal)
               uint    Raw[CCRTNGFC_MAX_DAC_CHANNELS]
```

```
            double  Float[CCRTNGFC_MAX_DAC_CHANNELS]
   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_DAC_IS_NOT_ACTIVE      (DAC is not active)
 ***************************************************************************/
```

2.1.15 ccrtNGFC_DC_DAC_Get_Positive_Cal()

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

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Positive_Cal (void                          *DacHandle,
                                    _ccrtngfc_dac_channel_mask_t   ChanMask,
                                    ccrtngfc_dac_cal_t             *cal)

   Description: Get Daughter Card DAC Positive Calibration data.

   Input:   void                         *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_MASK
   Output:  ccrtngfc_dac_cal_t           *cal        (pointer to board cal)
               uint    Raw[CCRTNGFC_MAX_DAC_CHANNELS]
               double  Float[CCRTNGFC_MAX_DAC_CHANNELS]
   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_DAC_IS_NOT_ACTIVE      (DAC is not active)
 ***************************************************************************/
```

2.1.16 ccrtNGFC_DC_DAC_Get_Power_Up_Status()

This call returns the power up status of the DAC module. The module must be activated before the status is returned.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Power_Up_Status(void                           *DacHandle,
                                      _ccrtngfc_dac_power_up_status_t *PowerUpStatus)

   Description: Get Power Up Status

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

```
   Output:  _ccrtngfc_dac_power_up_status_t     *PowerUpStatus          (Power Up Status)
               _ccrtngfc_dac_all_enable_t       current_state           (active/deactive)
                  # CCRTNGFC_DAC_ALL_DISABLE
                  # CCRTNGFC_DAC_ALL_ENABLE
               _ccrtngfc_presence_status_t      Plus5VoltsStatus
                  # CCRTNGFC_DAC_STATUS_NOT_PRESENT
                  # CCRTNGFC_DAC_STATUS_PRESENT
               _ccrtngfc_presence_status_t      Plus15VoltsStatus
                  # CCRTNGFC_DAC_STATUS_NOT_PRESENT
                  # CCRTNGFC_DAC_STATUS_PRESENT
               _ccrtngfc_presence_status_t      Minus15VoltsStatus
                  # CCRTNGFC_DAC_STATUS_NOT_PRESENT
                  # CCRTNGFC_DAC_STATUS_PRESENT
               _ccrtngfc_dac_power_fail_code_t  PowerFailCode
                  # CCRTNGFC_DAC_PFC_NO_ERRORS
                  # CCRTNGFC_DAC_PFC_BEFORE_PLUS_5_VOLT_OFF_ERROR
                  # CCRTNGFC_DAC_PFC_BEFORE_PLUS_15_VOLT_OFF_ERROR
                  # CCRTNGFC_DAC_PFC_BEFORE_MINUS_15_VOLT_OFF_ERROR
                  # CCRTNGFC_DAC_PFC_FIRST_PLUS_5_VOLT_OFF_ERROR
                  # CCRTNGFC_DAC_PFC_FIRST_PLUS_15_VOLT_OFF_ERROR
                  # CCRTNGFC_DAC_PFC_FIRST_MINUS_15_VOLT_OFF_ERROR
                  # CCRTNGFC_DAC_PFC_AFTER_PLUS_5_VOLT_OFF_ERROR
                  # CCRTNGFC_DAC_PFC_AFTER_PLUS_15_VOLT_OFF_ERROR
                  # CCRTNGFC_DAC_PFC_AFTER_MINUS_15_VOLT_OFF_ERROR
                char                            PowerFailCodeName[CCRTNGFC_LIB_ERROR_NAME_SIZE]
                char                            PowerFailCodeDesc[CCRTNGFC_LIB_ERROR_DESC_SIZE]
   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)
 ***************************************************************************/
```

2.1.17 ccrtNGFC_DC_DAC_Get_Speed_Mode()

This call returns the current speed of the DACs. These DACs can operate in to speeds.

1) Normal Speed
2) High Speed

When the DACs are in the high speed mode, they operate twice as fast as the normal speed mode, however, the number of usable DAC channels are reduced to half. In this case, the ODD channels are ignored and only the EVEN channels are available for use.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Speed_Mode (void                           *DacHandle,
                                   _ccrtngfc_dac_high_speed_mode_t *dac_speed_mode)

   Description: Get Daughter Card DAC Speed Mode

   Input:   void                              *DacHandle      (DAC handle pointer)
            _ccrtngfc_dac_high_speed_mode_t  &dac_speed_mode (DAC speed_mode)
               # CCRTNGFC_DAC_NORMAL_SPEED_MODE
               # CCRTNGFC_DAC_HIGH_SPEED_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_DAC_IS_NOT_ACTIVE    (DAC is not active)
 ***************************************************************************/
```

2.1.18 ccrtNGFC_DC_DAC_Get_Update_Source_Select()

This call returns to the user the current setting of the selected update source.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Get_Update_Source_Select (void                              *DacHandle,
                                             _ccrtngfc_daccsr_update_source_t *update_source)

   Description: Get Daughter Card DAC Update Source Select

   Input:   void                                *DacHandle     (DAC handle pointer)
   Output:  _ccrtngfc_daccsr_update_source_t    *update_source (pointer to update source)
               # CCRTNGFC_DAC_UPDATE_SOFTWARE
               # CCRTNGFC_DAC_UPDATE_CLOCK_0
               # CCRTNGFC_DAC_UPDATE_CLOCK_1
               # CCRTNGFC_DAC_UPDATE_CLOCK_2
               # CCRTNGFC_DAC_UPDATE_CLOCK_3
               # CCRTNGFC_DAC_UPDATE_CLOCK_4
   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_DAC_IS_NOT_ACTIVE (DAC is not active)
***************************************************************************/
```

2.1.19 ccrtNGFC_DC_DAC_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_DC_DAC_Get_Value (void                     *DacHandle,
                              CCRTNGFC_DC_DAC_CONTROL  cmd,
                              void                     *value)

   Description: Get Daughter Card DAC board register

   Input:   void                     *DacHandle  (DAC Handle pointer)
            CCRTNGFC_DC_DAC_CONTROL cmd         (register definition)
               -- structure in ccrtngfc_dac.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)
 ***************************************************************************/
```

2.1.20 ccrtNGFC_DC_DAC_MsgDma_Configure_Channel()

This call assists the user in setting up modular scatter-gather DMA descriptors for performing channel writes. A moduler scatter-gather DMA engine needs to be first acquired via the *ccrtNGFC_MsgDma_Seize()* API.

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 user also needs to supply to this call a Virtual PCI DMA Memory address which can be obtained with the *ccrtNGFC_MMap_Physical_Memory()* API. Additionally the user will also need to supply a start and end channel number. If the start channel is not divisible by a quad-word or the number of channels selected for the transfer are not a multiple of a quad-word, then the user is restricted to using only *CCRTNGFC_MSGDMA_ENGINE_4* or *CCRTNGFC_MSGDMA_ENGINE_5* engines.

```
/*************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_MsgDma_Configure_Channel (void                      *DacHandle,
                                             _ccrtngfc_msgdma_engine_t MsgDmaEngine,
                                             void                      *PciDmaMemory,
                                             uint                      StartChannelNumber,
                                             uint                      EndChannelNumber)

   Description: Configure Daughter Card DAC Modular Scatter-Gather MSG Channel Register
                descriptor

   Input:   void                              *DacHandle            (DAC handle pointer)
            _ccrtngfc_msgdma_engine_t         MsgDmaEngine
               # CCRTNGFC_MSGDMA_ENGINE_0
               # CCRTNGFC_MSGDMA_ENGINE_1
               # CCRTNGFC_MSGDMA_ENGINE_2
               # CCRTNGFC_MSGDMA_ENGINE_3
               # CCRTNGFC_MSGDMA_ENGINE_4
               # CCRTNGFC_MSGDMA_ENGINE_5
            void                               *PciDmaMemory      (Virtual PCI DMA Memory pointer)
            uint                               StartChannelNumber (Start Channel Number to Write)
            uint                               EndChannelNumber   (End Channel Number 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_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_DATA_WIDTH_NOT_MULTIPLE_OR_ALIGNED
                                                (Data Width Not Multiple or Aligned)
 ***************************************************************************/
```

2.1.21 ccrtNGFC_DC_DAC_MsgDma_Configure_Fifo()

This call assists the user in setting up modular scatter-gather DMA descriptors for performing FIFO writes. An moduler scatter-gather DMA engine needs to be first acquired via the *ccrtNGFC_MsgDma_Seize()* API.

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 user also needs to supply to this call a Virtual PCI DMA Memory address which can be obtained with the *ccrtNGFC_MMap_Physical_Memory()* API. Additionally the user will also need to supply number of samples to be read from the FIFO. If the number of samples to be read from the FIFO is not a multiple of a quad-word, then the user is restricted to using only *CCRTNGFC_MSGDMA_ENGINE_4* or *CCRTNGFC_MSGDMA_ENGINE_5* engines.

This call, on successful completion will return a pointer to the Last Descriptor ID *LastDescriptorId* which will be used by subsequent *ccrtNGFC_DC_DAC_MsgDma_Fire_Fifo()* calls to acquire the samples.

```
/***************************************************************************
   ccrtNGFC_DC_DAC_MsgDma_Configure_Fifo()

   Description: Configure Daughter Card DAC Modular Scatter-Gather MSG Fifo descriptor

   Input:   void                              *DacHandle        (DAC handle pointer)
            _ccrtngfc_msgdma_engine_t         MsgDmaEngine
               # CCRTNGFC_MSGDMA_ENGINE_0
               # CCRTNGFC_MSGDMA_ENGINE_1
               # CCRTNGFC_MSGDMA_ENGINE_2
               # CCRTNGFC_MSGDMA_ENGINE_3
               # CCRTNGFC_MSGDMA_ENGINE_4
               # CCRTNGFC_MSGDMA_ENGINE_5
            void                              *PciDmaMemory     (Virtual PCI DMA Memory pointer)
            uint                              NumberOfSamples   (number of FIFO samples to write)
   Output:  _ccrtngfc_msgdma_descriptors_id_t *LastDescriptorId (pointer to last descriptor id)
   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_DESCRIPTORS_AVAILABLE (no free descriptors available)
               # 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_DATA_WIDTH_NOT_MULTIPLE_OR_ALIGNED
                                                (Data Width Not Multiple or Aligned)
 ************************************************************************/
```

2.1.22 ccrtNGFC_DC_DAC_MsgDma_Fire_Channel()

This call initiates a scatter-gather DMA operation that has been previously configured and setup by the *ccrtNGFC_DC_DAC_MsgDma_Configure_Channel()* API for a given MsgDma Engine.

Once the scatter-gather DMA operation commences, it performs a DMA operation on the selected channels in the *ccrtNGFC_DC_DAC_MsgDma_Configure_Channel()* API until the DMA operation completes.

```
/************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_MsgDma_Fire_Channel (void                           *DacHandle,
                                        _ccrtngfc_msgdma_engine_t      MsgDmaEngine)

   Description: Fire DAC Daughter Card Channel Modular Scatter-Gather DMA descriptor

   Input:   void                               *DacHandle          (DAC handle pointer)
            _ccrtngfc_msgdma_engine_t          MsgDmaEngine
               # CCRTNGFC_MSGDMA_ENGINE_0
               # CCRTNGFC_MSGDMA_ENGINE_1
               # CCRTNGFC_MSGDMA_ENGINE_2
               # CCRTNGFC_MSGDMA_ENGINE_3
               # CCRTNGFC_MSGDMA_ENGINE_4
               # CCRTNGFC_MSGDMA_ENGINE_5
   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)
 ************************************************************************/
```

2.1.23 ccrtNGFC_DC_DAC_MsgDma_Fire_Fifo()

This call initiates a scatter-gather DMA operation that has been previously configured and setup by the *ccrtNGFC_DC_DAC_MsgDma_Configure_Fifo()* API for a given MsgDma Engine.

Once the scatter-gather DMA operation commences, it performs a DMA operation on the number of samples specified in the *ccrtNGFC_DC_DAC_MsgDma_Configure_Fifo()* API until the DMA operation completes.

```
/************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_MsgDma_Fire_Fifo (void                              *DacHandle,
                                     _ccrtngfc_msgdma_engine_t         MsgDmaEngine,
                                     _ccrtngfc_msgdma_descriptors_id_t LastDescriptorId,
                                     int                               UseInterrupts)

   Description: Fire DAC Daughter Card Fifo Modular Scatter-Gather DMA descriptor

   Input:   void                               *DacHandle          (DAC handle pointer)
            _ccrtngfc_msgdma_engine_t          MsgDmaEngine
               # CCRTNGFC_MSGDMA_ENGINE_0
               # CCRTNGFC_MSGDMA_ENGINE_1
               # CCRTNGFC_MSGDMA_ENGINE_2
```

```
            # CCRTNGFC_MSGDMA_ENGINE_3
            # CCRTNGFC_MSGDMA_ENGINE_4
            # CCRTNGFC_MSGDMA_ENGINE_5
         _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)
 ***************************************************************************/
```

2.1.24 ccrtNGFC_DC_DAC_Open()

This open allows the user to access the various functions on the DAC daughter card which is physically located in one of the two FMC slots on the mother board. Since there can be up to two DAC daughter cards located on a mother board, this call requires the user to identify which DAC module is to be accessed via the *ModuleNumber CCRTNGFC_DAC_MODULE_0* or *CCRTNGFC_DAC_MODULE_1* argument.

This is the first call that needs to be issued by a user to open the DAC daughter card and access the various functions on the card, through the rest of the API calls. If the DAC daughter card is not installed on the mother board, the call will fail with an *CCRTNGFC_LIB_MODULE_NOT_PRESENT* error. Prior to issuing this call, the user can always query the mother board to see if the card is installed by issuing the *ccrtNGFC_DC_DAC_Presence* call.

What is returned on successful completion of the call is a handle to a DAC pointer *DacHandle* that is supplied as an argument to all the other DAC specific API calls that start with *ccrtNGFC_DC_DAC_*()*.

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 *write(2)* calls which will change the default writing in block mode where the driver will wait for the DAC FIFO to drain so that there is enough space to write the user supplied samples.

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_DC_DAC_Open (void                    **DacHandle,
                         void                    *Handle,
                         _ccrtngfc_dac_module_t  ModuleNumber,
                         int                     oflag)

   Description: Open a Daughter Card DAC module.

   Input:   void                    **DacHandle      (DAC handle pointer to pointer)
            void                    *Handle          (Device Handle pointer)
            _ccrtngfc_dac_module_t ModuleNumber     (DAC module number)
               # CCRTNGFC_DAC_MODULE_0              (DAC module on daughter card 0)
               # CCRTNGFC_DAC_MODULE_1              (DAC module on daughter card 1)
            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 mmapped)
               # CCRTNGFC_LIB_MMAP_SELECT_FAILED   (mmap selection failed)
               # CCRTNGFC_LIB_MMAP_FAILED          (mmap failed)
               # CCRTNGFC_LIB_MODULE_ALREADY_OPEN  (module already opened)
               # CCRTNGFC_LIB_MODULE_NOT_PRESENT   (module not present)
               # CCRTNGFC_LIB_MODULE_OPEN_FAILED   (module open failed)
 ***************************************************************************/
```

2.1.25 ccrtNGFC_DC_DAC_Perform_Auto_Calibration()

This single call performs a full DAC calibration of the selected channels using the internal reference voltages.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Perform_Auto_Calibration (void                          *DacHandle,
                                            _ccrtngfc_dac_channel_mask_t   ChanMask)

   Description: Perform Daughter Card DAC Auto Calibration

   Input:   void                        *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_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             (library not open)
               # CCRTNGFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCRTNGFC_LIB_NO_RESOURCE          (no free PLL available)
               # CCRTNGFC_LIB_IO_ERROR             (read error)
               # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE    (DAC is not active)
               # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE  (Clock is not active)
 ***************************************************************************/
```

2.1.26 ccrtNGFC_DC_DAC_Perform_Negative_Calibration()

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

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Perform_Negative_Calibration (void                          *DacHandle,
                                                _ccrtngfc_dac_channel_mask_t   ChanMask)

   Description: Perform Daughter Card DAC Negative Calibration
```

```
   Input:   void                          *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_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             (library not open)
               # CCRTNGFC_LIB_NO_LOCAL_REGION      (local region not present)
               # CCRTNGFC_LIB_NO_RESOURCE          (no free PLL available)
               # CCRTNGFC_LIB_IO_ERROR             (read error)
               # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE    (DAC is not active)
               # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE  (Clock is not active)
 ************************************************************************/
```

2.1.27 ccrtNGFC_DC_DAC_Perform_Offset_Calibration()

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

```
/************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Perform_Offset_Calibration (void                          *DacHandle,
                                               _ccrtngfc_dac_channel_mask_t ChanMask)

   Description: Perform Daughter Card DAC Offset Calibration

   Input:   void                          *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_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             (library not open)
```

```
            # CCRTNGFC_LIB_NO_LOCAL_REGION       (local region not present)
            # CCRTNGFC_LIB_NO_RESOURCE           (no free PLL available)
            # CCRTNGFC_LIB_IO_ERROR              (read error)
            # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE     (DAC is not active)
            # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
 ***************************************************************************/
```

2.1.28 ccrtNGFC_DC_DAC_Perform_Positive_Calibration()

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

```
/***************************************************************************
   ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Perform_Positive_Calibration (void                          *DacHandle,
                                                 _ccrtngfc_dac_channel_mask_t  ChanMask)

   Description: Perform Daughter Card DAC Positive Calibration

   Input:   void                         *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_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              (library not open)
               # CCRTNGFC_LIB_NO_LOCAL_REGION       (local region not present)
               # CCRTNGFC_LIB_NO_RESOURCE           (no free PLL available)
               # CCRTNGFC_LIB_IO_ERROR              (read error)
               # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE     (DAC is not active)
               # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
 ***************************************************************************/
```

2.1.29 ccrtNGFC_DC_DAC_Presence()

This call is used to see if an DAC daughter card is installed in one of the two FMC slots on the mother board.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Presence (void                     *Handle,
                             _ccrtngfc_dac_module_t   ModuleNumber,
                             int                      *DacPresent)

   Description: Check for DAC Daughter Card Presence

   Input:   void                         *Handle         (Board Handle pointer)
            _ccrtngfc_dac_module_t       ModuleNumber    (DAC module number to check)
               # CCRTNGFC_DAC_MODULE_0                   (DAC module on daughter card 0)
               # CCRTNGFC_DAC_MODULE_1                   (DAC module on daughter card 1)
```

```
   Output:  int                          *DacPresent     (DAC Present)
               # CCRTNGFC_CARD_NOT_PRESENT
               # CCRTNGFC_CARD_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)
 ***************************************************************************/
```

2.1.30 ccrtNGFC_DC_DAC_ReadBack_Channels()

This call loops back the output DAC channels and read the values via the ADC located on the same daughter card.

The user can also supply the *adc_csr* belonging to the ADC Channel *ADCChan* or *NULL* argument. If the user specifies NULL for the *adc_csr* then this call reads the current hardware CSR to convert the RAW ADC value to the corresponding floating point voltage. This is done everytime the API is executed. If the *adc_csr* is not expected to change between channel writes, the user can get the current *adc_csr* once and supply that as an arugment to the call, thus speeding up the readback channel operation.

If the *adc_csr* is specified by the user, they need to ensure that the update clock is one of the available update clocks and is programmed. Additionally the input signal must be *CCRTNGFC_ADC_CALIBRATION_BUS* and the data format must be *CCRTNGFC_ADC_TWOS_COMPLEMENT*.

```
/***************************************************************************
_ccrtngfc_lib_error_number_t
ccrtNGFC_DC_DAC_ReadBack_Channels (void                          *DacHandle,
                                   _ccrtngfc_dac_channel_mask_t DACChanMask,
                                   _ccrtngfc_adc_channel_t      ADCChan,
                                   _ccrtngfc_adc_csr_t          *adc_csr,
                                   ccrtngfc_dac_volts_t         *dac_volts)

   Description: Read Back Daughter Card DAC Channels using ADC

   Input:   void                                  *DacHandle    (DacHandle pointer)
            _ccrtngfc_dac_channel_mask_t          DACChanMask   (specify DAC channel mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_MASK
            _ccrtngfc_adc_channel_t                  ADCChan    (ADC channel to read)
            _ccrtngfc_adc_csr_t                      *adc_csr   (pointer to ADC csr)
               _ccrtngfc_adccsr_update_clock_t       adc_update_clock
                  # CCRTNGFC_ADC_UPDATE_CLOCK_NONE
                  # CCRTNGFC_ADC_UPDATE_CLOCK_0
                  # CCRTNGFC_ADC_UPDATE_CLOCK_1
                  # CCRTNGFC_ADC_UPDATE_CLOCK_2
                  # CCRTNGFC_ADC_UPDATE_CLOCK_3
                  # CCRTNGFC_ADC_UPDATE_CLOCK_4
               _ccrtngfc_adccsr_input_signal_t      adc_input_signal
```

```
               # CCRTNGFC_ADC_EXTERNAL_SIGNAL
               # CCRTNGFC_ADC_CALIBRATION_BUS
            _ccrtngfc_adccsr_data_format_t       adc_data_format
               # CCRTNGFC_ADC_OFFSET_BINARY
               # CCRTNGFC_ADC_TWOS_COMPLEMENT
   Output:  ccrtngfc_dac_volts_t                 *dac_volts (pointer to DAC volts)
               uint    Raw[CCRTNGFC_MAX_DAC_CHANNELS];
               double  Float[CCRTNGFC_MAX_DAC_CHANNELS];
   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_NO_LOCAL_REGION      (local region not present)
            # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE    (DAC is not active)
            # CCRTNGFC_LIB_ADC_IS_NOT_ACTIVE    (ADC is not active)
            # CCRTNGFC_LIB_CLOCK_IS_NOT_ACTIVE  (Clock is not active)
            # CCRTNGFC_LIB_ADC_INCORRECTLY_CONFIGURED
                                                (ADC incorrectly configured for DAC readback)
 ****************************************************************************/
```

2.1.31 ccrtNGFC_DC_DAC_Read_Channels()

This routine reads the DAC channel data and returns the raw and floating point voltages in the *dac_volts* argument. This is the data that was written to the DAC channel registers.

```
/****************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Read_Channels (void                          *DacHandle,
                                  _ccrtngfc_dac_channel_mask_t ChanMask,
                                  _ccrtngfc_dac_csr_t          *dac_csr,
                                  ccrtngfc_dac_volts_t         *dac_volts)

   Description: Read Daughter Card DAC Channels

   Input:   void                          *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_MASK
            _ccrtngfc_dac_csr_t             *dac_csr (pointer to DAC csr)
               _ccrtngfc_daccsr_update_clock_t      dac_update_clock
                  # CCRTNGFC_DAC_UPDATE_CLOCK_NONE
                  # CCRTNGFC_DAC_UPDATE_CLOCK_0
                  # CCRTNGFC_DAC_UPDATE_CLOCK_1
                  # CCRTNGFC_DAC_UPDATE_CLOCK_2
                  # CCRTNGFC_DAC_UPDATE_CLOCK_3
                  # CCRTNGFC_DAC_UPDATE_CLOCK_4
               _ccrtngfc_daccsr_input_signal_t      dac_input_signal
                  # CCRTNGFC_DAC_EXTERNAL_SIGNAL
                  # CCRTNGFC_DAC_CALIBRATION_BUS
```

```
            _ccrtngfc_daccsr_data_format_t       dac_data_format
               # CCRTNGFC_DAC_OFFSET_BINARY
               # CCRTNGFC_DAC_TWOS_COMPLEMENT
   Output:  ccrtngfc_dac_volts_t            *dac_volts (pointer to DAC volts)
               uint    Raw[CCRTNGFC_MAX_DAC_CHANNELS];
               double  Float[CCRTNGFC_MAX_DAC_CHANNELS];
   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_NO_LOCAL_REGION       (local region not present)
               # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE     (DAC is not active)
 ************************************************************************/
```

2.1.32 ccrtNGFC_DC_DAC_Read_Channels_Calibration()

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

```
/************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Read_Channels_Calibration (void *DacHandle,
                                              char *filename)

   Description: Read Daughter Card DAC Channels Calibration

   Input:    void   *DacHandle                   (DAC handle pointer)
   Output:   char   *filename                    (pointer to filename)
   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_DAC_IS_NOT_ACTIVE  (DAC is not active)
 ************************************************************************/
```

2.1.33 ccrtNGFC_DC_DAC_Reset_Calibration()

This call resets the DAC calibration values on the card.

```
/************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Reset_Calibration (void                          *DacHandle,
                                     _ccrtngfc_dac_channel_mask_t  ChanMask)

   Description: Reset Daughter Card DAC Calibration

   Input:   void                         *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
```

```
            # CCRTNGFC_DAC_CHANNEL_MASK_10
            # CCRTNGFC_DAC_CHANNEL_MASK_11
            # CCRTNGFC_ALL_DAC_CHANNELS_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_NO_LOCAL_REGION   (local region not present)
            # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
 ***************************************************************************/
```

2.1.34 ccrtNGFC_DC_DAC_Reset_Fifo()

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

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Reset_Fifo (void                       *DacHandle,
                               _ccrtngfc_dac_fifo_reset_t  activate)

   Description: Reset Daughter Card DAC Fifo

   Input:   void                       *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_fifo_reset_t  activate    (activate converter)
               # CCRTNGFC_DAC_FIFO_ACTIVATE
               # CCRTNGFC_DAC_FIFO_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_INVALID_ARG       (invalid argument)
               # CCRTNGFC_LIB_NO_LOCAL_REGION   (local region not present)
               # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE (DAC is not active)
 ***************************************************************************/
```

2.1.35 ccrtNGFC_DC_DAC_Set_CSR()

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

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_CSR (void                  *DacHandle,
                            _ccrtngfc_dac_mask_t  dac_mask,
                            _ccrtngfc_dac_csr_t   *dac_csr)

   Description: Set Daughter Card DAC Control and Status information

   Input:   void                                    *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_mask_t                    dac_mask    (selected DAC mask)
               # CCRTNGFC_DAC_MASK_0_1
               # CCRTNGFC_DAC_MASK_2_3
               # CCRTNGFC_DAC_MASK_4_5
               # CCRTNGFC_DAC_MASK_6_7
               # CCRTNGFC_DAC_MASK_8_9
               # CCRTNGFC_DAC_MASK_10_11
            _ccrtngfc_dac_csr_t                     *dac_csr    (pointer to DAC csr)
               _ccrtngfc_daccsr_updmode_t           dac_update_mode
                  # CCRTNGFC_DAC_MODE_IMMEDIATE
```

```
                  # CCRTNGFC_DAC_MODE_SYNCHRONIZED
                  # CCRTNGFC_DAC_MODE_DO_NOT_CHANGE
               _ccrtngfc_daccsr_dataformat_t            dac_data_format
                  # CCRTNGFC_DAC_OFFSET_BINARY
                  # CCRTNGFC_DAC_TWOS_COMPLEMENT
                  # CCRTNGFC_DAC_DATA_FORMAT_DO_NOT_CHANGE
               _ccrtngfc_daccsr_output_select_t         dac_output_select
                  # CCRTNGFC_DAC_SINGLE_ENDED
                  # CCRTNGFC_DAC_DIFFERENTIAL
                  # CCRTNGFC_DAC_OUTPUT_SELECT_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_DAC_IS_NOT_ACTIVE        (DAC is not active)
 ***************************************************************************/
```

2.1.36 ccrtNGFC_DC_DAC_Set_Driver_Write_Mode()

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

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_Driver_Write_Mode (void                                *DacHandle,
                                          _ccrtngfc_driver_DAC_write_mode_t   mode)

   Description: Set Driver Daughter Card DAC Write Mode

   Input:   void                              *DacHandle    (DAC handle pointer)
            _ccrtngfc_driver_DAC_write_mode_t mode          (select DAC write mode)
               # CCRTNGFC_DAC_PIO_CHANNEL
               # CCRTNGFC_DAC_PIO_FIFO
   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)
 ***************************************************************************/
```

2.1.37 ccrtNGFC_DC_DAC_Set_Fifo_Channel_Select()

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

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_Fifo_Channel_Select (void                *DacHandle,
                                            _ccrtngfc_dac_channel_mask_t
                                                                dac_fifo_channel_select_mask)

   Description: Set Daughter Card DAC Fifo Channel Selection

   Input:   void                          *DacHandle                     (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t dac_fifo_channel_select_mask   (channel select mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
```

```
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_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_NO_LOCAL_REGION     (local region not present)
               # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE   (DAC is not active)
 ****************************************************************************/
/****************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_Fifo_Channel_Select (void                        *DacHandle,
                                            _ccrtngfc_dac_channel_mask_t
                                                                 dac_fifo_channel_select_mask)

   Description: Set Daughter Card DAC Fifo Channel Selection

   Input:   void                        *DacHandle                    (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t dac_fifo_channel_select_mask  (channel select mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_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_NO_LOCAL_REGION     (local region not present)
               # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE   (DAC is not active)
 ****************************************************************************/
```

2.1.38 ccrtNGFC_DC_DAC_Set_Fifo_Threshold()

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

```
/****************************************************************************
   ccrtNGFC_DC_DAC_Set_Fifo_Threshold()
  _ccrtngfc_lib_error_number_t
```

```
  ccrtNGFC_DC_DAC_Set_Fifo_Threshold (void *DacHandle,
                                      uint dac_threshold)

   Description: Set Daughter Card DAC Fifo Threshold

   Input:   void                    *Handle        (handle pointer)
            uint                    dac_threshold  (DAC fifo threshold)
   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_DAC_IS_NOT_ACTIVE    (DAC is not active)
 ***************************************************************************/
```

2.1.39 ccrtNGFC_DC_DAC_Set_Fifo_Write_Count()

This call allows the user to set the DAC FIFO write count.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_Fifo_Write_Count (void *DacHandle,
                                         uint dac_write_count)

   Description: Set Daughter Card DAC Fifo Write Count

   Input:   void                    *DacHandle       (DAC handle pointer)
            uint                    dac_write_count  (DAC fifo write count)
   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_DAC_IS_NOT_ACTIVE      (DAC is not active)
 ***************************************************************************/
```

2.1.40 ccrtNGFC_DC_DAC_Set_Interrupt_Timeout_Seconds()

This call sets the write DAC *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 to wait for the DAC Fifo to drain so as to have enough space for the samples to be written before the write operation commences. The device should have been opened in the block mode (*O_NONBLOCK* not set) for write to wait for the operation to complete.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_Interrupt_Timeout_Seconds (void    *DacHandle,
                                                  int     timeout_secs)

   Description: Set Daughter Card DAC Interrupt Timeout Seconds

   Input:   void         *DacHandle         (DAC 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)
 ***************************************************************************/
```

2.1.41 ccrtNGFC_DC_DAC_Set_Negative_Cal()

This call allows the user to set the negative calibration data for all the channels by supplying floating point *Float* gains to the call. Additionally, this call will return the *RAW* value of the gain supplied that is written to the board.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_Negative_Cal (void                        *DacHandle,
                                    _ccrtngfc_dac_channel_mask_t ChanMask,
                                    ccrtngfc_dac_cal_t           *cal)

   Description: Set Daughter Card DAC Negative Calibration data.

   Input:   void                         *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_MASK
            ccrtngfc_dac_cal_t         *cal       (pointer to board cal)
               uint    Raw[CCRTNGFC_MAX_DAC_CHANNELS]
               double  Float[CCRTNGFC_MAX_DAC_CHANNELS]
   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_DAC_IS_NOT_ACTIVE  (DAC is not active)
 ***************************************************************************/
```

2.1.42 ccrtNGFC_DC_DAC_Set_Offset_Cal()

This call allows the user to set the offset calibration data for all the channels by supplying floating point *Float* offset to the call. Additionally, this call will return the *Raw* value of the offset supplied that is written to the board.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_Offset_Cal (void                        *DacHandle,
                                   _ccrtngfc_dac_channel_mask_t ChanMask,
                                   ccrtngfc_dac_cal_t           *cal)

   Description: Set Daughter Card DAC Offset Calibration data.

   Input:   void                         *DacHandle  (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
```

```
            # CCRTNGFC_DAC_CHANNEL_MASK_1
            # CCRTNGFC_DAC_CHANNEL_MASK_2
            # CCRTNGFC_DAC_CHANNEL_MASK_3
            # CCRTNGFC_DAC_CHANNEL_MASK_4
            # CCRTNGFC_DAC_CHANNEL_MASK_5
            # CCRTNGFC_DAC_CHANNEL_MASK_6
            # CCRTNGFC_DAC_CHANNEL_MASK_7
            # CCRTNGFC_DAC_CHANNEL_MASK_8
            # CCRTNGFC_DAC_CHANNEL_MASK_9
            # CCRTNGFC_DAC_CHANNEL_MASK_10
            # CCRTNGFC_DAC_CHANNEL_MASK_11
            # CCRTNGFC_ALL_DAC_CHANNELS_MASK
         ccrtngfc_dac_cal_t         *cal       (pointer to board cal)
            uint    Raw[CCRTNGFC_MAX_DAC_CHANNELS]
            double  Float[CCRTNGFC_MAX_DAC_CHANNELS]
   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_DAC_IS_NOT_ACTIVE  (DAC is not active)
 ************************************************************************/
```

2.1.43 ccrtNGFC_DC_DAC_Set_Positive_Cal()

This call allows the user to set the positive calibration data for all the channels by supplying floating point *Float* gains to the call. Additionally, this call will return the *Raw* value of the gain supplied that is written to the board.

```
/************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_Positive_Cal (void                        *DacHandle,
                                    _ccrtngfc_dac_channel_mask_t ChanMask,
                                    ccrtngfc_dac_cal_t           *cal)

   Description: Set Daughter Card DAC Positive Calibration data.

   Input:   void                        *DacHandle   (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t ChanMask    (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_MASK
            ccrtngfc_dac_cal_t         *cal       (pointer to board cal)
               uint    Raw[CCRTNGFC_MAX_DAC_CHANNELS]
               double  Float[CCRTNGFC_MAX_DAC_CHANNELS]
   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_DAC_IS_NOT_ACTIVE   (DAC is not active)
 ***************************************************************************/
```

2.1.44 ccrtNGFC_DC_DAC_Set_Speed_Mode()

This call sets the current speed of the DACs. These DACs can operate in to speeds.

1) Normal Speed
2) High Speed

When the DACs are in the high speed mode, they operate twice as fast as the normal speed mode, however, the number of usable DAC channels are reduced to half. In this case, the ODD channels are ignored and only the EVEN channels are available for use.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_Speed_Mode (void                          *DacHandle,
                                   _ccrtngfc_dac_high_speed_mode_t dac_speed_mode)

   Description: Set Daughter Card DAC Speed Mode

   Input:   void                              *DacHandle     (DAC handle pointer)
            _ccrtngfc_dac_high_speed_mode_t  dac_speed_mode (DAC speed_mode)
               # CCRTNGFC_DAC_NORMAL_SPEED_MODE
               # CCRTNGFC_DAC_HIGH_SPEED_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_DAC_IS_NOT_ACTIVE    (DAC is not active)
***************************************************************************/
```

2.1.45 ccrtNGFC_DC_DAC_Set_Update_Source_Select()

This call sets the update source. It can be either set to software update or based on one of the available clocks.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Set_Update_Source_Select (void                              *DacHandle,
                                             _ccrtngfc_daccsr_update_source_t update_source)

   Description: Set Daughter Card DAC Update Source Select

   Input:   void                                 *DacHandle     (DAC handle pointer)
            _ccrtngfc_daccsr_update_source_t     update_source  (pointer to update source)
               # CCRTNGFC_DAC_UPDATE_SOFTWARE
               # CCRTNGFC_DAC_UPDATE_CLOCK_0
               # CCRTNGFC_DAC_UPDATE_CLOCK_1
               # CCRTNGFC_DAC_UPDATE_CLOCK_2
               # CCRTNGFC_DAC_UPDATE_CLOCK_3
               # CCRTNGFC_DAC_UPDATE_CLOCK_4
   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_DAC_IS_NOT_ACTIVE (DAC is not active)
 ************************************************************************/
```

2.1.46 ccrtNGFC_DC_DAC_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_DC_DAC_Set_Value (void                     *DacHandle,
                              CCRTNGFC_DC_DAC_CONTROL  cmd,
                              void                     *value)

   Description: Set Daughter Card DAC value of the specified board register.

   Input:   void                     *DacHandle     (DAC Handle pointer)
            CCRTNGFC_DC_DAC_CONTROL cmd            (register definition)
               -- structure in ccrtngfc_dac.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)
 ************************************************************************/
```

2.1.47 ccrtNGFC_DC_DAC_Wait_For_Channel_Idle()

If the user is updating the channel registers very fast, it is possible that an update can get missed as the hardware is busy processing the previous channel update. The user can use this call to ensure the update is complete my monitoring the DAC to become idle.

```
/************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Wait_For_Channel_Idle (void                    *DacHandle,
                                          _ccrtngfc_dac_channel_t dac_channel)

   Description: Wait for Daughter Card DAC Channel to go idle

   Input:   void                        *DacHandle      (DAC handle pointer)
            _ccrtngfc_dac_channel_t     dac_channel     (dac channel number)
               # CCRTNGFC_DAC_CHANNEL_0
               # CCRTNGFC_DAC_CHANNEL_1
               # CCRTNGFC_DAC_CHANNEL_2
               # CCRTNGFC_DAC_CHANNEL_3
               # CCRTNGFC_DAC_CHANNEL_4
               # CCRTNGFC_DAC_CHANNEL_5
               # CCRTNGFC_DAC_CHANNEL_6
               # CCRTNGFC_DAC_CHANNEL_7
               # CCRTNGFC_DAC_CHANNEL_8
               # CCRTNGFC_DAC_CHANNEL_9
               # CCRTNGFC_DAC_CHANNEL_10
               # CCRTNGFC_DAC_CHANNEL_11
```

```
   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_INVALID_ARG             (Invalid argument)
               # CCRTNGFC_LIB_DAC_FIFO_UNDERFLOW      (DAC Fifo underflow)
               # CCRTNGFC_LIB_DAC_IS_BUSY             (DAC is busy)
 ***************************************************************************/
```

2.1.48 ccrtNGFC_DC_DAC_Wait_For_Fifo_To_Drain()

Before writing to the DAC FIFO, the user needs to ensure that there is sufficient space available to write the requested samples. This call allows the user to block and wait until there is enough space available in the FIFO to write the samples.

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Wait_For_Fifo_To_Drain (void   *DacHandle,
                                           uint   fifo_threshold)

   Description: Wait for Daughter Card DAC Fifo to drain

   Input:   void                      *DacHandle      (DAC handle pointer)
            uint                      fifo_threshold  (fifo threshold)
   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_INVALID_ARG             (Invalid argument)
               # CCRTNGFC_LIB_DAC_FIFO_UNDERFLOW      (DAC Fifo underflow)
 ***************************************************************************/
```

2.1.49 ccrtNGFC_DC_DAC_Write()

This call performs a DAC driver write operation

```
/***************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Write (void    *DacHandle,
                          void    *buf,
                          int     size,
                          int     *bytes_written,
                          int     *error)

   Description: Perform Daughter Card DAC driver write operation.

   Input:   void     *DacHandle                (DAC handle pointer)
            int      size                      (number of bytes to write)
   Output:  void     *buf                      (pointer to buffer)
            int      *bytes_written            (bytes written)
            int      *error                    (returned errno)
   Return:  _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          (write failed)
               # CCRTNGFC_LIB_NOT_IMPLEMENTED   (call not implemented)
```

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

2.1.50 ccrtNGFC_DC_DAC_Write_Channels()

This call assists the user in writing to the DAC channels. The user needs to supply a floating point DAC voltage in the *dac_volts* argument. Additionally, the user can also supply the *dac_csr* or *NULL* argument. If the user specifies NULL for the *dac_csr* then this call reads the current hardware CSR to convert the user specified floating point voltage to the RAW value that needs to be written to the channel. This is done everytime the API is executed. If the *dac_csr* is not expected to change between channel writes, the user can get the current *dac_csr* once and supply that as an arugment to the call, thus speeding up the write channel operation.

```
/****************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Write_Channels (void                          *DacHandle,
                                   _ccrtngfc_dac_channel_mask_t ChanMask,
                                   _ccrtngfc_dac_csr_t          *dac_csr,
                                   ccrtngfc_dac_volts_t         *dac_volts)

   Description: Write Daughter Card DAC Channels

   Input:   void                          *DacHandle   (DAC handle pointer)
            _ccrtngfc_dac_channel_mask_t  ChanMask     (channel selection mask)
               # CCRTNGFC_DAC_CHANNEL_MASK_0
               # CCRTNGFC_DAC_CHANNEL_MASK_1
               # CCRTNGFC_DAC_CHANNEL_MASK_2
               # CCRTNGFC_DAC_CHANNEL_MASK_3
               # CCRTNGFC_DAC_CHANNEL_MASK_4
               # CCRTNGFC_DAC_CHANNEL_MASK_5
               # CCRTNGFC_DAC_CHANNEL_MASK_6
               # CCRTNGFC_DAC_CHANNEL_MASK_7
               # CCRTNGFC_DAC_CHANNEL_MASK_8
               # CCRTNGFC_DAC_CHANNEL_MASK_9
               # CCRTNGFC_DAC_CHANNEL_MASK_10
               # CCRTNGFC_DAC_CHANNEL_MASK_11
               # CCRTNGFC_ALL_DAC_CHANNELS_MASK
            _ccrtngfc_dac_csr_t                    *dac_csr (pointer to DAC csr)
               _ccrtngfc_daccsr_busy_t             dac_interface_busy
                  # CCRTNGFC_DAC_IDLE
                  # CCRTNGFC_DAC_BUSY
               _ccrtngfc_daccsr_updmode_t          dac_update_mode
                  # CCRTNGFC_DAC_MODE_IMMEDIATE
                  # CCRTNGFC_DAC_MODE_SYNCHRONIZED
               _ccrtngfc_daccsr_data_format_t      dac_data_format
                  # CCRTNGFC_DAC_OFFSET_BINARY
                  # CCRTNGFC_DAC_TWOS_COMPLEMENT
               _ccrtngfc_daccsr_output_select_t    dac_output_select
                  # CCRTNGFC_DAC_SINGLE_ENDED
                  # CCRTNGFC_DAC_DIFFERENTIAL
            ccrtngfc_dac_volts_t                   *dac_volts (pointer to DAC volts)
                  uint    Raw[CCRTNGFC_MAX_DAC_CHANNELS];
                  double  Float[CCRTNGFC_MAX_DAC_CHANNELS];
   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_NO_LOCAL_REGION      (local region not present)
               # CCRTNGFC_LIB_INVALID_ARG          (invalid argument)
               # CCRTNGFC_LIB_VOLTAGE_NOT_IN_RANGE (voltage not in range)
               # CCRTNGFC_LIB_DAC_IS_NOT_ACTIVE    (DAC is not active)
 ****************************************************************************/
```

2.1.51 ccrtNGFC_DC_DAC_Write_Channels_Calibration()

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

```
/*************************************************************************
   _ccrtngfc_lib_error_number_t
   ccrtNGFC_DC_DAC_Write_Channels_Calibration (void *DacHandle,
                                               char *filename)

   Description: Write Daughter Card DAC Channels Calibration

   Input:    void   *DacHandle                    (DAC handle pointer)
   Output:   char   *filename                     (pointer to filename)
   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_DAC_IS_NOT_ACTIVE  (DAC is not active)
 *************************************************************************/
```

This page intentionally left blank