Software Interface
CCURDSCC (WC-AD3224-DS)

PCIe 32-Channel Delta Sigma Converter Card (DSCC)

Driver	ccurdscc (WC-AD3224-DS)	Rev 6.3
OS	RedHawk	Rev 6.3
Vendor	Concurrent Computer Corporation	
Hardware	PCIe 32-Channel Delta Sigma Converter Card (CP-AD3224-DS)	
Date	August 2nd, 2013	

This page intentionally left blank

Table of Contents

This page intentionally left blank

1. Introduction

This document provides the software interface to the ***ccurdscc*** driver which communicates with the Concurrent Computer Corporation PCI Express 32-Channel Delta Sigma Converter Card (DSCC). For additional information on programming, please refer to the *Concurrent Computer Corporation PCIe 32-Channel Delta Sigma Converter Cards (DSCC) Design Specification (No. 0610099)* 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 results.

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 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 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 writing their applications.

1.1 Related Documents

- Analog Input Driver Installation on RedHawk Release Notes by Concurrent Computer Corporation.
- PCIe 32-Channel Delta Sigma Converter Card (DSCC) Design Specification (No. 0610099) by Concurrent Computer Corporation.

2. Software Support

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

2.1 Direct Driver Access

2.1.1 open(2) system call

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

```
int   fp;
fp = open("/dev/ccurdscc0", O_RDWR);
```

The file pointer '*fp'* is then used as an argument to other system calls. The user can also supply the O_NONBLOCK flag if the user does not wish to block waiting for reads to complete. In that case, if the read is not satisfied, the call will fail. The device name specified is of the format "/dev/ccurdscc<num>" where *num* is a digit 0..9 which represents the board number that is to be accessed.

2.1.2 ioctl(2) system call

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

```
int   status;
int   arg;
```

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

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

Driver IOCTL command:

```
IOCTL_CCURDSCC_ABORT_DMA
IOCTL_CCURDSCC_ADD_IRQ
IOCTL_CCURDSCC_ALLOCATE_DMA_BUFFERS
IOCTL_CCURDSCC_DISABLE_PCI_INTERRUPTS
IOCTL_CCURDSCC_ENABLE_PCI_INTERRUPTS
IOCTL_CCURDSCC_GET_DRIVER_ERROR
IOCTL_CCURDSCC_GET_DRIVER_INFO
IOCTL_CCURDSCC_GET_NUM_DMA_BUFFERS
IOCTL_CCURDSCC_GET_PHYSICAL_MEMORY
IOCTL_CCURDSCC_GET_READ_MODE
IOCTL_CCURDSCC_INIT_BOARD
IOCTL_CCURDSCC_INTERRUPT_TIMEOUT_SECONDS
IOCTL_CCURDSCC_MMAP_SELECT
IOCTL_CCURDSCC_NO_COMMAND
IOCTL_CCURDSCC_PRESERVE_LIB_INFO
IOCTL_CCURDSCC_READ_EEPROM
IOCTL_CCURDSCC_REMOVE_DMA_BUFFERS
IOCTL_CCURDSCC_REMOVE_IRQ
IOCTL_CCURDSCC_RESET_BOARD
IOCTL_CCURDSCC_RESET_DMA_CONTINUOUS_BUFFERS
IOCTL_CCURDSCC_SELECT_READ_MODE
IOCTL_CCURDSCC_WAIT_FOR_INTERRUPT
IOCTL_CCURDSCC_WRITE_EEPROM
```

IOCTL_CCURDSCC_ABORT_DMA: This *ioctl* does not have any arguments. Its purpose is to abort any DMA already in progress. It will also reset the FIFO and the DMA continuous buffers.

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

IOCTL_CCURDSCC_ALLOCATE_DMA_BUFFERS: This *ioctl* creates DMA buffers that are to be used during reads, when operating in the *CCURDSCC_DMA_CONTINUOUS* mode. The argument is a pointer to an *unsigned short* that specifies the number of buffers to be allocated. If the buffer count is 0, no buffers are allocated and the user will be unable to perform reads using the *CCURDSCC_DMA_CONTINUOUS* mode. Each DMA buffer allocated is 48K 32-bit samples (¾ the FIFO size of 64K samples) or 192K bytes. By default, when the driver is loaded, 10 DMA buffers are allocated for each board that is present in the system. This number can be changed at driver load time by editing the *ccurdscc_config* file located in the driver installation directory and re-installing the driver *(make load)*. The driver may fail to allocate buffers if the count is very large and DMA buffers are not available in the system. Basically, the only reason to increase this number is if the application has periods during a run where it takes time to read the next buffer. In that case, the driver is queuing data into the allocated buffers to be used by the application at a later time. If the application fails to read the data prior to the driver exhausting the allocated buffers, then an overflow condition will be reported.

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

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

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

IOCTL_CCURDSCC_GET_DRIVER_INFO: The argument supplied to this *ioctl* is a pointer to the *ccurdscc_ ccurdscc_driver_info_t* structure. Information on the structure is located in the *ccurdscc_user.h* include file. This *ioctl* provides useful driver information.

IOCTL_CCURDSCC_GET_NUM_DMA_BUFFERS: The argument is a pointer to an *unsigned short*. This call returns the number of DMA buffers that have been allocated by the driver.

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

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

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

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

IOCTL_CCURDSCC_MMAP_SELECT: The argument to this *ioctl* is a pointer to the *ccurdscc_mmap_select_t* structure. Information on the structure is located in the *ccurdscc_user.h* include file. This call needs to be made prior to the *mmap(2)* system call so as to direct the *mmap(2)* call to perform the requested mapping specified by this *ioctl*. The three possible mappings that are performed by the driver are to *mmap* the local register space *(CCURDSCC_SELECT_LOCAL_MMAP),* the configuration register space *(CCURDSCC_SELECT_CONFIG_MMAP)* and a physical memory *(CCURDSCC_SELECT_PHYS_MEM_MMAP)* that is created by the the *mmap(2)* system call.

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

IOCTL_CCURDSCC_PRESERVE_LIB_INFO: The argument to this *ioctl* is a pointer to the *_ccurdscc_preserve_t* structure. Information on the structure is located in the *ccurdscc_user.h* include file. This call is specifically used by the API to control its initialization and should not be used by the user.

IOCTL_CCURDSCC_READ_EEPROM: The argument to this *ioctl* is a pointer to the *ccurdscc_eeprom_t* structure. Information on the structure is located in the *ccurdscc_user.h* include file. This call is specifically used by the supplied *eeprom* application and should not be used by the user.

IOCTL_CCURDSCC_REMOVE_DMA_BUFFERS: This *ioctl* does not have any arguments. The purpose of this call is to remove the previously allocated DMA buffers. Once the DMA buffers are freed, the user will be unable to perform reads in the *CCURDSCC_DMA_CONTINUOUS* mode until DMA buffers have been reallocated with the *IOCTL_CCURDSCC_ALLOCATE_DMA_BUFFERS* call.

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

IOCTL_CCURDSCC_RESET_BOARD: This *ioctl* does not have any arguments. This call resets the board to a known initial default state. Additionally, the Converters, Clocks and FIFO are reset along with internal pointers and clearing of interrupts. This call is currently identical to the *IOCTL_CCURDSCC_INIT_BOARD* call.

IOCTL_CCURDSCC_RESET_DMA_CONTINUOUS_BUFFERS: This *ioctl* does not have any arguments. The DMA pointers are managed internally by the driver and the library. This call resets the pointers and should not normally be called by the user.

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

IOCTL_CCURDSCC_WAIT_FOR_INTERRUPT: The argument to this *ioctl* is a pointer to the *ccurdscc_driver_int_t* structure. Information on the structure is located in the *ccurdscc_user.h* include file. The user can wait for either a FIFO low to high transition interrupt or a DMA complete interrupt. If a time out value greater than zero is specified, the call will time out after the specified seconds, otherwise it will not time out.

IOCTL_CCURDSCC_WRITE_EEPROM: The argument to this *ioctl* is a pointer to the *ccurdscc_eeprom_t* structure. Information on the structure is located in the *ccurdscc_user.h* include file. This call is specifically used by the supplied *eeprom* application and should not be used by the user.

2.1.3 mmap(2) system call

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

```
int *munmap_local_ptr;
ccurdscc_local_ctrl_data_t  *local_ptr;
ccurdscc_mmap_select_t  mmap_select;
unsigned long mmap_local_size;

mmap_select.select = CCURDSCC_SELECT_LOCAL_MMAP;
mmap_select.offset=0;
mmap_select.size=0;

ioctl(fp, IOCTL_CCURDSCC_MMAP_SELECT,(void *)&mmap_select);
```

```
mmap_local_size = mmap_select.size;


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

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

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

2.1.4 read(2) system call

Prior to issuing this call to read the FIFO, the user needs to select the type of read operation they would like to perform. The only reason for providing various read modes is because the board allows it and that it gives the user the ability to choose the optimal mode for their particular application. The read mode is specified by the *ioctl* call with the *IOCTL_CCURDSCC_SELECT_READ_MODE* command. The following are the possible read modes:

CCURDSCC_PIO_CHANNEL: This mode returns the data from the latest converted channels from 1 to 32 channels. The relative offset within the returned buffer determines the channel number. The data content is a 24-bit analog input raw value. The driver uses Programmed I/O to perform this operation. In this mode, samples read are the latest samples that are being continuously converted by the hardware.

CCURDSCC_PIO_FIFO: This mode returns 32-bit data values from FIFO using Programmed I/O operation. Each 32-bit data value read contains a 24-bit channel data in the low three bytes of the word, while the most significant byte contains the channel number. The FIFO can contain any channels in any order. This is dependent on the channel mask used and the clock speed specified for the particular converter. If the user stops issuing reads and causes the FIFO to fill, a FIFO overflow error would result.

CCURDSCC_DMA_CHANNEL: This mode of operation is identical to the *CCURDSCC_PIO_CHANNEL* mode with the exception that the driver performs a DMA operation instead of Programmed I/O to complete the operation. In this mode, samples read are the latest samples that are being continuously converted by the hardware. Normally, this is the preferred of the two modes as it takes less processing time and is faster.

CCURDSCC_DMA_FIFO: This mode is identical to the *CCURDSCC_PIO_FIFO* mode with the exception that the driver performs a DMA operation instead of Programmed I/O to complete the operation. Normally, this is the preferred of the two modes as it takes less processing time and is faster.

CCURDSCC_DMA_CONTINUOUS: This mode is similar to the *CCURDSCC_DMA_FIFO* with the exception that when the first read is issued, the driver will automatically fill internal DMA buffers with data as long as DMA buffers are available. This allows applications that have delays between reads to buffer the data without any loss, until of course the system runs out of allocated buffers at which point, a FIFO overflow error would result.

2.2 Application Program Interface (API) Access

The API is the recommended method of communicating with the board for most users. The following are a list of calls that are available.

```
ccurDSCC_Abort_DMA()
ccurDSCC_Add_Irq()
ccurDSCC_Allocate_DMA_Continuous_Buffers()
ccurDSCC_Clear_Driver_Error()
ccurDSCC_Clear_Lib_Error()
ccurDSCC_Close()
ccurDSCC_Compute_PLL_Clock()
ccurDSCC_Configure_Channels()
ccurDSCC_Configure_Channels_Info()
ccurDSCC_Data_To_Volts()
ccurDSCC_Disable_Pci_Interrupts()
ccurDSCC_Enable_Pci_Interrupts()
ccurDSCC_Fraction_To_Hex()
ccurDSCC_Get_Board_CSR()
ccurDSCC_Get_Board_Info()
ccurDSCC_Get_Converter_Cal_CSR()
ccurDSCC_Get_Converter_CSR()
ccurDSCC_Get_Converter_Info()
ccurDSCC_Get_Converter_Negative_Cal()
ccurDSCC_Get_Converter_Offset_Cal()
ccurDSCC_Get_Converter_Positive_Cal()
ccurDSCC_Get_Driver_Error()
ccurDSCC_Get_Driver_Info()
ccurDSCC_Get_Driver_Read_Mode()
ccurDSCC_Get_Fifo_Channel_Select()
ccurDSCC_Get_Fifo_Info()
ccurDSCC_Get_Interrupt_Control()
ccurDSCC_Get_Interrupt_Status()
ccurDSCC_Get_Interrupt_Timeout_Seconds()
ccurDSCC_Get_Lib_Error()
ccurDSCC_Get_Mapped_Config_Ptr()
ccurDSCC_Get_Mapped_Local_Ptr()
ccurDSCC_Get_Num_DMA_Continuous_Buffers()
ccurDSCC_Get_Open_File_Descriptor()
ccurDSCC_Get_Physical_Memory()
ccurDSCC_Get_PLL_Info()
ccurDSCC_Get_PLL_Status()
ccurDSCC_Get_PLL_Sync()
ccurDSCC_Get_Value()
ccurDSCC_Hex_To_Fraction()
ccurDSCC_Initialize_Board()
ccurDSCC_Initialize_PLL_Input_Struct()
ccurDSCC_MMap_Physical_Memory()
ccurDSCC_Munmap_Physical_Memory()
ccurDSCC_Open()
ccurDSCC_Perform_Auto_Calibration()
ccurDSCC_Perform_External_Input_Negative_Calibration()
ccurDSCC_Perform_External_Input_Offset_Calibration()
ccurDSCC_Perform_External_Input_Positive_Calibration()
ccurDSCC_Perform_Negative_Calibration()
ccurDSCC_Perform_Offset_Calibration()
```

```
ccurDSCC_Perform_Positive_Calibration()
ccurDSCC_Program_CPM_Advanced()
ccurDSCC_Program_PLL_Advanced()
ccurDSCC_Program_PLL_Clock()
ccurDSCC_Read()
ccurDSCC_Read_Channels()
ccurDSCC_Read_Channels_Calibration()
ccurDSCC_Remove_DMA_Continuous_Buffers()
ccurDSCC_Remove_Irq()
ccurDSCC_Reset_Board()
ccurDSCC_Reset_Converter()
ccurDSCC_Reset_DMA_Continuous_Buffers()
ccurDSCC_Reset_Fifo()
ccurDSCC_Select_Driver_Read_Mode()
ccurDSCC_Set_Board_CSR()
ccurDSCC_Set_Converter_Cal_CSR()
ccurDSCC_Set_Converter_Clock_Source()
ccurDSCC_Set_Converter_Negative_Cal()
ccurDSCC_Set_Converter_Offset_Cal()
ccurDSCC_Set_Converter_Positive_Cal()
ccurDSCC_Set_Fifo_Channel_Select()
ccurDSCC_Set_Fifo_Threshold()
ccurDSCC_Set_Interrupt_Control()
ccurDSCC_Set_Interrupt_Status()
ccurDSCC_Set_Interrupt_Timeout_Seconds()
ccurDSCC_Set_PLL_Sync()
ccurDSCC_Set_Value()
ccurDSCC_Shutdown_PLL_Clock()
ccurDSCC_Start_PLL_Clock()
ccurDSCC_Stop_PLL_Clock()
ccurDSCC_Volts_To_Data()
ccurDSCC_Wait_For_Interrupt()
ccurDSCC_Write()
ccurDSCC_Write_Channels_Calibration()
```

2.2.1 ccurDSCC_Abort_DMA()

This call will abort any DMA operation that is in progress. On-board input FIFO is reset and so are DMA CONTINUOUS mode pointers. Normally, the user should not use this call unless they are providing their own DMA handling.

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

   Description: Abort any DMA in progress

   Input:       void *Handle                    (handle pointer)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_NO_LOCAL_REGION    (error)
                CCURDSCC_LIB_IOCTL_FAILED       (error)
 ************************************************************************/
```

2.2.2 ccurDSCC_Add_Irq()

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

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

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

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_IOCTL_FAILED       (driver ioctl call failed)
 ************************************************************************/
```

2.2.3 ccurDSCC_Allocate_DMA_Continuous_Buffers()

This call creates DMA buffers that are to be used during reads, when operating in the *CCURDSCC_DMA_CONTINUOUS* mode. If the buffer count is 0, no buffers are allocated and the user will be unable to perform reads using the *CCURDSCC_DMA_CONTINUOUS* mode. Each DMA buffer allocated is 48K 32-bit samples (¾ the FIFO size of 64K samples) or 192K bytes. By default, when the driver is loaded, 10 DMA buffers are allocated for each board that is present in the system. This number can be changed at driver load time by editing the *ccurdscc_config* file located in the driver installation directory and re-installing the driver *(make load)*. The driver may fail to allocate buffers if the count is very large and DMA buffers are not available in the system. Basically, the only reason to increase this number is if the application has periods during a run where it takes time to read the next buffer. In that case, the driver is queuing data into the allocated buffers to be used by the application at a later time. If the application fails to read the data prior to the driver exhausting the allocated buffers, then an overflow condition will be reported.

```
/*****************************************************************************
   int ccurDSCC_Allocate_DMA_Continuous_Buffers(void *Handle, ushort nbufs)

   Description: Allocate DMA Continuous Buffers

   Input:       void *Handle                    (handle pointer)
                ushort                nbufs     (number of buffers)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_IOCTL_FAILED       (driver ioctl call failed)
 *****************************************************************************
```

2.2.4 ccurDSCC_Clear_Driver_Error()

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

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

   Description: Clear any previously generated driver related error.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_IOCTL_FAILED       (driver ioctl call failed)
 *****************************************************************************/
```

2.2.5 ccurDSCC_Clear_Lib_Error()

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

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

   Description: Clear any previously generated library related error.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
 *****************************************************************************
```

2.2.6 ccurDSCC_Close()

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

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

   Description: Close a previously opened device.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
```

```
                CCURDSCC_LIB_NOT_OPEN           (device not open)
 ***************************************************************************/
```

2.2.7 ccurDSCC_Compute_PLL_Clock()

This call is supplied for advanced users who wish to understand the parameters involved in programming a PLL clock based on a set of requirements. No actual board programming is performed with this call. The call simply accepts a set of inputs and computes the parameters needed to program a particular PLL for the given inputs. Refer to the *ccurdscc_pll.c* file located in the *.../test/lib* directory for usage of this call. Refer to the *.../lib/ccurdscc_lib.h* include file for structure definitions.

```
/***************************************************************************
   int ccurDSCC_Compute_PLL_Clock(void *Handle, ccurdscc_PLL_setting_t *input,
                                  ccurdscc_solution_t *solution)

   Description: Return the value of the specified PLL information.

   Input:       void                    *Handle    (handle pointer)
                ccurdscc_PLL_setting_t *input     (pll input setting)
   Output:      ccurdscc_solution_t    *solution; (pointer to solution struct)
   Return:      CCURDSCC_LIB_NO_ERROR             (successful)
                CCURDSCC_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN             (device not open)
                CCURDSCC_LIB_INVALID_ARG          (invalid argument)
****************************************************************************
```

Following is the information supplied to the call:

```
typedef struct {
    double  fDesired;           /* MHz - Desired Output Clock Frequency */
    int     max_tol;            /* ppm - parts/million - Maximum tolerance */
    int     maximizeVCOspeed;   /* Maximize VCO Speed flag */
    double  fRef;               /* MHz - Reference Input PLL Oscillator
                                                          Frequency */
    double  fPFDmin;            /* MHz - Minimum allowable Freq at phase-
                                                          detector */
    double  kfVCO;              /* MHz/Volts - VCO gain to be used */
    double  fVcoMin;            /* MHz - Minimum VCO frequency */
    double  fVcoMax;            /* MHz - Maximum VCO frequency */
    double  nRefMin;            /* minimum reference divider */
    double  nRefMax;            /* maximum reference divider */
    double  nFbkMin;            /* minimum feedback divider */
    double  nFbkMax;            /* maximum feedback divider */
} ccurdscc_PLL_setting_t;
```

Refer to the *ccurDSCC_Get_PLL_Info()* call for information on the *ccurdscc_PLL_struct_t* structure. Returned solution for the input is under:

```
typedef struct {
    int product;
    int post_divider1;
    int post_divider2;
    int post_divider3;
} ccurdscc_postDividerData_t;

typedef struct {
    int                         NREF;
    int                         NFBK;
    ccurdscc_postDividerData_t  NPOST;
    double                      synthErr;
    double                      fVCO;
```

```
    double                      ClkFreq;
    int                         tol_found;
    double                      gain_margin;
    uint                        charge_pump_current;
    uint                        loop_resistor;
    uint                        loop_capacitor;
    ccurdscc_PLL_struct_t       setup;
} ccurdscc_solution_t;
```

2.2.8 ccurDSCC_Configure_Channels()

This board is divided into four channel groups. Each channel group can be associated with an individual PLL clock. There are four independent PLL clocks available in this board. The user can program all four channel groups to be driven by a single PLL clock or conversely, have each channel group connected to its own PLL clock operating at different sampling rates. This is the main API that allows a user to program a channel group and associate with a PLL clock. The API internals takes care of determining the closest clock frequency and programming the PLL and associating with a PLL based on user request. Prior to completion, this call checks to see if there are any active PLLs that are no longer being used by any converters and if so, it shuts them down to reduce any noise.

```
/*****************************************************************************
   int ccurDSCC_Configure_Channels(void *Handle,
                                   ccurdscc_configure_channels_t *cc)

   Description: Configure Channels

   Input:       void *Handle                       (handle pointer)
                ccurdscc_configure_channels_t *cc (pointer to config struct)
   Output:      ccurdscc_configure_channels_t *cc (pointer to config struct)
   Return:      CCURDSCC_LIB_NO_ERROR              (successful)
                CCURDSCC_LIB_BAD_HANDLE            (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN              (device not open)
                CCURDSCC_LIB_INVALID_ARG           (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION       (local region not present)
                CCURDSCC_LIB_NO_RESOURCE           (no free PLL available)
 *****************************************************************************/
```

The *ccurdscc_configure_channels_t* struct is used both as input and output arguments to this call.

```
typedef struct {
    uint    clock_select;           /* user supplied - clock selection */
    uint    channel_group;          /* user supplied - converter selection */
    double  sample_rate;            /* user supplied - samples/second */
    uint    high_pass_filter;       /* user supplied - high pass filter mask */
    double  actual_sample_rate;     /* returned - samples/second */
    uint    assigned_clock;         /* returned - selected clock */
    double  actual_clock_freq;      /* returned - actual clock frequency */
} ccurdscc_configure_channels_t;
```

clock_select: This argument requests a particular clock for the channel group. If a particular clock is already assigned with a different channel group, the call will fail if programming the clock is going to be different from its current programming. In short, the sample rate selected by the shared channel groups must be such that re-programming of the common PLL is not necessary. The user can always let this API select the clock by using the *CCURDSCC_CLOCK_AUTO_SELECT* argument instead of specifying the clock. In that case, this call will associate the requested channel group with either a PLL that is in use if no PLL programming is required or it will select a new PLL and dedicate to the selected channel group. Options to this argument are:

- CCURDSCC_CLOCK_PLL_0
- CCURDSCC_CLOCK_PLL_1

- CCURDSCC_CLOCK_PLL_2
- CCURDSCC_CLOCK_PLL_3
- CCURDSCC_CLOCK_EXTERNAL
- CCURDSCC_CLOCK_AUTO_SELECT

channel_group: This argument selects one of the following channel groups:

- CCURDSCC_CHANNELS_0_7
- CCURDSCC_CHANNELS_8_15
- CCURDSCC_CHANNELS_16_23
- CCURDSCC_CHANNELS_24_31

sample_rate: This argument selects the samples/second (SPS) programming for the channel group. The range of sample_rate is 2000 SPS to 216000 SPS. The call will make the best effort to program the board as close to this rate as possible. The actual sample rate that the board was programmed to will be returned in the *actual_sample_rate.*

high_pass_filter: This argument is used to enable or disable a high pass filter that exists for each channel. When a particular bit is set LOW in the filter register, the corresponding high pass filter is enabled. Mask values can be:

- CCURDSCC_CONVERTER_MASK_CH0
- CCURDSCC_CONVERTER_MASK_CH1
- CCURDSCC_CONVERTER_MASK_CH2
- CCURDSCC_CONVERTER_MASK_CH3
- CCURDSCC_CONVERTER_MASK_CH4
- CCURDSCC_CONVERTER_MASK_CH5
- CCURDSCC_CONVERTER_MASK_CH6
- CCURDSCC_CONVERTER_MASK_CH7
- CCURDSCC_CONVERTER_MASK_ALL

actual_sample_rate: This argument returns to the user the actual sample rate that the call was able to program the board to. This may be different from the requested sample rate and is restricted by the hardware. In most cases, the actual sample rate will be very close to the requested sample rate.

assigned_clock: This argument returns to the user the actual clock that has been assigned to the converter. It can be one of the following:

- CCURDSCC_CLOCK_PLL_0
- CCURDSCC_CLOCK_PLL_1
- CCURDSCC_CLOCK_PLL_2
- CCURDSCC_CLOCK_PLL_3
- CCURDSCC_CLOCK_EXTERNAL

actual_clock_frequency: This argument returns to the user the actual clock frequency that the board PLL was programmed to. The clock frequency can range from 512 KHz to 13.824 MHz.

2.2.9 ccurDSCC_Configure_Channels_Info()

This call provides some useful information about actual PLL frequency and which converters are connected to which PLL. If the *print* argument is set to *CCURDSCC_TRUE,* the information will be printed.

```
/*****************************************************************************
   int ccurDSCC_Configure_Channels_Info(void *Handle,
                                        _ccurdscc_preserve_t *info, int print)

   Description: Return Configured Channel Info in preserved structure

   Input:       void *Handle                    (handle pointer)
                int print                       (print flag)
   Output:      _ccurdscc_preserve_t *info      (pointer to preserve struct)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
                CCURDSCC_LIB_NO_RESOURCE        (no free PLL available)
 *****************************************************************************

typedef struct {
    double  actual_freq;
    uint    converter_multiplier[CCURDSCC_MAX_CONVERTERS];
} _programmed_pll_t;

typedef struct {
    int                 action;        /* read(0)/write(1) preserve action */
    double              last_specified_fRef;
    _programmed_pll_t  programmed_PLL[CCURDSCC_PLL_MAX_WITH_EXTERNAL];
                                       /* +1 for external clock */
} _ccurdscc_preserve_t;
```

2.2.10 ccurDSCC_Data_To_Volts()

This routine takes a raw analog input data value and converts it to a floating point voltage based on the supplied format. Format can be *CCURDSCC_TWOS_COMPLEMENT* or *CCURDSCC_OFFSET_BINARY*.

```
/*****************************************************************************
   double ccurDSCC_Data_To_Volts(void *Handle, int us_data, int format)

   Description: Convert Data to volts

   Input:       void     *Handle               (handle pointer)
                int      us_data               (data to convert)
                int      format                (conversion format)
   Output:      none
   Return:      double   volts                 (returned volts)
 *****************************************************************************/
```

2.2.11 ccurDSCC_Disable_Pci_Interrupts()

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

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

   Description: Disable interrupts being generated by the board.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
```

```
                CCURDSCC_LIB_IOCTL_FAILED       (driver ioctl call failed)
 ****************************************************************************/
```

2.2.12 ccurDSCC_Enable_Pci_Interrupts()

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

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

   Description: Enable interrupts being generated by the board.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_IOCTL_FAILED       (driver ioctl call failed)
 ****************************************************************************/
```

2.2.13 ccurDSCC_Fraction_To_Hex()

This call simply converts a floating point decimal fraction to a hexadecimal value. It is used internally by the library for setting negative and positive calibration.

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

   Description: Convert Fractional Decimal to Hexadecimal

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

2.2.14 ccurDSCC_Get_Board_CSR()

This call can be used to get the data and the external clock output settings.

```
/****************************************************************************
   int ccurDSCC_Get_Board_CSR(void *Handle, ccurdscc_board_csr_t *bcsr)

   Description: Get Board Control and Status information

   Input:       void *Handle                    (handle pointer)
   Output:      ccurdscc_board_csr_t   *bcsr    (pointer to board csr)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ****************************************************************************/

typedef struct {
    int     data_format;             /* data format selection */
    int     external_clock_output;   /* external clock selection */
```

```
} ccurdscc_board_csr_t;

// data_format
- CCURDSCC_OFFSET_BINARY
- CCURDSCC_TWOS_COMPLEMENT

//external_clock_output
- CCURDSCC_EXT_CLOCK_OUTPUT_PLL_0
- CCURDSCC_EXT_CLOCK_OUTPUT_PLL_1
- CCURDSCC_EXT_CLOCK_OUTPUT_PLL_2
- CCURDSCC_EXT_CLOCK_OUTPUT_PLL_3
- CCURDSCC_EXT_CLOCK_OUTPUT_INPUT_LINE
```

2.2.15 ccurDSCC_Get_Board_Info()

This call returns the board id, the board type and the firmware revision level for the selected board. This board id is *0x9277* and board type is *0x1*.

```
/*************************************************************************
   int ccurDSCC_Get_Board_Info(void *Handle, ccurdscc_board_info_t *binfo)
   Description: Get Board Information

   Input:       void *Handle                      (handle pointer)
   Output:      ccurdscc_board_info_t   *binfo    (pointer to board info)
   Return:      CCURDSCC_LIB_NO_ERROR             (successful)
                CCURDSCC_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN             (device not open)
                CCURDSCC_LIB_INVALID_ARG          (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION      (local region not present)
 *************************************************************************/

typedef struct {
    uint    board_id;
    uint    board_type;
    uint    firmware_rev;
    double  input_voltage_range;
    double  cal_ref_voltage;
} ccurdscc_board_info_t;
```

2.2.16 ccurDSCC_Get_Converter_Cal_CSR()

This call returns the current calibration voltage control register setting.

```
/*************************************************************************
   int ccurDSCC_Get_Converter_Cal_CSR(void *Handle,
                                      ccurdscc_converter_cal_csr_t *cal)

   Description: Get the Converter Calibration Voltage

   Input:       void *Handle                         (handle pointer)
   Output:      ccurdscc_converter_cal_csr_t *cal;   (pointer to cal csr struct)
   Return:      CCURDSCC_LIB_NO_ERROR                (successful)
                CCURDSCC_LIB_BAD_HANDLE              (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN                (device not open)
                CCURDSCC_LIB_INVALID_ARG             (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION         (local region not present)
 *************************************************************************/

typedef struct {
    uint    voltage_select;
} ccurdscc_converter_cal_csr_t;
```

Voltage Select is one of the following:

```
- CCURDSCC_CAL_VOLT_SEL_INPUT_SIGNAL          : Input Signal
- CCURDSCC_CAL_VOLT_SEL_GROUND                : Ground (All Converters)
- CCURDSCC_CAL_VOLT_SEL_PLUS_REFERENCE        : +Ref (All Converters) (+<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_MINUS_REFERENCE       : -Ref (All Converters) (-<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_00_07_GROUND          : Ground (Converter 0)
- CCURDSCC_CAL_VOLT_SEL_00_07_PLUS_REFERENCE  : +Ref (Converter 0) (+<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_00_07_MINUS_REFERENCE: -Ref (Converter 0) (-<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_08_15_GROUND          : Ground (Converter 1)
- CCURDSCC_CAL_VOLT_SEL_08_15_PLUS_REFERENCE  : +Ref (Converter 1) (+<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_08_15_MINUS_REFERENCE: -Ref (Converter 1) (-<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_16_23_GROUND          : Ground (Converter 2)
- CCURDSCC_CAL_VOLT_SEL_16_23_PLUS_REFERENCE  : +Ref (Converter 2) (+<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_16_23_MINUS_REFERENCE: -Ref (Converter 2) (-<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_24_31_GROUND          : Ground (Converter 3)
- CCURDSCC_CAL_VOLT_SEL_24_31_PLUS_REFERENCE  : +Ref (Converter 3) (+<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_24_31_MINUS_REFERENCE: -Ref (Converter 3) (-<ref> Volts)
```

2.2.17 ccurDSCC_Get_Converter_CSR()

This call returns control information on the selected converter.

```
/*************************************************************************
   int ccurDSCC_Get_Converter_CSR(void *Handle, CCURDSCC_CONVERTER conv,
                                  ccurdscc_converter_csr_t *ccsr)

   Description: Get Converter Control and Status information

   Input:       void *Handle                    (handle pointer)
                CCURDSCC_CONVERTER      conv    (selected converter)
   Output:      ccurdscc_board_csr_t   *ccsr    (pointer to converter csr)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 *************************************************************************/


// CCURDSCC_CONVERTER
- CCURDSCC_CONVERTER_0
- CCURDSCC_CONVERTER_1
- CCURDSCC_CONVERTER_2
- CCURDSCC_CONVERTER_3

typedef struct {
    uint    clock_source;
    uint    converter_reset;
    uint    converter_overflow;
    uint    converter_interface_busy;
} ccurdscc_converter_csr_t;

// clock_source
- CCURDSCC_CLOCK_PLL_0
- CCURDSCC_CLOCK_PLL_1
- CCURDSCC_CLOCK_PLL_2
- CCURDSCC_CLOCK_PLL_3
- CCURDSCC_CLOCK_EXTERNAL
// converter_reset
```

```
- CCURDSCC_CONVERTER_ACTIVE
- CCURDSCC_CONVERTER_ACTIVATE     (same as CCURDSCC_CONVERTER_ACTIVE)
- CCURDSCC_CONVERTER_RESET

// converter_overflow
- CCURDSCC_CONVERTER_NO_OVERFLOW
- CCURDSCC_CONVERTER_OVERFLOW

// converter_interface_busy
- CCURDSCC_CONVERTER_IDLE
- CCURDSCC_CONVERTER_BUSY
```

2.2.18 ccurDSCC_Get_Converter_Info()

This call returns the programmed information for the selected converter. If an error code of *CCURDSCC_LIB_CONVERTER_RESET* is returned, no converter information can be returned.

```
/*************************************************************************
   int ccurDSCC_Get_Converter_Info(void *Handle, CCURDSCC_CONVERTER conv,
                                   ccurdscc_CPM_struct_t *info)

   Description: Return the value of the specified Converter information.

   Input:      void                  *Handle (handle pointer)
               CCURDSCC_CONVERTER     conv   (converter selection)
   Output:     ccurdscc_CPM_struct_t *info;  (pointer to converter info struct)
   Return:     CCURDSCC_LIB_NO_ERROR         (successful)
               CCURDSCC_LIB_BAD_HANDLE       (no/bad handler supplied)
               CCURDSCC_LIB_NOT_OPEN         (device not open)
               CCURDSCC_LIB_INVALID_ARG      (invalid argument)
               CCURDSCC_LIB_NO_LOCAL_REGION  (local region not present)
               CCURDSCC_LIB_CONVERTER_RESET  (converter in reset state)
 *************************************************************************/

typedef struct {
    uint  chip_revision;         /* [3:0] */
    uint  chip_id;               /* [3:0] */

    uint  mode_select;           /* CCURDSCC_MODE_SELECT_SSM */
                                 /* CCURDSCC_MODE_SELECT_DSM */
                                 /* CCURDSCC_MODE_SELECT_QSM */

    uint  serial_format;         /* CCURDSCC_SERIAL_FORMAT_LEFT_JUSTIFIED */
                                 /* CCURDSCC_SERIAL_FORMAT_12S */
                                 /* CCURDSCC_SERIAL_FORMAT_TDM */

    uint  clock_divider;         /* CCURDSCC_CLOCK_DIVIDER_1 */
                                 /* CCURDSCC_CLOCK_DIVIDER_2 */
                                 /* CCURDSCC_CLOCK_DIVIDER_2a */
                                 /* CCURDSCC_CLOCK_DIVIDER_4 */
                                 /* CCURDSCC_CLOCK_DIVIDER_1_5 */
                                 /* CCURDSCC_CLOCK_DIVIDER_3 */
                                 /* CCURDSCC_CLOCK_DIVIDER_3a */

    uint  control_port_enable;   /* CCURDSCC_CONTROL_PORT_DISABLE */
                                 /* CCURDSCC_CONTROL_PORT_ENABLE */




    uint  overflow_status;       /* CCURDSCC_CONVERTER_MASK_CH0 */
```

```
                                /* CCURDSCC_CONVERTER_MASK_CH1 */
                                /* CCURDSCC_CONVERTER_MASK_CH2 */
                                /* CCURDSCC_CONVERTER_MASK_CH3 */
                                /* CCURDSCC_CONVERTER_MASK_CH4 */
                                /* CCURDSCC_CONVERTER_MASK_CH5 */
                                /* CCURDSCC_CONVERTER_MASK_CH6 */
                                /* CCURDSCC_CONVERTER_MASK_CH7 */

        uint  overflow_mask;    /* CCURDSCC_CONVERTER_MASK_CH0 */
                                /* CCURDSCC_CONVERTER_MASK_CH1 */
                                /* CCURDSCC_CONVERTER_MASK_CH2 */
                                /* CCURDSCC_CONVERTER_MASK_CH3 */
                                /* CCURDSCC_CONVERTER_MASK_CH4 */
                                /* CCURDSCC_CONVERTER_MASK_CH5 */
                                /* CCURDSCC_CONVERTER_MASK_CH6 */
                                /* CCURDSCC_CONVERTER_MASK_CH7 */

        uint  high_pass_filter; /* CCURDSCC_CONVERTER_MASK_CH0 */
                                /* CCURDSCC_CONVERTER_MASK_CH1 */
                                /* CCURDSCC_CONVERTER_MASK_CH2 */
                                /* CCURDSCC_CONVERTER_MASK_CH3 */
                                /* CCURDSCC_CONVERTER_MASK_CH4 */
                                /* CCURDSCC_CONVERTER_MASK_CH5 */
                                /* CCURDSCC_CONVERTER_MASK_CH6 */
                                /* CCURDSCC_CONVERTER_MASK_CH7 */

        uint  power_down;       /* CCURDSCC_POWER_DOWN_MASK_CH0_1 */
                                /* CCURDSCC_POWER_DOWN_MASK_CH2_3 */
                                /* CCURDSCC_POWER_DOWN_MASK_CH4_5 */
                                /* CCURDSCC_POWER_DOWN_MASK_CH6_7 */

        uint  power_down_oscillator; /* CCURDSCC_POWER_DOWN_OSCILLATOR_ENABLE */
                                /* CCURDSCC_POWER_DOWN_OSCILLATOR_DISABLE */

        uint  power_down_bandgap;    /* CCURDSCC_POWER_DOWN_BANDGAP_ENABLE */
                                     /* CCURDSCC_POWER_DOWN_BANDGAP_DISABLE */
        uint  mute_control;     /* CCURDSCC_CONVERTER_MASK_CH0 */
                                /* CCURDSCC_CONVERTER_MASK_CH1 */
                                /* CCURDSCC_CONVERTER_MASK_CH2 */
                                /* CCURDSCC_CONVERTER_MASK_CH3 */
                                /* CCURDSCC_CONVERTER_MASK_CH4 */
                                /* CCURDSCC_CONVERTER_MASK_CH5 */
                                /* CCURDSCC_CONVERTER_MASK_CH6 */
                                /* CCURDSCC_CONVERTER_MASK_CH7 */

        uint  serial_data;      /* CCURDSCC_SERIAL_DATA_MASK_CH0_1 */
                                /* CCURDSCC_SERIAL_DATA_MASK_CH2_3 */
                                /* CCURDSCC_SERIAL_DATA_MASK_CH4_5 */
                                /* CCURDSCC_SERIAL_DATA_MASK_CH6_7 */

} ccurdscc_CPM_struct_t;
```

2.2.19 ccurDSCC_Get_Converter_Negative_Cal()

This call returns the raw and floating point value of the negative calibration for each of the channels that is maintained by the card. This negative gain is automatically applied to the analog input data that is returned for each channel by the hardware. This calibration information is set using the *ccurDSCC_Set_Converter_Negative_Cal*() call.

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

```
   int ccurDSCC_Get_Converter_Negative_Cal(void *Handle,
                                           ccurdscc_converter_cal_t *cal)

   Description: Return the Converter Negative Calibration data.

   Input:       void                      *Handle   (handle pointer)
   Output:      ccurdscc_converter_cal_t  *cal      (pointer to board cal)
   Return:      CCURDSCC_LIB_NO_ERROR               (successful)
                CCURDSCC_LIB_BAD_HANDLE             (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN               (device not open)
                CCURDSCC_LIB_INVALID_ARG            (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION        (local region not present)
 ****************************************************************************/

typedef struct {
    uint    Raw[CCURDSCC_MAX_CHANNELS];
    double  Float[CCURDSCC_MAX_CHANNELS];
} ccurdscc_converter_cal_t;
```

2.2.20 ccurDSCC_Get_Converter_Offset_Cal()

This call returns the raw and floating point value of the offset calibration for each of the channels that is maintained by the card. This zero offset is automatically applied to the analog input data that is returned for each channel by the hardware. This calibration information is set using the *ccurDSCC_Set_Converter_Offset_Cal*() call.

```
/****************************************************************************
   int ccurDSCC_Get_Converter_Offset_Cal(void *Handle,
                                         ccurdscc_converter_cal_t *cal)

   Description: Return the Converter Positive Calibration data.

   Input:       void                       *Handle  (handle pointer)
   Output:      ccurdscc_converter_cal_t   *cal     (pointer to board cal)
   Return:      CCURDSCC_LIB_NO_ERROR               (successful)
                CCURDSCC_LIB_BAD_HANDLE             (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN               (device not open)
                CCURDSCC_LIB_INVALID_ARG            (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION        (local region not present)
 ****************************************************************************/

typedef struct {
    uint    Raw[CCURDSCC_MAX_CHANNELS];
    double  Float[CCURDSCC_MAX_CHANNELS];
} ccurdscc_converter_cal_t;
```

2.2.21 ccurDSCC_Get_Converter_Positive_Cal()

This call returns the raw and floating point value of the positive calibration for each of the channels that is maintained by the card. This positive gain is automatically applied to the analog input data that is returned for each channel by the hardware. This calibration information is set using the *ccurDSCC_Set_Converter_Positive_Cal*() call.

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

```
   int ccurDSCC_Get_Converter_Positive_Cal(void *Handle,
                                           ccurdscc_converter_cal_t *cal)

   Description: Return the Converter Positive Calibration data.

   Input:       void                       *Handle  (handle pointer)
   Output:      ccurdscc_converter_cal_t   *cal     (pointer to board cal)
   Return:      CCURDSCC_LIB_NO_ERROR               (successful)
                CCURDSCC_LIB_BAD_HANDLE             (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN               (device not open)
                CCURDSCC_LIB_INVALID_ARG            (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION        (local region not present)
 ****************************************************************************/

typedef struct {
    uint    Raw[CCURDSCC_MAX_CHANNELS];
    double  Float[CCURDSCC_MAX_CHANNELS];
} ccurdscc_converter_cal_t;
```

2.2.22 ccurDSCC_Get_Driver_Error()

This call returns the last error generated by the driver.

```
/****************************************************************************
   int ccurDSCC_Get_Driver_Error(void *Handle, ccurdscc_user_error_t *ret_err)

   Description: Get the last error generated by the driver.

   Input:       void *Handle                     (handle pointer)
   Output:      ccurdscc_user_error_t *ret_err   (error struct pointer)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
                CCURDSCC_LIB_IOCTL_FAILED        (driver ioctl call failed)
 ****************************************************************************/

#define CCURDSCC_ERROR_NAME_SIZE    64
#define CCURDSCC_ERROR_DESC_SIZE    128
typedef struct _ccurdscc_user_error_t {
    uint    error;                          /* error number */
    char    name[CCURDSCC_ERROR_NAME_SIZE];   /* error name used in driver */
    char    desc[CCURDSCC_ERROR_DESC_SIZE];   /* error description */
} ccurdscc_user_error_t;

enum    {
    CCURDSCC_SUCCESS = 0,
    CCURDSCC_INVALID_PARAMETER,
    CCURDSCC_FIFO_THRESHOLD_TIMEOUT,
    CCURDSCC_DMA_TIMEOUT,
    CCURDSCC_OPERATION_CANCELLED,
    CCURDSCC_RESOURCE_ALLOCATION_ERROR,
    CCURDSCC_INVALID_REQUEST,
    CCURDSCC_FAULT_ERROR,
    CCURDSCC_BUSY,
    CCURDSCC_ADDRESS_IN_USE,
    CCURDSCC_USER_INTERRUPT_TIMEOUT,
    CCURDSCC_DMA_INCOMPLETE,
};
```

2.2.23 ccurDSCC_Get_Driver_Info()

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

```
/*****************************************************************************
   int ccurDSCC_Get_Driver_Info(void *Handle,  ccurdscc_driver_info_t *info)

   Description: Get device information from driver.

   Input:       void *Handle                 (handle pointer)
   Output:      ccurdscc_driver_info_t *info (info struct pointer)
                -- char                   version[12]
                -- char                   built[32]
                -- char                   module_name[16]
                -- int                    board_index
                -- char                   board_desc[32]
                -- int                    bus
                -- int                    slot
                -- int                    func
                -- int                    vendor_id
                -- int                    sub_vendor_id
                -- int                    board_id
                -- int                    board_type
                -- int                    sub_device_id
                -- int                    board_info
                -- int                    msi_support
                -- int                    irqlevel
                -- int                    firmware
                -- double                 input_voltage_range
                -- double                 cal_ref_voltage;
                -- ccurdscc_driver_int_t  interrupt
                -- int                    Ccurdscc_Max_Region
                -- ccurdscc_dev_region_t  mem_region[CCURDSCC_MAX_REGION]

   Return:      CCURDSCC_LIB_NO_ERROR        (successful)
                CCURDSCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN        (device not open)
                CCURDSCC_LIB_INVALID_ARG     (invalid argument)
                CCURDSCC_LIB_IOCTL_FAILED    (driver ioctl call failed)
 *****************************************************************************/

typedef struct {
    unsigned long long  count;
    u_int               status;
    u_int               mask;
    int                 timeout_seconds;
} ccurdscc_driver_int_t;

typedef    struct
{
    uint   physical_address;
    uint   size;
    uint   flags;
    uint   *virtual_address;
} ccurdscc_dev_region_t;
#define CCURDSCC_MAX_REGION 32

typedef struct
{
    char                  version[12];     /* driver version */
    char                  built[32];       /* driver date built */
    char                  module_name[16]; /* driver name */
```

```
    int                     board_index;     /* board index */
    char                    board_desc[32];  /* board description */
    int                     bus;             /* bus number */
    int                     slot;            /* slot number */
    int                     func;            /* function number */
    int                     vendor_id;       /* vendor id */
    int                     sub_vendor_id;   /* sub-vendor id */
    int                     board_id;        /* board id */
    int                     board_type;      /* board type */
    int                     sub_device_id;   /* sub device id */
    int                     board_info;      /* board_info if applicable */
    int                     msi_support;     /* msi flag 1=MSI support, 0=NO MSI */
    int                     irqlevel;        /* IRQ level */
    int                     firmware;        /* firmware number if applicable */
    double                  input_voltage_range;/* board input voltage range */
    double                  cal_ref_voltage; /* calibration reference voltage */
    ccurdscc_driver_int_t interrupt;       /* interrupt information */
    int                     Ccurdscc_Max_Region; /*kernel DEVICE_COUNT_RESOURCE*/

    ccurdscc_dev_region_t mem_region[CCURDSCC_MAX_REGION];
} ccurdscc_driver_info_t;
```

2.2.24 ccurDSCC_Get_Driver_Read_Mode()

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

```
/*************************************************************************
   int ccurDSCC_Get_Driver_Read_Mode(void *Handle,
                                     CCURDSCC_DRIVER_READ_MODE *mode)

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

   Input:       void *Handle                       (handle pointer)
   Output:      CCURDSCC_DRIVER_READ_MODE *mode  (pointer to read mode)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION     (local region error)
                CCURDSCC_LIB_IOCTL_FAILED        (ioctl error)
 *************************************************************************/

typedef enum {
    CCURDSCC_PIO_CHANNEL,
    CCURDSCC_PIO_FIFO,
    CCURDSCC_DMA_CHANNEL,
    CCURDSCC_DMA_FIFO,
    CCURDSCC_DMA_CONTINUOUS,
}  CCURDSCC_DRIVER_READ_MODE;
```

2.2.25 ccurDSCC_Get_Fifo_Channel_Select()

The hardware is capable of selecting which active channels are to be monitored and converted data placed in the FIFO. This call returns the current channel selection mask. By default, all active channels are selected for storage into the FIFO. The mask has channel 0 as the least significant bit and channel 31 as the most significant bit.

```
/*****************************************************************************
   int ccurDSCC_Get_Fifo_Channel_Select(void *Handle, uint *fifo_chan_sel)

   Description: Get FIFO Channel Select Mask

   Input:       void *Handle                    (handle pointer)
   Output:      uint             *fifo_chan_sel (pointer to fifo chan select)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region error)
 *****************************************************************************/
```

2.2.26 ccurDSCC_Get_Fifo_Info()

This call provides additional information about the FIFO. The FIFO needs to be in the active state and at least one active channel to be selected before converted data can be placed in the FIFO.

```
/*****************************************************************************
   int ccurDSCC_Get_Fifo_Info(void *Handle, ccurdscc_fifo_info_t *fifo)

   Description: Get FIFO Control and Status information

   Input:       void *Handle                    (handle pointer)
   Output:      ccurdscc_board_csr_t   *fifo    (pointer to board csr)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region error)
 *****************************************************************************/

typedef struct {
    uint    reset;
    uint    overflow;
    uint    underflow;
    uint    full;
    uint    threshold_exceeded;
    uint    empty;
    uint    data_counter;
    uint    threshold;
} ccurdscc_fifo_info_t;

// reset
- CCURDSCC_FIFO_ACTIVE
- CCURDSCC_FIFO_ACTIVATE        (same as CCURDSCC_FIFO_ACTIVE)
- CCURDSCC_FIFO_RESET

// overflow
- CCURDSCC_FIFO_NO_OVERFLOW
- CCURDSCC_FIFO_OVERFLOW

// underflow
- CCURDSCC_FIFO_NO_UNDERFLOW
- CCURDSCC_FIFO_UNDERFLOW

// full
- CCURDSCC_FIFO_NOT_FULL
- CCURDSCC_FIFO_FULL
```

```
// threshold_exceeded
- CCURDSCC_FIFO_THRESHOLD_NOT_EXCEEDED
- CCURDSCC_FIFO_THRESHOLD_EXCEEDED

// empty
- CCURDSCC_FIFO_NOT_EMPTY
- CCURDSCC_FIFO_EMPTY

// data_counter
```
– this field ranges from 0 to 65536 entries representing the number of samples currently present in the FIFO.

```
// threshold
```
– this field ranges from 0 to 65536 entries representing the number of samples in the FIFO where the threshold interrupt should occur.

2.2.27 ccurDSCC_Get_Interrupt_Control()

This call displays the current state of the Interrupt Control Register.

```
/************************************************************************
   int ccurDSCC_Get_Interrupt_Control(void *Handle, ccurdscc_interrupt_t *intr)

   Description: Get Interrupt Control information

   Input:       void *Handle                    (handle pointer)
   Output:      ccurdscc_interrupt_t   *intr    (pointer to interrupt control)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ************************************************************************/

typedef struct {
    int     global_int;
    int     fifo_buffer_lo_hi_int;
    int     plx_local_int;
} ccurdscc_interrupt_t;

// global_int
- CCURDSCC_GLOBAL_INT_DISABLE
- CCURDSCC_GLOBAL_INT_ENABLE

// fifo_buffer_lo_hi_int
- CCURDSCC_FIFO_INT_LO_HI_DISABLE
- CCURDSCC_FIFO_INT_LO_HI_ENABLE

// plx_local_int
- CCURDSCC_PLX_LOCAL_INT_DISABLE
- CCURDSCC_PLX_LOCAL_INT_ENABLE
```

2.2.28 ccurDSCC_Get_Interrupt_Status()

This call displays the current state of the Interrupt Status Register.

```
/*****************************************************************************
   int ccurDSCC_Get_Interrupt_Status(void *Handle, ccurdscc_interrupt_t *intr)

   Description: Get Interrupt Status information

   Input:       void *Handle                     (handle pointer)
   Output:      ccurdscc_interrupt_t   *intr     (pointer to interrupt status)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION     (local region not present)
 *****************************************************************************/

typedef struct {
    int     global_int;
    int     fifo_buffer_lo_hi_int;
    int     plx_local_int;
} ccurdscc_interrupt_t;

// global_int
- not used

// fifo_buffer_lo_hi_int
- CCURDSCC_FIFO_INT_LO_HI_IGNORE
- CCURDSCC_FIFO_INT_LO_HI_RESET

// plx_local_int
- CCURDSCC_PLX_LOCAL_INT_IGNORE
- CCURDSCC_PLX_LOCAL_INT_RESET
```

2.2.29 ccurDSCC_Get_Interrupt_Timeout_Seconds()

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

```
/*****************************************************************************
   int ccurDSCC_Get_Interrupt_Timeout_Seconds(void *Handle,
                                              int *int_timeout_secs)

   Description: Get Interrupt Timeout Seconds

   Input:       void *Handle                     (handle pointer)
   Output:      int         *int_timeout_secs    (pointer to int tout secs)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION     (local region not present)
                CCURDSCC_LIB_IOCTL_FAILED        (ioctl error)
 *****************************************************************************/
```

2.2.30 ccurDSCC_Get_Lib_Error()

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

```
/*****************************************************************************
   int ccurDSCC_Get_Lib_Error(void *Handle, ccurdscc_lib_error_t *lib_error)

   Description: Get last error generated by the library.

   Input:       void *Handle                     (handle pointer)
   Output:      ccurdscc_lib_error_t *lib_error (error struct pointer)
                -- uint error                    (error number)
                -- char name[CCURDSCC_LIB_ERROR_NAME_SIZE] (error name)
                -- char desc[CCURDSCC_LIB_ERROR_DESC_SIZE] (error description)
                -- int  line_number             (error line number in lib)
                -- char function[CCURDSCC_LIB_ERROR_FUNC_SIZE]
                                                (library function in error)
   Return:      CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                Last Library Error
 *****************************************************************************/

typedef struct _ccurdscc_lib_error_t {
    uint    error;                              /* lib error number */
    char    name[CCURDSCC_LIB_ERROR_NAME_SIZE]; /* error name used in lib */
    char    desc[CCURDSCC_LIB_ERROR_DESC_SIZE]; /* error description */
    int     line_number;                        /* line number in library */
    char    function[CCURDSCC_LIB_ERROR_FUNC_SIZE];
                                               /* library function */
} ccurdscc_lib_error_t;
```

2.2.31 ccurDSCC_Get_Mapped_Config_Ptr()

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

```
/*****************************************************************************
   int ccurDSCC_Get_Mapped_Config_Ptr(void *Handle,
                                      ccurdscc_config_local_data_t **config_ptr)

   Description: Get mapped configuration pointer.

   Input:       void *Handle                     (handle pointer)
   Output:      ccurdscc_config_local_data_t **config_ptr (config struct ptr)
                -- structure in ccurdscc_user.h
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
                CCURDSCC_LIB_NO_CONFIG_REGION    (config region not present)
 *****************************************************************************/
```

2.2.32 ccurDSCC_Get_Mapped_Local_Ptr()

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

```
/************************************************************************
   int ccurDSCC_Get_Mapped_Local_Ptr(void *Handle,
                                     ccurdscc_local_ctrl_data_t **local_ptr)

   Description: Get mapped local pointer.

   Input:       void *Handle                      (handle pointer)
   Output:      ccurdscc_local_ctrl_data_t **local_ptr (local struct ptr)
                -- structure in ccurdscc_user.h
   Return:      CCURDSCC_LIB_NO_ERROR             (successful)
                CCURDSCC_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN             (device not open)
                CCURDSCC_LIB_INVALID_ARG          (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION      (local region not present)
 ************************************************************************/
```

2.2.33 ccurDSCC_Get_Num_DMA_Continuous_Buffers()

This call returns the number of DMA buffers that are being used by the driver when operating in the *CCURDSCC_DMA_CONTINUOUS* read mode.

```
/************************************************************************
   int ccurDSCC_Get_Num_DMA_Continuous_Buffers(void *Handle, ushort *nbufs)

   Description: Get Number of DMA Continuous Buffers

   Input:       void *Handle                      (handle pointer)
                ushort                 *nbufs     (pointer to number of buffers)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR             (successful)
                CCURDSCC_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN             (device not open)
                CCURDSCC_LIB_INVALID_ARG          (invalid argument)
 ************************************************************************/
```

2.2.34 ccurDSCC_Get_Open_File_Descriptor()

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

```
/*****************************************************************************
   int ccurDSCC_Get_Open_File_Descriptor(void *Handle, int *fd)

   Description: Get Open File Descriptor

   Input:       void *Handle                     (handle pointer)
   Output:      int                   *fd        (open file descriptor)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
 *****************************************************************************/
```

2.2.35 ccurDSCC_Get_Physical_Memory()

This call returns to the user the physical memory pointer and size that was previously allocated by the *ccurDSCC_Mmap_Physical_Memory()* call. The physical memory is allocated by the user when they wish to perform their own DMA and bypass the API. Once again, this call is only useful for advanced users.

```
/*****************************************************************************
   int ccurDSCC_Get_Physical_Memory(void *Handle,
                                    ccurdscc_phys_mem_t *phys_mem)

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

   Input:       void *Handle                     (handle pointer)
   Output:      ccurdscc_phys_mem_t *phys_mem    (mem struct pointer)
                -- void *phys_mem
                -- u_int phys_mem_size
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
                CCURDSCC_LIB_IOCTL_FAILED        (driver ioctl call failed)
 *****************************************************************************/

typedef struct {
    void            *phys_mem;      /* physical memory: physical address    */
    unsigned int    phys_mem_size;  /* physical memory: memory size - bytes */
} ccurdscc_phys_mem_t;
```

2.2.36 ccurDSCC_Get_PLL_Info()

This call returns the programmed information for the selected PLL.

```
/*****************************************************************************
   int ccurDSCC_Get_PLL_Info(void *Handle, CCURDSCC_PLL pll,
                             ccurdscc_PLL_struct_t *info)

   Description: Return the value of the specified PLL information.

   Input:       void                    *Handle  (handle pointer)
                CCURDSCC_PLL             pll     (pll selection)
   Output:      ccurdscc_PLL_struct_t  *info;   (pointer to pll info struct)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
 *****************************************************************************/
```

```
typedef struct {
    uint        ref_freq_divider;       /* [11:00] */

    uint        ref_freq_divider_src;   /* CCURDSCC_REF_DIVIDER_SRC_OSCILLATOR */
                                        /* CCURDSCC_REF_DIVIDER_SRC_PIN */

    uint        shutdown_1;             /* CCURDSCC_RUNNING */
                                        /* CCURDSCC_SHUTDOWN */

    uint        post_divider1;          /* CCURDSCC_POST_DIVIDER1_1 */
                                        /* CCURDSCC_POST_DIVIDER1_2 */
                                        /* CCURDSCC_POST_DIVIDER1_3 */
                                        /* CCURDSCC_POST_DIVIDER1_4 */
                                        /* CCURDSCC_POST_DIVIDER1_5 */
                                        /* CCURDSCC_POST_DIVIDER1_6 */
                                        /* CCURDSCC_POST_DIVIDER1_7 */
                                        /* CCURDSCC_POST_DIVIDER1_8 */
                                        /* CCURDSCC_POST_DIVIDER1_9 */
                                        /* CCURDSCC_POST_DIVIDER1_10*/
                                        /* CCURDSCC_POST_DIVIDER1_11 */
                                        /* CCURDSCC_POST_DIVIDER1_12 */

    uint        post_divider2;          /* CCURDSCC_POST_DIVIDER2_1 */
                                        /* CCURDSCC_POST_DIVIDER2_2 */
                                        /* CCURDSCC_POST_DIVIDER2_3 */
                                        /* CCURDSCC_POST_DIVIDER2_4 */
                                        /* CCURDSCC_POST_DIVIDER2_5 */
                                        /* CCURDSCC_POST_DIVIDER2_6 */
                                        /* CCURDSCC_POST_DIVIDER2_7 */
                                        /* CCURDSCC_POST_DIVIDER2_8 */
                                        /* CCURDSCC_POST_DIVIDER2_9 */
                                        /* CCURDSCC_POST_DIVIDER2_10*/
                                        /* CCURDSCC_POST_DIVIDER2_11 */
                                        /* CCURDSCC_POST_DIVIDER2_12 */

    uint        post_divider3;          /* CCURDSCC_POST_DIVIDER3_1 */
                                        /* CCURDSCC_POST_DIVIDER3_2 */
                                        /* CCURDSCC_POST_DIVIDER3_4 */
                                        /* CCURDSCC_POST_DIVIDER3_8 */

    uint        feedback_divider;       /* [13:00] */

    uint        feedback_divider_src;   /* CCURDSCC_FEEDBACK_DIVIDER_SRC_VCO */
                                        /* CCURDSCC_FEEDBACK_DIVIDER_SRC_POST */

    uint        clock_output;           /* CCURDSCC_CLOCK_OUTPUT_PECL */
                                        /* CCURDSCC_CLOCK_OUTPUT_CMOS */

    uint        charge_pump_current;    /* CCURDSCC_CHARGE_PUMP_CURRENT_2UA */
                                        /* CCURDSCC_CHARGE_PUMP_CURRENT_4_5UA */
                                        /* CCURDSCC_CHARGE_PUMP_CURRENT_11UA */
                                        /* CCURDSCC_CHARGE_PUMP_CURRENT_22_5UA */

    uint        loop_resistor;          /* CCURDSCC_LOOP_RESISTOR_400K */
                                        /* CCURDSCC_LOOP_RESISTOR_133K */
                                        /* CCURDSCC_LOOP_RESISTOR_30K */
                                        /* CCURDSCC_LOOP_RESISTOR_12K */

    uint        loop_capacitor;         /* CCURDSCC_LOOP_CAPACITOR_185PF */
                                        /* CCURDSCC_LOOP_CAPACITOR_500PF */

    uint        sync_enable;            /* CCURDSCC_SYNC_DISABLE */
                                        /* CCURDSCC_SYNC_ENABLE */
```

```
    uint        sync_polarity;          /* CCURDSCC_SYNC_POLARITY_NEGATIVE */
                                        /* CCURDSCC_SYNC_POLARITY_POSITIVE */

    uint        shutdown_2;             /* CCURDSCC_RUNNING */
                                        /* CCURDSCC_SHUTDOWN */

    /* below should not be supplied by user */
    double      last_specified_fRef;    /* Last Specified Reference Frequency */
    double      fActual;                /* Computed PLL Clock Frequency */
    uint        post_divider_product;   /* post divider product */
}  ccurdscc_PLL_struct_t;
```

2.2.37 ccurDSCC_Get_PLL_Status()

This call returns the status of the selected PLL.

```
/*************************************************************************
   int ccurDSCC_Get_PLL_Status(void *Handle, CCURDSCC_PLL pll,
                               ccurdscc_PLL_status_t *status)

   Description: Return the status of the PLL

   Input:       void                  *Handle   (handle pointer)
                CCURDSCC               pll      (select pll)
   Output:      ccurdscc_PLL_status_t *status;  (pointer to status struct)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 *************************************************************************/

typedef struct {
    uint    busy;
    uint    error;
} ccurdscc_PLL_status_t;


// PLL Interface Busy
- CCURDSCC_PLL_IDLE
- CCURDSCC_PLL_BUSY

// PLL Interface Error
- CCURDSCC_PLL_NO_ERROR
- CCURDSCC_PLL_ERROR
```

2.2.38 ccurDSCC_Get_PLL_Sync()

This call returns the PLL Synchronization information maintained by the hardware.

```
/*************************************************************************
   int ccurDSCC_Get_PLL_Sync(void *Handle, ccurdscc_PLL_sync_t *sync)

   Description: Return the value of the PLL Sync information.

   Input:       void              *Handle       (handle pointer)
   Output:      ccurdscc_PLL_sync_t  *sync;     (pointer to pll sync struct)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
```

```
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ***************************************************************************/

typedef struct {
    uint    sync_start[CCURDSCC_MAX_PLLS];
    uint    external_go;
    uint    external_sync;
} ccurdscc_PLL_sync_t;

// PLL Sync Start
- CCURDSCC_PLL_START
- CCURDSCC_PLL_STOP

// External Go
- CCURDSCC_EXTERNAL_GO_ENABLE
- CCURDSCC_EXTERNAL_GO_DISABLE

// External Sync
- CCURDSCC_EXTERNAL_SYNC_ENABLE
- CCURDSCC_EXTERNAL_SYNC_DISABLE
```

2.2.39 ccurDSCC_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. The *CCURDSCC_CHANNEL_DATA*, *CCURDSCC_POSITIVE_CALIBRATION*, *CCURDSCC_NEGATIVE_CALIBRATION* and the *CCURDSCC_OFFSET_CALIBRATION* return *CCURDSCC_MAX_CHANNELS* unsigned integers. The *CCURDSCC_SPI_RAM* command returns *CCURDSCC_SPI_RAM_SIZE* unsigned integers.

```
/***************************************************************************
   int ccurDSCC_Get_Value(void *Handle, CCURDSCC_CONTROL cmd, void *value)

   Description: Return the value of the specified board register.

   Input:       void              *Handle       (handle pointer)
                CCURDSCC_CONTROL  cmd           (register definition)
   Output:      void              *value;       (pointer to value)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ***************************************************************************/

typedef enum {
    CCURDSCC_BOARD_INFORMATION,             /* R Only */
    CCURDSCC_BOARD_CSR,                     /* R/W */

    CCURDSCC_INTERRUPT_CONTROL,             /* R/W */
    CCURDSCC_INTERRUPT_STATUS,              /* R/W */

    CCURDSCC_CONVERTER_0_CPM_CSR,           /* R/W */
    CCURDSCC_CONVERTER_0_CPM_ACCESS,        /* R/W */
    CCURDSCC_CONVERTER_0_CPM_READ_1,        /* R/W */
    CCURDSCC_CONVERTER_0_CPM_READ_2,        /* R Only */

    CCURDSCC_CONVERTER_1_CPM_CSR,           /* R/W */
    CCURDSCC_CONVERTER_1_CPM_ACCESS,        /* R/W */
    CCURDSCC_CONVERTER_1_CPM_READ_1,        /* R/W */
    CCURDSCC_CONVERTER_1_CPM_READ_2,        /* R Only */
```

```
    CCURDSCC_CONVERTER_2_CPM_CSR,           /* R/W */
    CCURDSCC_CONVERTER_2_CPM_ACCESS,        /* R/W */
    CCURDSCC_CONVERTER_2_CPM_READ_1,        /* R/W */
    CCURDSCC_CONVERTER_2_CPM_READ_2,        /* R Only */

    CCURDSCC_CONVERTER_3_CPM_CSR,           /* R/W */
    CCURDSCC_CONVERTER_3_CPM_ACCESS,        /* R/W */
    CCURDSCC_CONVERTER_3_CPM_READ_1,        /* R/W */
    CCURDSCC_CONVERTER_3_CPM_READ_2,        /* R Only */

    CCURDSCC_PLL_SYNC,                      /* R/W */

    CCURDSCC_CALIBRATION_VOLTAGE_CONTROL,   /* R/W */
    CCURDSCC_FIFO_CSR,                      /* R/W */
    CCURDSCC_FIFO_THRESHOLD,                /* R/W */
    CCURDSCC_FIFO_CHANNEL_SELECT,           /* R/W */

    CCURDSCC_PLL_0_STATUS,                  /* R Only */
    CCURDSCC_PLL_0_ACCESS,                  /* R/W */
    CCURDSCC_PLL_0_READ_1,                  /* R/W */
    CCURDSCC_PLL_0_READ_2,                  /* R Only */

    CCURDSCC_PLL_1_STATUS,                  /* R Only */
    CCURDSCC_PLL_1_ACCESS,                  /* R/W */
    CCURDSCC_PLL_1_READ_1,                  /* R/W */
    CCURDSCC_PLL_1_READ_2,                  /* R Only */

    CCURDSCC_PLL_2_STATUS,                  /* R Only */
    CCURDSCC_PLL_2_ACCESS,                  /* R/W */
    CCURDSCC_PLL_2_READ_1,                  /* R/W */
    CCURDSCC_PLL_2_READ_2,                  /* R Only */

    CCURDSCC_PLL_3_STATUS,                  /* R Only */
    CCURDSCC_PLL_3_ACCESS,                  /* R/W */
    CCURDSCC_PLL_3_READ_1,                  /* R/W */
    CCURDSCC_PLL_3_READ_2,                  /* R Only */

    CCURDSCC_FIRMWARE_SPI_COUNTER_STATUS,   /* R/W */
    CCURDSCC_CHANNEL_DATA,                  /* R Only */

    CCURDSCC_FIFO_DATA,                     /* R Only */

    CCURDSCC_POSITIVE_CALIBRATION,          /* R/W */
    CCURDSCC_NEGATIVE_CALIBRATION,          /* R/W */

    CCURDSCC_SPI_RAM,                       /* R/W */

    CCURDSCC_OFFSET_CALIBRATION,            /* R/W */
} CCURDSCC_CONTROL;
```

2.2.40 ccurDSCC_Hex_To_Fraction()

This call converts a hexadecimal value to a fractional decimal value. This conversion is used internally by the API to get the positive and negative calibration information.

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

   Description: Convert Hexadecimal to Fractional Decimal

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

2.2.41 ccurDSCC_Initialize_Board()

This call resets the board to a default initial state. This call is currently identical to the *ccurDSCC_Reset_Board()* call.

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

   Description: Initialize the board.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_IOCTL_FAILED       (driver ioctl call failed)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ************************************************************************/
```

2.2.42 ccurDSCC_Initialize_PLL_Input_Struct()

This call simply initializes the user supplied *ccurdscc_PLL_setting_t* clock structure to default values so that it can be used as input to the *ccurDSCC_Compute_PLL_Clock()* API call. This call is again only supplied for advanced users.

```
/************************************************************************
   int ccurDSCC_Initialize_PLL_Input_Struct(void *Handle,
                                            ccurdscc_PLL_setting_t *input)

   Description: Initialize the clock structure.

   Input:       void                   *Handle  (handle pointer)
                ccurdscc_PLL_setting_t *input   (pointer to input clock struct)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ************************************************************************/
```

```
typedef struct {
    double  fDesired;        /* MHz - Desired Output Clock Frequency */
    int     max_tol;         /* ppm - parts/million - Maximum tolerance */
    int     maximizeVCOspeed;/* Maximize VCO Speed flag */
    double  fRef;            /* MHz - Reference Input PLL Oscillator Frequency */
    double  fPFDmin;         /* MHz - Minimum allowable Freq at phase-detector */
    double  kfVCO;           /* MHz/Volts - VCO gain to be used */
    double  fVcoMin;         /* MHz - Minimum VCO frequency */
    double  fVcoMax;         /* MHz - Maximum VCO frequency */
    double  nRefMin;         /* minimum reference divider */
    double  nRefMax;         /* maximum reference divider */
    double  nFbkMin;         /* minimum feedback divider */
    double  nFbkMax;         /* maximum feedback divider */
} ccurdscc_PLL_setting_t;

- CCURDSCC_DEFAULT                     (-1)    /* Set defaults */
- CCURDSCC_DEFAULT_REFERENCE_FREQ      (65.536)/* MHz */
- CCURDSCC_DEFAULT_TOLERANCE           (1000)  /* ppm (parts per million) */
- CCURDSCC_DEFAULT_MIN_ALLOWABLE_FREQ (1.0)   /* MHz */
- CCURDSCC_DEFAULT_VCO_GAIN            (520)   /* MHz/volts */
- CCURDSCC_DEFAULT_MIN_VCO_FREQ        (100)   /* MHz */
- CCURDSCC_DEFAULT_MAX_VCO_FREQ        (400)   /* MHz */
- CCURDSCC_DEFAULT_MIN_REF_DIVIDER     (1)     /* minimum reference divider */
- CCURDSCC_DEFAULT_MAX_REF_DIVIDER     (4095)  /* maximum reference divider */
- CCURDSCC_DEFAULT_MIN_FEEDBK_DIVIDER (12)    /* minimum feedback divider */
- CCURDSCC_DEFAULT_MAX_FEEDBK_DIVIDER (16383) /* maximum feedback divider */

fRef             = CCURDSCC_DEFAULT_REFERENCE_FREQ;
maximizeVCOspeed = CCURDSCC_DEFAULT_VCO_SPEED;
fPFDmin          = CCURDSCC_DEFAULT_MIN_ALLOWABLE_FREQ;
max_tol          = CCURDSCC_DEFAULT_TOLERANCE;
kfVCO            = CCURDSCC_DEFAULT_VCO_GAIN;
fVcoMin          = CCURDSCC_DEFAULT_MIN_VCO_FREQ;
fVcoMax          = CCURDSCC_DEFAULT_MAX_VCO_FREQ;
nRefMin          = CCURDSCC_DEFAULT_MIN_REF_DIVIDER;
nRefMax          = CCURDSCC_DEFAULT_MAX_REF_DIVIDER;
nFbkMin          = CCURDSCC_DEFAULT_MIN_FEEDBK_DIVIDER;
nFbkMax          = CCURDSCC_DEFAULT_MAX_FEEDBK_DIVIDER;
fDesired         = CCURDSCC_DEFAULT;
```

2.2.43 ccurDSCC_MMap_Physical_Memory()

This call is provided for advanced users to create a physical memory of specified size that can be used for DMA. The allocated DMA memory is rounded to a page size. If a physical memory has been previously allocated, this call will fail, at which point the user will need to issue the *ccurDSCC_Munmap_Physical_Memory()* API call to remove the previously allocated physical memory.

Please note that this physical memory is not the same as that used internally by the driver during the *CCURDSCC_DMA_CONTINUOUS* read mode.

```
/*****************************************************************************
   int ccurDSCC_MMap_Physical_Memory(void *Handle, int size, void **mem_ptr)

   Description: Allocate a physical DMA memory for size bytes.

   Input:       void *Handle                    (handle pointer)
                int size                        (size in bytes)
   Output:      void **mem_ptr                  (mapped memory pointer)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_MMAP_SELECT_FAILED (mmap selection failed)
                CCURDSCC_LIB_MMAP_FAILED        (mmap failed)
 *****************************************************************************/
```

2.2.44 ccurDSCC_Munmap_Physical_Memory()

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

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

   Description: Unmap a previously mapped physical DMA memory.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_MUNMAP_FAILED      (failed to un-map memory)
                CCURDSCC_LIB_NOT_MAPPED         (memory not mapped)
 *****************************************************************************/
```

2.2.45 ccurDSCC_Open()

This is the first call that needs to be issued by a user to open a device and access the board through the rest of the API calls. What is returned is a handle to a *void pointer* that is supplied as an argument to the other API calls. The *Board_Number* is a valid board number [0..9] that is associated with a physical card. There must exist a character special file */dev/ccurdscc<Board_Number>* for the call to be successful. One character special file is created for each board found when the driver is successfully loaded.

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

```
/*****************************************************************************
   int ccurDSCC_Open(void **My_Handle, int Board_Number, int oflag)

   Description: Open a device.
   Input:       void **Handle                   (handle pointer to pointer)
                int Board_Number                (0-9 board number)
                int oflag                       (open flags)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_ALREADY_OPEN       (device already opened)
                CCURDSCC_LIB_OPEN_FAILED        (device open failed)
                CCURDSCC_LIB_ALREADY_MAPPED     (memory already mmapped)
                CCURDSCC_LIB_MMAP_SELECT_FAILED (mmap selection failed)
```

```
                CCURDSCC_LIB_MMAP_FAILED        (mmap failed)
 ***************************************************************************/
```

2.2.46 ccurDSCC_Perform_Auto_Calibration()

This call is used to create the offset, positive and negative gain values for all 32 channels. This offset and gain is then applied to each channel by the hardware when returning analog input values. Prior to issuing this call, the board must be initialized and clocks enabled and running, otherwise the call will fail as no analog input data is collected. The call performs calibration using an internal reference voltage whose value is determined by the board type selected.

This call takes approximately one minute to run and is normally issued after the system is rebooted and whenever the clocks are re-programmed to a different value. If the board has not been calibrated after a system reboot, then voltages returned will be unpredictable.

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

   Description: Perform Auto Calibration

   Input:       void   *Handle                  (handle pointer)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
                CCURDSCC_LIB_NO_RESOURCE        (no free PLL available)
                CCURDSCC_LIB_IO_ERROR           (read error)
 ***************************************************************************/
```

2.2.47 ccurDSCC_Perform_External_Input_Negative_Calibration()

This call is used to create the negative gain values for the user specified channels that have been connected to a precise voltage source. This gain is then applied to each channel by the hardware when returning analog input values. Prior to issuing this call, the board must be initialized and clocks enabled and running, otherwise the call will fail as no analog input data is collected. The external voltage supplied to the channels must be as close to the negative voltage whose value is defined by the board calibration reference voltage *(when external_ref_voltage == 0)* or specified by the user in *external_ref_voltage (non-zero negative value).*

This call is used when the user wishes to bypass the internal reference voltage for calibration and instead use their voltage source supplied to the external input.

It is important to note that prior to this call, the user must first perform the external offset calibration using the *ccurDSCC_Perform_External_Input_Offset_Calibration()* call, otherwise the calibrated values will be incorrect.

```
/***************************************************************************
   int ccurDSCC_Perform_External_Input_Negative_Calibration(void *Handle,
                           int chan_start, int chan_end,
                           double external_ref_voltage)

   Description: Perform External Input Negative Calibration

   Input:       void   *Handle                  (handle pointer)
                int    chan_start               (channel start number)
                int    chan_end                 (channel end number)
                double external_ref_voltage     (external reference voltage)
   Output:      none
```

```
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
                CCURDSCC_LIB_NO_RESOURCE        (no free PLL available)
                CCURDSCC_LIB_IO_ERROR           (read error)
 ************************************************************************/
```

2.2.48 ccurDSCC_Perform_External_Input_Offset_Calibration()

This call is used to create the offset values for the user specified channels that have been connected to a precise voltage source. This offset is then applied to each channel by the hardware when returning analog input values. Prior to issuing this call, the board must be initialized and clocks enabled and running, otherwise the call will fail as no analog input data is collected. The external voltage supplied to the channels must be close to zero volts.

This call is used when the user wishes to bypass the internal reference voltage for calibration and instead use their voltage source supplied to the external input.

```
/************************************************************************
   int ccurDSCC_Perform_External_Input_Offset_Calibration(void *Handle,
                           int chan_start, int chan_end)

   Description: Perform External Input Offset Calibration

   Input:       void   *Handle                  (handle pointer)
                int    chan_start               (channel start number)
                int    chan_end                 (channel end number)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
                CCURDSCC_LIB_NO_RESOURCE        (no free PLL available)
                CCURDSCC_LIB_IO_ERROR           (read error)
 ************************************************************************/
```

2.2.49 ccurDSCC_Perform_External_Input_Positive_Calibration()

This call is used to create the positive gain values for the user specified channels that have been connected to a precise voltage source. This gain is then applied to each channel by the hardware when returning analog input values. Prior to issuing this call, the board must be initialized and clocks enabled and running, otherwise the call will fail as no analog input data is collected. The external voltage supplied to the channels must be as close to the positive voltage whose value is defined by the board calibration reference voltage *(when external_ref_voltage == 0)* or specified by the user in *external_ref_voltage (non-zero positive value).*

This call is used when the user wishes to bypass the internal reference voltage for calibration and instead use their voltage source supplied to the external input.

It is important to note that prior to this call, the user must first perform the external offset calibration using the *ccurDSCC_Perform_External_Input_Offset_Calibration()* call, otherwise the calibrated values will be incorrect.

```
/*****************************************************************************
   int ccurDSCC_Perform_External_Input_Positive_Calibration(void *Handle,
                            int chan_start, int chan_end,
                            double external_ref_voltage)

   Description: Perform External Input Positive Calibration

   Input:       void   *Handle                  (handle pointer)
                int    chan_start               (channel start number)
                int    chan_end                 (channel end number)
                double external_ref_voltage     (external reference voltage)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
                CCURDSCC_LIB_NO_RESOURCE        (no free PLL available)
                CCURDSCC_LIB_IO_ERROR           (read error)
 *****************************************************************************/
```

2.2.50 ccurDSCC_Perform_Negative_Calibration()

This call is used to create the negative gain values for all 32 channels. This gain is then applied to each channel by the hardware when returning analog input values. Prior to issuing this call, the board must be initialized and clocks enabled and running, otherwise the call will fail as no analog input data is collected. The call performs calibration using an internal reference voltage whose value is determined by the board type selected.

It is important to note that prior to this call, the user must first perform the offset calibration using the *ccurDSCC_Perform_ Offset_Calibration()* call, otherwise the calibrated values will be incorrect.

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

   Description: Perform Negative Calibration

   Input:       void   *Handle                  (handle pointer)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
                CCURDSCC_LIB_NO_RESOURCE        (no free PLL available)
                CCURDSCC_LIB_IO_ERROR           (read error)
 *****************************************************************************/
```

2.2.51 ccurDSCC_Perform_Offset_Calibration()

This call is used to create the offset values for all 32 channels. This offset is then applied to each channel by the hardware when returning analog input values. Prior to issuing this call, the board must be initialized and clocks enabled and running, otherwise the call will fail as no analog input data is collected. The call performs calibration using a zero internal voltage.

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

   Description: Perform Offset Calibration

   Input:       void   *Handle                  (handle pointer)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
                CCURDSCC_LIB_NO_RESOURCE        (no free PLL available)
                CCURDSCC_LIB_IO_ERROR           (read error)
 ************************************************************************/
```

2.2.52 ccurDSCC_Perform_Positive_Calibration()

This call is used to create the positive gain values for all 32 channels. This gain is then applied to each channel by the hardware when returning analog input values. Prior to issuing this call, the board must be initialized and clocks enabled and running, otherwise the call will fail as no analog input data is collected. The call performs calibration using an internal reference voltage whose value is determined by the board type selected.

It is important to note that prior to this call, the user must first perform the offset calibration using the *ccurDSCC_Perform_ Offset_Calibration()* call, otherwise the calibrated values will be incorrect.

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

   Description: Perform Positive Calibration

   Input:       void   *Handle                  (handle pointer)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
                CCURDSCC_LIB_NO_RESOURCE        (no free PLL available)
                CCURDSCC_LIB_IO_ERROR           (read error)
 ************************************************************************/
```

2.2.53 ccurDSCC_Program_CPM_Advanced()

This call is available for use by advanced users to setup a specified converter. This call requires an intimate knowledge of the boards programming registers. Normally, the *ccurDSCC_Configure_Channels()* API call will be sufficient to program the board. If the converter is not in a *reset* state, the user can always issue the *ccurDSCC_Get_Converter_Info()* call to retrieve the current converter settings, and then edit specific options with this call. The user can also use the *CCURDSCC_DO_NOT_CHANGE* parameter for any argument value in the *ccurdscc_CPM_struct_t* structure if they wish to preserve the current values. Upon successful completion of the call, the board will be programmed to the new settings, and will return both the current settings and the new settings of all the CPM registers in the *ccurdscc_CPM_encode_t* structure.

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

```
   int ccurDSCC_Program_CPM_Advanced(void *Handle, CCURDSCC_CONVERTER conv,
                                     int Program,
                                     ccurdscc_CPM_struct_t *input,
                                     ccurdscc_CPM_encode_t *current_encoded,
                                     ccurdscc_CPM_encode_t *new_encoded)

   Description: Program CPM Access values for the specified CPM.

   Input:       void                   *Handle   (handle pointer)
                CCURDSCC_CPM            conv     (converter selection)
                ccurdscc_CPM_struct_t *input    (pointer to CPM input struct)
   Output:      int Program                      (decide to program board)
                ccurdscc_CPM_encode_t *current_encoded (pointer to current
                                                        encoded CPM
                ccurdscc_CPM_encode_t *new_encoded (pointer to new encoded CPM
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
 ***************************************************************************/

// CCURDSCC_CONVERTER
- CCURDSCC_CONVERTER_0
- CCURDSCC_CONVERTER_1
- CCURDSCC_CONVERTER_2
- CCURDSCC_CONVERTER_3

typedef struct {
    uint  chip_revision;         /* [3:0] */
    uint  chip_id;               /* [3:0] */

    uint  mode_select;           /* CCURDSCC_MODE_SELECT_SSM */
                                 /* CCURDSCC_MODE_SELECT_DSM */
                                 /* CCURDSCC_MODE_SELECT_QSM */

    uint  serial_format;         /* CCURDSCC_SERIAL_FORMAT_LEFT_JUSTIFIED */
                                 /* CCURDSCC_SERIAL_FORMAT_12S */
                                 /* CCURDSCC_SERIAL_FORMAT_TDM */

    uint  clock_divider;         /* CCURDSCC_CLOCK_DIVIDER_1 */
                                 /* CCURDSCC_CLOCK_DIVIDER_2 */
                                 /* CCURDSCC_CLOCK_DIVIDER_2a */
                                 /* CCURDSCC_CLOCK_DIVIDER_4 */
                                 /* CCURDSCC_CLOCK_DIVIDER_1_5 */
                                 /* CCURDSCC_CLOCK_DIVIDER_3 */
                                 /* CCURDSCC_CLOCK_DIVIDER_3a */

    uint  control_port_enable;   /* CCURDSCC_CONTROL_PORT_DISABLE */
                                 /* CCURDSCC_CONTROL_PORT_ENABLE */

    uint  overflow_status;       /* CCURDSCC_CONVERTER_MASK_CH0 */
                                 /* CCURDSCC_CONVERTER_MASK_CH1 */
                                 /* CCURDSCC_CONVERTER_MASK_CH2 */
                                 /* CCURDSCC_CONVERTER_MASK_CH3 */
                                 /* CCURDSCC_CONVERTER_MASK_CH4 */
                                 /* CCURDSCC_CONVERTER_MASK_CH5 */
                                 /* CCURDSCC_CONVERTER_MASK_CH6 */
                                 /* CCURDSCC_CONVERTER_MASK_CH7 */

    uint  overflow_mask;         /* CCURDSCC_CONVERTER_MASK_CH0 */
                                 /* CCURDSCC_CONVERTER_MASK_CH1 */
                                 /* CCURDSCC_CONVERTER_MASK_CH2 */
                                 /* CCURDSCC_CONVERTER_MASK_CH3 */
```

```
                                    /* CCURDSCC_CONVERTER_MASK_CH4 */
                                    /* CCURDSCC_CONVERTER_MASK_CH5 */
                                    /* CCURDSCC_CONVERTER_MASK_CH6 */
                                    /* CCURDSCC_CONVERTER_MASK_CH7 */

    uint  high_pass_filter;         /* CCURDSCC_CONVERTER_MASK_CH0 */
                                    /* CCURDSCC_CONVERTER_MASK_CH1 */
                                    /* CCURDSCC_CONVERTER_MASK_CH2 */
                                    /* CCURDSCC_CONVERTER_MASK_CH3 */
                                    /* CCURDSCC_CONVERTER_MASK_CH4 */
                                    /* CCURDSCC_CONVERTER_MASK_CH5 */
                                    /* CCURDSCC_CONVERTER_MASK_CH6 */
                                    /* CCURDSCC_CONVERTER_MASK_CH7 */

    uint  power_down;               /* CCURDSCC_POWER_DOWN_MASK_CH0_1 */
                                    /* CCURDSCC_POWER_DOWN_MASK_CH2_3 */
                                    /* CCURDSCC_POWER_DOWN_MASK_CH4_5 */
                                    /* CCURDSCC_POWER_DOWN_MASK_CH6_7 */

    uint  power_down_oscillator;    /* CCURDSCC_POWER_DOWN_OSCILLATOR_ENABLE */
                                    /* CCURDSCC_POWER_DOWN_OSCILLATOR_DISABLE */

    uint  power_down_bandgap;       /* CCURDSCC_POWER_DOWN_BANDGAP_ENABLE */
                                        /* CCURDSCC_POWER_DOWN_BANDGAP_DISABLE */
    uint  mute_control;             /* CCURDSCC_CONVERTER_MASK_CH0 */
                                    /* CCURDSCC_CONVERTER_MASK_CH1 */
                                    /* CCURDSCC_CONVERTER_MASK_CH2 */
                                    /* CCURDSCC_CONVERTER_MASK_CH3 */
                                    /* CCURDSCC_CONVERTER_MASK_CH4 */
                                    /* CCURDSCC_CONVERTER_MASK_CH5 */
                                    /* CCURDSCC_CONVERTER_MASK_CH6 */
                                    /* CCURDSCC_CONVERTER_MASK_CH7 */

    uint  serial_data;              /* CCURDSCC_SERIAL_DATA_MASK_CH0_1 */
                                    /* CCURDSCC_SERIAL_DATA_MASK_CH2_3 */
                                    /* CCURDSCC_SERIAL_DATA_MASK_CH4_5 */
                                    /* CCURDSCC_SERIAL_DATA_MASK_CH6_7 */

}  ccurdscc_CPM_struct_t;

typedef struct {
    uint   reg[CCURDSCC_CPM_AR_REGISTER_ADDRESS_MAX];
} ccurdscc_CPM_encode_t;
```

2.2.54 ccurDSCC_Program_PLL_Advanced()

This call is available for use by advanced users to setup a specified clock. This call requires an intimate knowledge of the boards programming registers. Normally, the *ccurDSCC_Configure_Channels()* API call will be sufficient to program the board. The user can always issue the *ccurDSCC_Get_PLL_Info()* call to retrieve the current clock settings, and then edit specific options with this call. The user can also use the *CCURDSCC_DO_NOT_CHANGE* parameter for any argument value in the *ccurdscc_PLL_struct_t* structure if they wish to preserve the current values. Upon successful completion of the call, the board will be programmed to the new settings, and will return both the current settings and the new settings of all the PLL registers in the *ccurdscc_PLL_encode_t* structure.

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

```
   int ccurDSCC_Program_PLL_Advanced(void *Handle, CCURDSCC_PLL pll,
                                     int Program,
                                    ccurdscc_PLL_struct_t *input,
                                    ccurdscc_PLL_encode_t *current_encoded,
                                    ccurdscc_PLL_encode_t *new_encoded)

   Description: Program PLL Access values for the specified PLL.

   Input:       void                  *Handle   (handle pointer)
                CCURDSCC_PLL          pll       (pll selection)
                ccurdscc_PLL_struct_t *input    (pointer to pll input struct)
   Output:      int Program                     (decide to program board)
                ccurdscc_PLL_encode_t *current_encoded (pointer to current
                                                        encoded PLL
                ccurdscc_PLL_encode_t *new_encoded (pointer to new encoded PLL
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
 ************************************************************************/

typedef struct {
    uint        ref_freq_divider;       /* [11:00] */

    uint        ref_freq_divider_src;   /* CCURDSCC_REF_DIVIDER_SRC_OSCILLATOR */
                                        /* CCURDSCC_REF_DIVIDER_SRC_PIN */

    uint        shutdown_1;             /* CCURDSCC_RUNNING */
                                        /* CCURDSCC_SHUTDOWN */

    uint        post_divider1;          /* CCURDSCC_POST_DIVIDER1_1 */
                                        /* CCURDSCC_POST_DIVIDER1_2 */
                                        /* CCURDSCC_POST_DIVIDER1_3 */
                                        /* CCURDSCC_POST_DIVIDER1_4 */
                                        /* CCURDSCC_POST_DIVIDER1_5 */
                                        /* CCURDSCC_POST_DIVIDER1_6 */
                                        /* CCURDSCC_POST_DIVIDER1_7 */
                                        /* CCURDSCC_POST_DIVIDER1_8 */
                                        /* CCURDSCC_POST_DIVIDER1_9 */
                                        /* CCURDSCC_POST_DIVIDER1_10*/
                                        /* CCURDSCC_POST_DIVIDER1_11 */
                                        /* CCURDSCC_POST_DIVIDER1_12 */

    uint        post_divider2;          /* CCURDSCC_POST_DIVIDER2_1 */
                                        /* CCURDSCC_POST_DIVIDER2_2 */
                                        /* CCURDSCC_POST_DIVIDER2_3 */
                                        /* CCURDSCC_POST_DIVIDER2_4 */
                                        /* CCURDSCC_POST_DIVIDER2_5 */
                                        /* CCURDSCC_POST_DIVIDER2_6 */
                                        /* CCURDSCC_POST_DIVIDER2_7 */
                                        /* CCURDSCC_POST_DIVIDER2_8 */
                                        /* CCURDSCC_POST_DIVIDER2_9 */
                                        /* CCURDSCC_POST_DIVIDER2_10*/
                                        /* CCURDSCC_POST_DIVIDER2_11 */
                                        /* CCURDSCC_POST_DIVIDER2_12 */

    uint        post_divider3;          /* CCURDSCC_POST_DIVIDER3_1 */
                                        /* CCURDSCC_POST_DIVIDER3_2 */
                                        /* CCURDSCC_POST_DIVIDER3_4 */
                                        /* CCURDSCC_POST_DIVIDER3_8 */

    uint        feedback_divider;       /* [13:00] */
    uint        feedback_divider_src;   /* CCURDSCC_FEEDBACK_DIVIDER_SRC_VCO */
```

```
                                        /* CCURDSCC_FEEDBACK_DIVIDER_SRC_POST */

    uint        clock_output;           /* CCURDSCC_CLOCK_OUTPUT_PECL */
                                        /* CCURDSCC_CLOCK_OUTPUT_CMOS */

    uint        charge_pump_current;    /* CCURDSCC_CHARGE_PUMP_CURRENT_2UA */
                                        /* CCURDSCC_CHARGE_PUMP_CURRENT_4_5UA */
                                        /* CCURDSCC_CHARGE_PUMP_CURRENT_11UA */
                                        /* CCURDSCC_CHARGE_PUMP_CURRENT_22_5UA */

    uint        loop_resistor;          /* CCURDSCC_LOOP_RESISTOR_400K */
                                        /* CCURDSCC_LOOP_RESISTOR_133K */
                                        /* CCURDSCC_LOOP_RESISTOR_30K */
                                        /* CCURDSCC_LOOP_RESISTOR_12K */

    uint        loop_capacitor;         /* CCURDSCC_LOOP_CAPACITOR_185PF */
                                        /* CCURDSCC_LOOP_CAPACITOR_500PF */

    uint        sync_enable;            /* CCURDSCC_SYNC_DISABLE */
                                        /* CCURDSCC_SYNC_ENABLE */

    uint        sync_polarity;          /* CCURDSCC_SYNC_POLARITY_NEGATIVE */
                                        /* CCURDSCC_SYNC_POLARITY_POSITIVE */

    uint        shutdown_2;             /* CCURDSCC_RUNNING */
                                        /* CCURDSCC_SHUTDOWN */

    /* below should not be supplied by user */
    double      last_specified_fRef;    /* Last Specified Reference Frequency */
    double      fActual;                /* Computed PLL Clock Frequency */
    uint        post_divider_product;   /* post divider product */
}  ccurdscc_PLL_struct_t;

typedef struct {
    uint   reg[CCURDSCC_PLL_AR_REGISTER_ADDRESS_MAX];
} ccurdscc_PLL_encode_t;
```

2.2.55 ccurDSCC_Program_PLL_Clock()

This call is available for use by advanced users to program a specified clock. This *ccurDSCC_Program_PLL_Clock()* call is a higher level call than the above *ccurDSCC_Program_PLL_Advanced()* call. In this case, the user only needs to supply the desired clock frequency *(that ranges from 512 KHz to 13.824 MHz)* and the maximum allowed tolerance in *ppm*. If the call is successful, it returns the actual clock frequency and the clock frequency error in *ppm*. If the *Program* flag is set to *CCURDSCC_TRUE*, the board is programmed with the new clock frequency at the completion of the call, otherwise only information on the actual frequency and the frequency error are returned to the user.

Normally, the advanced user needs to start with a sample rate and then determine the actual clock frequency that satisfies the sample rate. They then need to associate the clock with a selected channel group prior to starting data collection. All this is accomplished with the single API call *ccurDSCC_Configure_Channels()*.

```
/************************************************************************
   int ccurDSCC_Program_PLL_Clock(void *Handle, CCURDSCC_PLL pll, int Program,
                                  ccurdscc_PLL_clock_t *clock)

   Description: Program PLL Clock for give maximum tolerance

   Input:       void *Handle                     (handle pointer)
                CCURDSCC_PLL pll                 (selected PLL)
                int Program                      (decide to program board)
                ccurdscc_PLL_clock_t  *clock     (pointer to user clock struct)
   Output:      ccurdscc_PLL_clock_t  *clock     (pointer to user clock struct)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
                CCURDSCC_LIB_NO_SOLUTION_FOUND   (no solution found)
                CCURDSCC_LIB_NO_LOCAL_REGION     (local region not present)
 ************************************************************************/

typedef struct {
    double  fDesired;           /* MHz - Desired Output Clock Frequency */
    int     max_tol;            /* ppm - parts/million - Maximum tolerance */
    double  fActual;            /* MHz - Actual Output Clock Frequency */
    double  synthErr;           /* clock frequency error - ppm */
} ccurdscc_PLL_clock_t;
```

2.2.56 ccurDSCC_Read()

This call is provided for users to receive converted sample data from the channels. It basically calls the *read(2)* system call with the exception that it performs necessary *locking* and returns the *errno* returned from the system call in the pointer to the *error* variable.

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

```
/************************************************************************
   int ccurDSCC_Read(void *Handle, void *buf, int size, int *bytes_read,
                     int *error)

   Description: Perform a read operation.

   Input:       void *Handle                     (handle pointer)
                int  size                        (size of buffer in bytes)
   Output:      void *buf                        (pointer to buffer)
                int  *bytes_read                 (bytes read)
                int  *error                      (returned errno)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_IO_ERROR            (read failed)
                CCURDSCC_LIB_FIFO_OVERFLOW       (FIFO overflow)
 ************************************************************************/
```

2.2.57 ccurDSCC_Read_Channels()

This call performs a programmed I/O read of all the channels and returns the raw data in the *channel_data* field. Additionally, the user can request the corresponding voltage for each channel by setting the *convert_data_to_volts* to *CCURDSCC_TRUE*. In this case, the variable *volts* in the *ccrdscc_read_channels_t* structure will contain the floating point voltage of each channel.

This call is similar to the standard *read(2)* system call while operating in the *CCURDSCC_PIO_CHANNEL* mode with the exception that only raw data is returned.

```
/************************************************************************
   int ccurDSCC_Read_Channels(void *Handle, ccurdscc_read_channels_t *rdc)

   Description: Read Channel

   Input:       void *Handle                     (handle pointer)
                ccurdscc_read_channels_t *rdc    (perform_convertion)
   Output:      ccurdscc_read_channels_t *rdc    (pointer to rdc struct)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
 ************************************************************************/

typedef struct {
    uint    convert_data_to_volts;
    uint    channel_data[CCURDSCC_MAX_CHANNELS];
    double  volts[CCURDSCC_MAX_CHANNELS];
} ccurdscc_read_channels_t;
```

2.2.58 ccurDSCC_Read_Channels_Calibration()

This call reads the on-board channel calibration information and writes it out to a user specified output file. This file is created if it does not exist and must be writeable. If the output file argument is *NULL*, the calibration information is written to *stdout*. Entries in this file can be edited and use as input to the *ccurDSCC_Write_Channels_Calibration()* routine. Any blank lines or entries starting with '#' or '*' are ignored during parsing.

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

   int ccurDSCC_Read_Channels_Calibration(void *Handle, char *filename)

   Description: Read Channels Calibration information

   Input:       void *Handle                     (handle pointer)
   Output:      char *filename                   (pointer to filename)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_NO_LOCAL_REGION     (local region not present)
                CCURDSCC_LIB_CANNOT_OPEN_FILE    (file not readable)

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

Format:

```
#Chan   Negative  Offset    Positive
#====   ========  ======    ========
 ch00:  1.130771 -0.003152  1.130929
 ch01:  1.130661 -0.000795  1.130785
 ch02:  1.130400  0.001271  1.130840
 ----
 ch30:  1.130196  0.001695  1.130285
 ch31:  1.130440  0.001074  1.130285
```

2.2.59 ccurDSCC_Remove_DMA_Continuous_Buffers()

The purpose of this call is to remove the previously allocated DMA buffers. Once the DMA buffers are freed, the user will be unable to perform reads in the *CCURDSCC_DMA_CONTINUOUS* mode until DMA buffers have been reallocated with the *ccurDSCC_Allocate_DMA_Continuous_Buffers()* call.

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

   Description: Remove DMA Continuous Buffers

   Input:       void *Handle                    (handle pointer)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
 ************************************************************************/
```

2.2.60 ccurDSCC_Remove_Irq()

The purpose of this call is to remove the interrupt handler that was previously set up. The interrupt handler is managed internally by the driver and the library. The user should not issue this call, otherwise reads will time out.

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

   Description: By default, the driver sets up a shared IRQ interrupt handler
                when the device is opened. Now if for any reason, another
                device is sharing the same IRQ as this driver, the interrupt
                handler will also be entered every time the other shared
                device generates an interrupt. There are times that a user,
                for performance reasons may wish to run the board without
                interrupts enabled. In that case, they can issue this ioctl
                to remove the interrupt handling capability from the driver.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_IOCTL_FAILED       (driver ioctl call failed)
 ************************************************************************/
```

2.2.61 ccurDSCC_Reset_Board()

This call resets the board to a known initial default state. Additionally, the Converters, Clocks and FIFO are reset along with internal pointers and clearing of interrupts. This call is currently identical to the *ccurDSCC_Initialize_Board()* call.

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

   Description: Reset the board.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
```

```
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_IOCTL_FAILED       (driver ioctl call failed)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ****************************************************************************/
```

2.2.62 ccurDSCC_Reset_Converter()

This call performs a converter reset to the specified converter. No converter programming can be performed until the converter is activated. To activate the converter after a reset, set the *activate* argument to *CCURDSCC_CONVERTER_ACTIVATE*.

```
/****************************************************************************
   int ccurDSCC_Reset_Converter(void *Handle, CCURDSCC_CONVERTER conv,
                                int activate)

   Description: Reset Specified Converter

   Input:       void *Handle                    (handle pointer)
                CCURDSCC_CONVERTER         conv (selected converter)
                int activate                    (activate converter)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ****************************************************************************/

// CCURDSCC_CONVERTER
- CCURDSCC_CONVERTER_0
- CCURDSCC_CONVERTER_1
- CCURDSCC_CONVERTER_2
- CCURDSCC_CONVERTER_3
```

2.2.63 ccurDSCC_Reset_DMA_Continuous_Buffers()

The DMA pointers are managed internally by the driver and the library. This call resets the pointers and should not normally be called by the user.

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

   Description: Reset DMA Continuous Buffers

   Input:       void *Handle                    (handle pointer)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_NO_LOCAL_REGION    (error)
                CCURDSCC_LIB_IOCTL_FAILED       (error)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ****************************************************************************/
```

2.2.64 ccurDSCC_Reset_Fifo()

This call performs a FIFO reset. All data held in the FIFO is cleared and the FIFO is rendered empty. Additionally, internal pointers maintained for DMA CONTINUOUS mode are reset. No new data can be collected until the FIFO is activated. To activate the FIFO, set the *activate* argument to CCURDSCC_FIFO_ACTIVATE.

```
/*****************************************************************************
   int ccurDSCC_Reset_Fifo(void *Handle, int activate)

   Description: Reset Fifo

   Input:       void *Handle                    (handle pointer)
                int activate                    (activate FIFO)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 *****************************************************************************/
```

2.2.65 ccurDSCC_Select_Driver_Read_Mode()

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

```
/*****************************************************************************
   int ccurDSCC_Select_Driver_Read_Mode(void *Handle,
                                        CCURDSCC_DRIVER_READ_MODE mode)

   Description: Reset Fifo

   Input:       void *Handle                    (handle pointer)
                CCURDSCC_DRIVER_READ_MODE mode  (select read mode)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 *****************************************************************************/
```

```
typedef enum {
    CCURDSCC_PIO_CHANNEL,
    CCURDSCC_PIO_FIFO,
    CCURDSCC_DMA_CHANNEL,
    CCURDSCC_DMA_FIFO,
    CCURDSCC_DMA_CONTINUOUS,
}  CCURDSCC_DRIVER_READ_MODE;
```

2.2.66 ccurDSCC_Set_Board_CSR()

This call can be used to set the data format to *CCURDSCC_OFFSET_BINARY* or *CCURDSCC_TWOS_COMPLEMENT*. Additionally, this call can also be used to set the external clock output to one of the four PLL's or the Input Line (pass-through). This is useful when you are trying to connect multiple cards to a single clock source. Users can supply the *CCURDSCC_DO_NOT_CHANGE* parameter if they do not wish to alter the existing state of the card for a particular field.

```
/*****************************************************************************
   int ccurDSCC_Set_Board_CSR(void *Handle, ccurdscc_board_csr_t *bcsr)
```

```
   Description: Set Board Control and Status information

   Input:       void *Handle                      (handle pointer)
   Output:      ccurdscc_board_csr_t   *bcsr      (pointer to board csr)
   Return:      CCURDSCC_LIB_NO_ERROR             (successful)
                CCURDSCC_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN             (device not open)
                CCURDSCC_LIB_INVALID_ARG          (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION      (local region not present)
 ****************************************************************************/

typedef struct {
    int     data_format;             /* data format selection */
    int     external_clock_output;   /* external clock selection */
} ccurdscc_board_csr_t;

// data_format
- CCURDSCC_OFFSET_BINARY
- CCURDSCC_TWOS_COMPLEMENT
- CCURDSCC_DO_NOT_CHANGE

//external_clock_output
- CCURDSCC_EXT_CLOCK_OUTPUT_PLL_0
- CCURDSCC_EXT_CLOCK_OUTPUT_PLL_1
- CCURDSCC_EXT_CLOCK_OUTPUT_PLL_2
- CCURDSCC_EXT_CLOCK_OUTPUT_PLL_3
- CCURDSCC_EXT_CLOCK_OUTPUT_INPUT_LINE
- CCURDSCC_DO_NOT_CHANGE
```

2.2.67 ccurDSCC_Set_Converter_Cal_CSR()

This call sets the calibration voltage control register.

```
/****************************************************************************
   int ccurDSCC_Set_Converter_Cal_CSR(void *Handle,
                                      ccurdscc_converter_cal_csr_t *cal)

   Description: Set the Converter Calibration Voltage

   Input:       void *Handle                      (handle pointer)
                ccurdscc_converter_cal_csr_t *cal;  (pointer to cal struct)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR             (successful)
                CCURDSCC_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN             (device not open)
                CCURDSCC_LIB_INVALID_ARG          (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION      (local region not present)
 ****************************************************************************/

typedef struct {
    uint    voltage_select;
} ccurdscc_converter_cal_csr_t;
```

Voltage Select is one of the following:

```
- CCURDSCC_CAL_VOLT_SEL_INPUT_SIGNAL          : Input Signal
- CCURDSCC_CAL_VOLT_SEL_GROUND                : Ground (All Converters)
- CCURDSCC_CAL_VOLT_SEL_PLUS_REFERENCE        : +Ref (All Converters) (+<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_MINUS_REFERENCE       : -Ref (All Converters) (-<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_00_07_GROUND          : Ground (Converter 0)
```

```
- CCURDSCC_CAL_VOLT_SEL_00_07_PLUS_REFERENCE : +Ref (Converter 0) (+<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_00_07_MINUS_REFERENCE: -Ref (Converter 0) (-<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_08_15_GROUND         : Ground (Converter 1)
- CCURDSCC_CAL_VOLT_SEL_08_15_PLUS_REFERENCE : +Ref (Converter 1) (+<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_08_15_MINUS_REFERENCE: -Ref (Converter 1) (-<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_16_23_GROUND         : Ground (Converter 2)
- CCURDSCC_CAL_VOLT_SEL_16_23_PLUS_REFERENCE : +Ref (Converter 2) (+<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_16_23_MINUS_REFERENCE: -Ref (Converter 2) (-<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_24_31_GROUND         : Ground (Converter 3)
- CCURDSCC_CAL_VOLT_SEL_24_31_PLUS_REFERENCE : +Ref (Converter 3) (+<ref> Volts)
- CCURDSCC_CAL_VOLT_SEL_24_31_MINUS_REFERENCE: -Ref (Converter 3) (-<ref> Volts)
- CCURDSCC_DO_NOT_CHANGE
```

2.2.68 ccurDSCC_Set_Converter_Clock_Source()

The purpose of this call is to associate the given converter with a clock source.

```
/*************************************************************************
   Int ccurDSCC_Set_Converter_Clock_Source(void *Handle,
                                           CCURDSCC_CONVERTER conv, uint clock)

   Description: Set Converter Control and Status information

   Input:       void *Handle                    (handle pointer)
                CCURDSCC_CONVERTER      conv    (selected converter)
                uint                    clock   (clock source)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 *************************************************************************/

// CCURDSCC_CONVERTER
- CCURDSCC_CONVERTER_0
- CCURDSCC_CONVERTER_1
- CCURDSCC_CONVERTER_2
- CCURDSCC_CONVERTER_3

// clock
- CCURDSCC_CLOCK_PLL_0
- CCURDSCC_CLOCK_PLL_1
- CCURDSCC_CLOCK_PLL_2
- CCURDSCC_CLOCK_PLL_3
- CCURDSCC_CLOCK_EXTERNAL
```

2.2.69 ccurDSCC_Set_Converter_Negative_Cal()

This call sets the floating point value of the negative calibration for each of the channels that is maintained by the card. This negative gain is applied to the analog input data returned for each channel automatically by the hardware. The raw value set by this call is returned in the *ccurdscc_converter_cal_t* structure. The user can specify a floating point value of *CCURDSCC_DO_NOT_CHANGE* for channels that you do not want to alter.

```
/*************************************************************************
   int ccurDSCC_Set_Converter_Negative_Cal(void *Handle,
                                           ccurdscc_converter_cal_t *cal)

   Description: Set the Converter Negative Calibration data.
```

```
   Input:       void                        *Handle    (handle pointer)
   Output:      ccurdscc_converter_cal_t  *cal       (pointer to board cal)
   Return:      CCURDSCC_LIB_NO_ERROR                  (successful)
                CCURDSCC_LIB_BAD_HANDLE                (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN                  (device not open)
                CCURDSCC_LIB_INVALID_ARG               (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION           (local region not present)
                CCURDSCC_LIB_CALIBRATION_RANGE_ERROR (range error)
 ****************************************************************************/

typedef struct {
    uint    Raw[CCURDSCC_MAX_CHANNELS];
    double  Float[CCURDSCC_MAX_CHANNELS];
} ccurdscc_converter_cal_t;
```

2.2.70 ccurDSCC_Set_Converter_Offset_Cal()

This call sets the floating point value of the offset calibration for each of the channels that is maintained by the card. This zero offset is applied to the analog input data returned for each channel automatically by the hardware. The raw value set by this call is returned in the *ccurdscc_converter_cal_t* structure. The user can specify a floating point value of *CCURDSCC_DO_NOT_CHANGE* for channels that you do not want to alter.

```
/****************************************************************************
   int ccurDSCC_Set_Converter_Offset_Cal(void *Handle,
                                         ccurdscc_converter_cal_t *cal)

   Description: Set the Converter Offset Calibration data.

   Input:       void                         *Handle (handle pointer)
   Output:      ccurdscc_converter_cal_t  *cal    (pointer to board cal)
   Return:      CCURDSCC_LIB_NO_ERROR             (successful)
                CCURDSCC_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN             (device not open)
                CCURDSCC_LIB_INVALID_ARG          (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION      (local region not present)
 ****************************************************************************/

typedef struct {
    uint    Raw[CCURDSCC_MAX_CHANNELS];
    double  Float[CCURDSCC_MAX_CHANNELS];
} ccurdscc_converter_cal_t;
```

2.2.71 ccurDSCC_Set_Converter_Positive_Cal()

This call sets the floating point value of the positive calibration for each of the channels that is maintained by the card. This positive gain is applied to the analog input data returned for each channel automatically by the hardware. The raw value set by this call is returned in the *ccurdscc_converter_cal_t* structure. The user can specify a floating point value of *CCURDSCC_DO_NOT_CHANGE* for channels that you do not want to alter.

```
/****************************************************************************
   int ccurDSCC_Set_Converter_Positive_Cal(void *Handle,
                                           ccurdscc_converter_cal_t *cal)

   Description: Set the Converter Positive Calibration data.

   Input:       void                          *Handle   (handle pointer)
   Output:      ccurdscc_converter_cal_t    *cal      (pointer to board cal)
   Return:      CCURDSCC_LIB_NO_ERROR                 (successful)
                CCURDSCC_LIB_BAD_HANDLE               (no/bad handler supplied)
```

```
                CCURDSCC_LIB_NOT_OPEN                (device not open)
                CCURDSCC_LIB_INVALID_ARG             (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION         (local region not present)
 ****************************************************************************/

typedef struct {
    uint    Raw[CCURDSCC_MAX_CHANNELS];
    double  Float[CCURDSCC_MAX_CHANNELS];
} ccurdscc_converter_cal_t;
```

2.2.72 ccurDSCC_Set_Fifo_Channel_Select()

The hardware is capable of letting the user select which active channels they wish to monitor and place its converted data into the FIFO. This call sets the current channel selection mask. By default, all active channels are selected for storage into the FIFO. The mask has channel 0 as the least significant bit and channel 31 as the most significant bit. The advantage of this feature is to allow the user to ignore channels they do not wish to monitor resulting in performance improvement.

```
/****************************************************************************
   int ccurDSCC_Set_Fifo_Channel_Select(void *Handle, uint fifo_chan_sel)

   Description: Set the Fifo Channel Selection Mask

   Input:       void      *Handle               (handle pointer)
                uint      fifo_chan_sel         (fifo channels to select)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ****************************************************************************/
```

2.2.73 ccurDSCC_Set_Fifo_Threshold()

This call is used to set the FIFO threshold register. When samples are collected in the FIFO, an interrupt is generated *(if enabled)* once the FIFO threshold is reached. This register is set internally by the library during read operations. If the user wishes to bypass the API and driver reads, then they can use this register to control their data requests; for example, they can wait until a certain number of samples have been collected in the FIFO and then perform a user level DMA or programmed I/O to read the FIFO. The threshold maximum is defined by *CCURDSCC_FIFO_THRESHOLD_MAX*.

```
/****************************************************************************
   int ccurDSCC_Set_Fifo_Threshold(void *Handle, uint threshold)

   Description: Set the value of the specified board register.

   Input:       void      *Handle               (handle pointer)
                uint      threshold             (threshold to set)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ****************************************************************************/
```

2.2.74 ccurDSCC_Set_Interrupt_Control()

This call is used to enable or disable interrupt handling.

```
/*****************************************************************************
   int ccurDSCC_Set_Interrupt_Control(void *Handle, ccurdscc_interrupt_t *intr)

   Description: Set Interrupt Control information

   Input:       void *Handle                    (handle pointer)
   Output:      ccurdscc_interrupt_t   *intr    (pointer to interrupt control)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 *****************************************************************************/

typedef struct {
    int     global_int;
    int     fifo_buffer_lo_hi_int;
    int     plx_local_int;
} ccurdscc_interrupt_t;

// global_int
- CCURDSCC_GLOBAL_INT_DISABLE
- CCURDSCC_GLOBAL_INT_ENABLE
- CCURDSCC_DO_NOT_CHANGE

// fifo_buffer_lo_hi_int
- CCURDSCC_FIFO_INT_LO_HI_DISABLE
- CCURDSCC_FIFO_INT_LO_HI_ENABLE
- CCURDSCC_DO_NOT_CHANGE


// plx_local_int
- CCURDSCC_PLX_LOCAL_INT_DISABLE
- CCURDSCC_PLX_LOCAL_INT_ENABLE
- CCURDSCC_DO_NOT_CHANGE
```

2.2.75 ccurDSCC_Set_Interrupt_Status()

This call is used to clear the interrupt condition.

```
/*****************************************************************************
   int ccurDSCC_Set_Interrupt_Status(void *Handle, ccurdscc_interrupt_t *intr)

   Description: Set Interrupt Status information

   Input:       void *Handle                    (handle pointer)
   Output:      ccurdscc_interrupt_t   *intr    (pointer to interrupt status)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
 *****************************************************************************/

typedef struct {
    int     global_int;
    int     fifo_buffer_lo_hi_int;
    int     plx_local_int;
} ccurdscc_interrupt_t;

// global_int
- not used

// fifo_buffer_lo_hi_int
```

```
- CCURDSCC_FIFO_INT_LO_HI_IGNORE
- CCURDSCC_FIFO_INT_LO_HI_RESET
- CCURDSCC_DO_NOT_CHANGE

// plx_local_int
- CCURDSCC_PLX_LOCAL_INT_IGNORE
- CCURDSCC_PLX_LOCAL_INT_RESET
- CCURDSCC_DO_NOT_CHANGE
```

2.2.76 ccurDSCC_Set_Interrupt_Timeout_Seconds()

This call sets the read *timeout* maintained by the driver. It allows the user to change the default time out from 30 seconds to a user specified value. It is the time that the FIFO read call will wait before it times out. The call could time out if either the FIFO fails to fill or a DMA fails to complete. The device should have been opened in the blocking mode *(O_NONBLOCK not set)* for reads to wait for the operation to complete.

```
/************************************************************************
   int ccurDSCC_Set_Interrupt_Timeout_Seconds(void *Handle,
                                              int *int_timeout_secs)
   Description: Set Interrupt Timeout Seconds

   Input:       void *Handle                    (handle pointer)
   Output:      int         *int_timeout_secs   (pointer to int tout secs)
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
 ************************************************************************/
```

2.2.77 ccurDSCC_Set_PLL_Sync()

This call is used to synchronize the starting of the clocks by selecting the *sync_start* argument. The *external_go* and *external_sync* arguments are not used at this time.

```
/************************************************************************
   int ccurDSCC_Set_PLL_Sync(void *Handle, ccurdscc_PLL_sync_t *sync)

   Description: Set the value of the PLL Synchronization Register

   Input:       void *Handle                    (handle pointer)
                ccurdscc_PLL_sync_t *sync;      (pointer to sync struct)
   Output:      none
   Return:      CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ************************************************************************/

typedef struct {
    uint    sync_start[CCURDSCC_MAX_PLLS];
    uint    external_go;
    uint    external_sync;
} ccurdscc_PLL_sync_t;

// PLL Sync Start
- CCURDSCC_PLL_START
- CCURDSCC_PLL_STOP
- CCURDSCC_DO_NOT_CHANGE

// External Go
- CCURDSCC_EXTERNAL_GO_ENABLE
- CCURDSCC_EXTERNAL_GO_DISABLE
- CCURDSCC_DO_NOT_CHANGE

// External Sync
- CCURDSCC_EXTERNAL_SYNC_ENABLE
- CCURDSCC_EXTERNAL_SYNC_DISABLE
```

```
- CCURDSCC_DO_NOT_CHANGE
```

2.2.78 ccurDSCC_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. The input argument *value* is an *int* and therefore, this call does not support the *CCURDSCC_POSITIVE_CALIBRATION*, *CCURDSCC_NEGATIVE_CALIBRATION*, *CCURDSCC_SPI_RAM* and *CCURDSCC_OFFSET_CALIBRATION* commands as these expect array inputs.

Normally, users should not be changing these registers as it will bypass the API integrity and could result in an unpredictable outcome.

```
/*****************************************************************************
   int ccurDSCC_Set_Value(void *Handle, CCURDSCC_CONTROL cmd, int value)

   Description: Set the value of the specified board register.

   Input:       void *Handle                    (handle pointer)
                CCURDSCC_CONTROL cmd            (register definition)
                int value                       (value to be set)
   Output:      None
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
 *****************************************************************************/

typedef enum {
    CCURDSCC_BOARD_INFORMATION,             /* R Only */
    CCURDSCC_BOARD_CSR,                     /* R/W */

    CCURDSCC_INTERRUPT_CONTROL,             /* R/W */
    CCURDSCC_INTERRUPT_STATUS,              /* R/W */

    CCURDSCC_CONVERTER_0_CPM_CSR,           /* R/W */
    CCURDSCC_CONVERTER_0_CPM_ACCESS,        /* R/W */
    CCURDSCC_CONVERTER_0_CPM_READ_1,        /* R/W */
    CCURDSCC_CONVERTER_0_CPM_READ_2,        /* R Only */

    CCURDSCC_CONVERTER_1_CPM_CSR,           /* R/W */
    CCURDSCC_CONVERTER_1_CPM_ACCESS,        /* R/W */
    CCURDSCC_CONVERTER_1_CPM_READ_1,        /* R/W */
    CCURDSCC_CONVERTER_1_CPM_READ_2,        /* R Only */

    CCURDSCC_CONVERTER_2_CPM_CSR,           /* R/W */
    CCURDSCC_CONVERTER_2_CPM_ACCESS,        /* R/W */
    CCURDSCC_CONVERTER_2_CPM_READ_1,        /* R/W */
    CCURDSCC_CONVERTER_2_CPM_READ_2,        /* R Only */

    CCURDSCC_CONVERTER_3_CPM_CSR,           /* R/W */
    CCURDSCC_CONVERTER_3_CPM_ACCESS,        /* R/W */
    CCURDSCC_CONVERTER_3_CPM_READ_1,        /* R/W */
    CCURDSCC_CONVERTER_3_CPM_READ_2,        /* R Only */

    CCURDSCC_PLL_SYNC,                      /* R/W */

    CCURDSCC_CALIBRATION_VOLTAGE_CONTROL,   /* R/W */
    CCURDSCC_FIFO_CSR,                      /* R/W */
    CCURDSCC_FIFO_THRESHOLD,                /* R/W */
```

```
    CCURDSCC_FIFO_CHANNEL_SELECT,           /* R/W */

    CCURDSCC_PLL_0_STATUS,                  /* R Only */
    CCURDSCC_PLL_0_ACCESS,                  /* R/W */
    CCURDSCC_PLL_0_READ_1,                  /* R/W */
    CCURDSCC_PLL_0_READ_2,                  /* R Only */

    CCURDSCC_PLL_1_STATUS,                  /* R Only */
    CCURDSCC_PLL_1_ACCESS,                  /* R/W */
    CCURDSCC_PLL_1_READ_1,                  /* R/W */
    CCURDSCC_PLL_1_READ_2,                  /* R Only */

    CCURDSCC_PLL_2_STATUS,                  /* R Only */
    CCURDSCC_PLL_2_ACCESS,                  /* R/W */
    CCURDSCC_PLL_2_READ_1,                  /* R/W */
    CCURDSCC_PLL_2_READ_2,                  /* R Only */
    CCURDSCC_PLL_3_STATUS,                  /* R Only */
    CCURDSCC_PLL_3_ACCESS,                  /* R/W */
    CCURDSCC_PLL_3_READ_1,                  /* R/W */
    CCURDSCC_PLL_3_READ_2,                  /* R Only */

    CCURDSCC_FIRMWARE_SPI_COUNTER_STATUS,   /* R/W */
    CCURDSCC_CHANNEL_DATA,                  /* R Only */

    CCURDSCC_FIFO_DATA,                     /* R Only */

    CCURDSCC_POSITIVE_CALIBRATION,          /* R/W */
    CCURDSCC_NEGATIVE_CALIBRATION,          /* R/W */

    CCURDSCC_SPI_RAM,                       /* R/W */

    CCURDSCC_OFFSET_CALIBRATION,            /* R/W */
} CCURDSCC_CONTROL;
```

2.2.79 ccurDSCC_Shutdown_PLL_Clock()

This board has up to four programmable clocks that can be assigned in any combination to digital converters. If a clock is programmed but has not been assigned to any converter, it is preferable to shut down the particular clock so as to reduce noise.

```
/*****************************************************************************
   int ccurDSCC_Shutdown_PLL_Clock(void *Handle, CCURDSCC_PLL pll)

   Description: Shutdown_PLL_Clock

   Input:       void                   *Handle  (handle pointer)
                CCURDSCC_PLL            pll     (pll selection)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
 *****************************************************************************/
```

2.2.80 ccurDSCC_Start_PLL_Clock()

This call is similar to the *ccurDSCC_Set_PLL_Sync()* which provides the ability to synchronize the starting of the selected clocks.

```
/*****************************************************************************
   int ccurDSCC_Start_PLL_Clock(void *Handle, uint clock_mask)
```

```
   Description: Start PLL Clock

   Input:       void *Handle                    (handle pointer)
                uint               clock_mask   (selected clock mask)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ****************************************************************************/

// clock mask
- CCURDSCC_CLOCK_MASK_PLL_0
- CCURDSCC_CLOCK_MASK_PLL_1
- CCURDSCC_CLOCK_MASK_PLL_2
- CCURDSCC_CLOCK_MASK_PLL_3
```

2.2.81 ccurDSCC_Stop_PLL_Clock()

This call is similar to the *ccurDSCC_Set_PLL_Sync()* which provides the ability to stop the running clocks.

```
/****************************************************************************
   int ccurDSCC_Stop_PLL_Clock(void *Handle, uint clock_mask)

   Description: Stop PLL Clock

   Input:       void *Handle                    (handle pointer)
                uint               clock_mask   (selected clock mask)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR           (successful)
                CCURDSCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN           (device not open)
                CCURDSCC_LIB_INVALID_ARG        (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION    (local region not present)
 ****************************************************************************/
```

2.2.82 ccurDSCC_Volts_To_Data()

This call returns to the user the raw converted value for the requested voltage in the specified format. Voltage supplied must be within the input range of the selected board type. If the voltage is out of range, the call sets the voltage to the appropriate limit value.

```
/****************************************************************************
   int ccurDSCC_Volts_To_Data(void *Handle, double volts, int format)

   Description: Convert Volts to Data

   Input:       void     *Handle                (handle pointer)
                double   volts                  (volts to convert)
                int      format                 (conversion format)
   Output:      none
   Return:      int      data                   (returned data)
 ****************************************************************************/

// format
- CCURDSCC_TWOS_COMPLEMENT
- CCURDSCC_OFFSET_BINARY
```

2.2.83 ccurDSCC_Wait_For_Interrupt()

This call is made available to advanced users to bypass the API and perform their own data collection. The user can wait for either a FIFO low to high transition interrupt or a DMA complete interrupt. If a time out value greater than zero is specified, the call will time out after the specified seconds, otherwise it will not time out.

```
/************************************************************************
   int ccurDSCC_Wait_For_Interrupt(void *Handle, ccurdscc_driver_int_t *drv_int)

   Description: Wait For Interrupt

   Input:       void *Handle                     (handle pointer)
   Output:      ccurdscc_driver_int_t  *drv_int (pointer to drv_int struct)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_NO_LOCAL_REGION     (local region not present)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
 ************************************************************************/

typedef struct {
    unsigned long long  count;
    u_int               status;
    u_int               mask;
    int                 timeout_seconds;
} ccurdscc_driver_int_t;

// mask
- CCURDSCC_INTSTAT_LOCAL_PLX_MASK
- CCURDSCC_INTSTAT_FIFO_LOHI_THRESHOLD_MASK
```

2.2.84 ccurDSCC_Write()

This call is not supported for this Analog Input card.

```
/************************************************************************
   int ccurDSCC_Write(void *Handle, void *buf, int size, int *bytes_written,
               int *error)
   Description: Perform a write operation.

   Input:       void *Handle                     (handle pointer)
                int  size                        (number of bytes to write)
   Output:      void *buf                        (pointer to buffer)
                int  *bytes_written              (bytes written)
                int  *error                      (returned errno)
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN            (device not open)
                CCURDSCC_LIB_IO_ERROR            (write failed)
                CCURDSCC_LIB_NOT_IMPLEMENTED     (call not implemented)
 ************************************************************************/
```

2.2.85 ccurDSCC_Write_Channels_Calibration()

This call writes the user supplied calibration information to the on-board channel memory. This file must exist and be readable. This file could have been created by the *ccurDSCC_Read_Channels_Calibration()* call. Those channels that are not specified in the file are not altered on the board. Any blank lines or entries starting with '#' or '*' are ignored during parsing.

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

   int ccurDSCC_Write_Channels_Calibration(void *Handle, char *filename)

   Description: Write Channels Calibration information

   Input:       void *Handle                          (handle pointer)
                char *filename                        (pointer to filename)
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR                 (successful)
                CCURDSCC_LIB_BAD_HANDLE               (no/bad handler supplied)
                CCURDSCC_LIB_NOT_OPEN                 (device not open)
                CCURDSCC_LIB_INVALID_ARG              (invalid argument)
                CCURDSCC_LIB_NO_LOCAL_REGION          (local region not present)
                CCURDSCC_LIB_IO_ERROR                 (read error)
                CCURDSCC_LIB_CANNOT_OPEN_FILE         (file not writeable)
                CCURDSCC_LIB_CALIBRATION_RANGE_ERROR (range error)
 *************************************************************************/
```

Format:

```
#Chan   Negative  Offset    Positive
#====   ========  ======    ========
 ch00:  1.130771 -0.003152  1.130929
 ch01:  1.130661 -0.000795  1.130785
 ch02:  1.130400  0.001271  1.130840
 ----
 ch30:  1.130196  0.001695  1.130285
 ch31:  1.130440  0.001074  1.130285
```

3. Test Programs

This driver and API are accompanied with an extensive set of test examples. Examples under the *Direct Driver Access* do not use the API, while those under *Application Program Interface Access* use the API.

3.1 Direct Driver Access Example Tests

These set of tests are located in the *.../test* directory and do not use the API. They communicate directly with the driver. Users should be extremely familiar with both the driver and the hardware registers if they wish to communicate directly with the hardware.

3.1.1 ccurdscc_disp

Useful program to display all the analog input channels using various read modes. This program uses the *curses* library.

```
Usage: ./ccurdscc_disp [-b board] [-d delay] [-f format] [-m mode] [-p] [
 -b <board>            (default = 0)
 -d <delay - msec)     (delay between screen refresh)
 -f <format 'b', '2'>  (default = 'b' Offset Binary)
 -md                   (user DMA read mode [FIFO])
 -mD                   (driver DMA read mode [FIFO])
 -mf                   (user PIO read mode [FIFO])
 -mF                   (driver PIO read mode [FIFO])
 -mp                   (user PIO read mode [CHANNEL])
 -mP                   (driver PIO read mode [CHANNEL])
 -N                    (open device with O_NONBLOCK flag)
 -p                    (program board to max clock first)
```

Example display:

```
Board Number       [-b]: 0  ==> '/dev/ccurdscc0'
Delay              [-d]: 0 milli-seconds
Data Format        [-f]: 'Offset Binary'
Read Mode          [-m]: 'Driver DMA (FIFO Data) [BLOCK mode]'
Program Board      [-p]: 'No'
Input Voltage Range    : +/-5.0 Volts
Calibration Ref Voltage: 4.955 Volts
Read Error?            : '=== no ==='

Scan count:       7332, Delta:  19.2 usec (min= 17.4,max=122.2,av= 19.7)

         ##### Raw Data #####
         [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]
         ======   ======   ======   ======   ======   ======   ======   ======
Conv[0]  8000aa   7ffb02   800039   800685   7fff45   800dc3   8005a1   7fff55

Conv[1]  800a6f   7ffad8   7ff92b   7fee4d   800e9a   8000da   8000d1   7ffbc2

Conv[2]  80059d   801cef   800c9c   800319   7ff7c7   8004f9   8008ac   7fed14

Conv[3]  8009e6   7ff770   800145   8003f1   7ffd88   8004c4   800931   7ffb51



         ##### Volts #####
         [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]
         =======  =======  =======  =======  =======  =======  =======  =======
Conv[0] +0.00010 -0.00076 +0.00003 +0.00099 -0.00011 +0.00210 +0.00086 -0.00010
```

```
Conv[1] +0.00159 -0.00079 -0.00104 -0.00270 +0.00223 +0.00013 +0.00012 -0.00065

Conv[2] +0.00086 +0.00441 +0.00192 +0.00047 -0.00125 +0.00076 +0.00132 -0.00289

Conv[3] +0.00151 -0.00131 +0.00019 +0.00060 -0.00038 +0.00073 +0.00140 -0.00071
```

3.1.2 ccurdscc_get_sps

This program is useful in calculating the actual sampling rate of a running clock. It basically determines the rate at which samples are being placed in the FIFO and computes the rate, either for all channels or a specific channel. Hence, if you are running the board with clocks running at different rates for a set of channel groups, then you can determine the approximate rate at which samples are being collected in the FIFO. Additionally, it displays the minimum, maximum and average of the rate. This program uses the *curses* library.

```
Usage: ./ccurdscc_get_sps [-b board]
 -b <board>            (default = 0)
 -c <channel number>   (default = all channels)
```

Example display:

```
Device Name: /dev/ccurdscc0
#### Active Channel Found: 32 #### (Channel Mask: 0xffffffff)
#### All channels tracked     ####
delta= 9463.627 usec, samples=65408 rate=215.9849 Ksps (215.901/216.020/215.958)

Device Name: /dev/ccurdscc0
#### Active Channel Found: 32 #### (Channel Mask: 0xffffffff)
#### Only Channel 12 tracked  ####
delta= 9465.700 usec, samples= 2044 rate=215.9375 Ksps (215.934/216.011/215.974)
```

3.1.3 ccurdscc_rdreg

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

```
Usage: ./ccurdscc_rdreg [-b board] [-o offset]
 -b board: board number -- default board is 0
 -o offset: hex offset to read from -- default offset is 0x0
```

Example display:

```
Read at offset 0x0000: 0x92770102
```

3.1.4 ccurdscc_regedit

This is an interactive test to display and write to local, configuration and physical memory.

Usage: ccurdscc_tst <device number>

Example display:

```
Device Name: /dev/ccurdscc0
Initialize_Board: Firmware Rev. 0x2 successful
Virtual Address: 0x7ffff7ffc000
   1 = Create Physical Memory         2 = Destroy Physical memory
   3 = Display Channel Data           4 = Display Driver Information
   5 = Display Firmware RAM           6 = Display Physical Memory Info
   7 = Display Registers (CONFIG)     8 = Display Registers (LOCAL)
```

```
   9 = Dump Physical Memory          10 = Reset Board
  11 = Write Register (LOCAL)        12 = Write Register (CONFIG)
  13 = Write Physical Memory

Main Selection ('h'=display menu, 'q'=quit)->
```

3.1.5 ccurdscc_tst

This is an interactive test to exercise some of the driver features.

Usage: ccurdscc_tst <device number>

Example display:

```
Device Name: /dev/ccurdscc0
Initialize_Board: Firmware Rev. 0x2 successful
  01 = add irq                       02 = disable pci interrupts
  03 = enable pci interrupts         04 = get device error
  05 = get driver info               06 = get physical mem
  07 = init board                    08 = mmap select
  09 = mmap(CONFIG registers)        10 = mmap(LOCAL registers)
  11 = mmap(physical memory)         12 = munmap(physical memory)
  13 = no command                    14 = read operation
  15 = remove irq                    16 = reset board
  17 = write operation

Main Selection ('h'=display menu, 'q'=quit)->
```

3.1.6 ccurdscc_wreg

This is a simple test to write to the local registers at the user specified offset.

```
Usage: ./ccurdscc_wreg [-b board] [-o offset] [-v value]
 -b board : board selection --  default board is 0
 -o offset: hex offset to write to -- default offset is 0x0
 -v value: hex value to write at offset -- default value is 0x0
```

Example display:

```
Writing 0x00000000 to offset 0x0000
Read at offset 0x0000: 0x92770102
```

3.2 Application Program Interface (API) Access Example Tests

These set of tests are located in the *.../test/lib* directory and use the API.

3.2.1 ccurdscc_calibrate

This program provides an easy mechanism for users to save a calibration currently programmed in the card to an external file (-o option). The user can use this file as an input (-i option) to restore the board to a known calibration setting. When a system is booted the first time, the cards are not calibrated. The user can at this point decide to either run the board auto calibration (-A option) which takes approximately a minute or restore a previously calibrated setting.

```
Usage: ./ccurdscc_calibrate [-A] [-b board] [-c] [-f format] [-F] [-i inCalFile]
[-o outCalFile] [-p] [-s sample_rate] [-v] [-X clock]
 -A                    (perform Auto Calibration and exit)
 -b <board>            (board #, default = 0)
```

```
 -c <C#P#>              (Assign PLL clock to Converter: C#=0..3,a P#=0..3,e)
 -C <chan sel mask>     (channel selection mask)
 -f <format 'b', '2'>   (default = 'b' Offset Binary)
 -F                     (Enable High-Pass Filter)
 -i <In Cal File>       (input calibration file)
 -o <Out Cal File>      (output calibration file)
 -p                     (program board to max clock first)
 -s <sample rate>       (sample rate: 2000 - 216000 sps)
 -vi                    (enable input signal)
 -vg                    (enable [All Converters] ground calibration)
 -v+                    (enable [All Converters] +Ref Volt calibration 0.000)
 -v-                    (enable [All Converters] -Ref Volt calibration 0.000)
 -vg[0..3]              (enable [Converter 0..3] ground calibration)
 -v+[0..3]              (enable [Converter 0..3] +Ref Volt calibration 0.000)
 -v-[0..3]              (enable [Converter 0..3] -Ref Volt calibration 0.000)
 -X[0..3,e]             (Board External Clock Output Selection)
 -Z                     (do not display channel mismatch message)
```

Example display:

```
Device Name: /dev/ccurdscc0

Clock Used                [-c]: P0->C0, P0->C1, P0->C2, P0->C3
Channels Selected Mask    [-C]: 0xffffffff (Number of Channels: 32)

===> Dump to 'stdout'
#Chan   Negative  Offset    Positive
#====   ========  ======    ========
 ch00:  1.130771 -0.003152  1.130929
 ch01:  1.130661 -0.000795  1.130785
 ch02:  1.130400  0.001271  1.130840
 ch03:  1.130533 -0.000376  1.130493
 ch04:  1.130892 -0.002595  1.131122
 ch05:  1.130679  0.004905  1.130052
 ch06:  1.130594  0.001187  1.130434
 ch07:  1.130858 -0.003906  1.130828
 ch08:  1.130163 -0.001864  1.130473
 ch09:  1.129886  0.000182  1.130112
 ch10:  1.130696  0.001528  1.130695
 ch11:  1.130563  0.002583  1.130792
 ch12:  1.130038  0.001292  1.130218
 ch13:  1.130138  0.004800  1.130193
 ch14:  1.130033  0.000365  1.130079
 ch15:  1.130020  0.000823  1.130043
 ch16:  1.131290 -0.000225  1.131115
 ch17:  1.131321  0.002701  1.131271
 ch18:  1.131366  0.000433  1.130900
 ch19:  1.130888 -0.001283  1.130872
 ch20:  1.131421  0.003538  1.131319
 ch21:  1.131316 -0.002158  1.131280
 ch22:  1.130463 -0.000131  1.130433
 ch23:  1.130485 -0.000681  1.130507
 ch24:  1.130377 -0.000809  1.130356
 ch25:  1.130297  0.001249  1.130281
 ch26:  1.130479  0.001277  1.130448
 ch27:  1.130490  0.003979  1.130407
 ch28:  1.130967  0.002018  1.131076
 ch29:  1.130365  0.002326  1.130266
 ch30:  1.130196  0.001695  1.130285
 ch31:  1.130440  0.001074  1.130285
```

3.2.2 ccurdscc_compute_pll_clock

This test does not program the board. It simply returns to the user useful clock settings for a given frequency as computed by the software using vendor supplied algorithms. Advanced users who have intimate knowledge of the hardware can choose to change these settings, however results will be unpredictable.

```
Usage: ./ccurdscc_compute_pll_clock -[ft]
          -f <desired freq>     (default = 13.824000 MHz)
          -f <freq_start,freq_end,freq_inc>
          -t <max error tolerance> (default = 1000 ppm)
          -v           (enable verbose)
          -s           (Minimize VCO Speed)
```

Example display:

```
Reference Frequency (fRef - MHz)         = 65.536000
Desired Frequency (fDesired - MHz)       = 13.824000,13.824000,1.000000
VCO Speed Mode                           = Maximize
Minimum Phase Detect Freq (fPFDmin - MHz)= 1.000000
Max Error Tolerance (tol - ppm)          = 1000
VCO gain (kfVCO - MHz/volt)              = 520.000000
Minimum VCO Frequency (fVcoMin - MHz)    = 100.000000
Maximum VCO Frequency (fVcoMax - MHz)    = 400.000000
Minimum Ref Frequency (nRefMin - MHz)    = 1.000000
Maximum Ref Frequency (nRefMax - MHz)    = 4095.000000
Minimum FeedBk Frequency (nFbkMin - MHz) = 12.000000
Maximum FeedBk Frequency (nFbkMax - MHz) = 16383.000000


  Requested Clock Freq       : 13.8240000000 MHz
  Actual Clock Freq          : 13.8240000000 MHz
  Frequency Delta            : 0.000000 Hz
  Reference Frequency Divider: 32
  Feedback Frequency Divider : 189
  Post Divider Product       : 28   (D1=6 D2=3 D3=0)
  fVCO                       : 387.072000 MHz
  synthErr                   : 0.0000000000 ppm
  Gain Margin                : 9.367013
  Tolerance Found            : 0
  Charge Pump                : 22.5 uAmp
  Loop Resistance            : 12 Kohm
  Loop Capacitance           : 185 pF
```

3.2.3 ccurdscc_disp

Useful program to display all the analog input channels using various read modes. This program uses the *curses* library.

```
Usage: ./ccurdscc_disp [-A#] [-b board] [-c] [-d delay] [-D debugfile] [-E
ExpInpVolt] [-f format] [-F] [-m mode] [-N] [-o outfile] [-p] [-s sample_rate] [-
v] [-X clock]
 -A <#>                 (display rolling average of # values.)
 -b <board>             (default = 0)
 -c <C#P#>              (Assign PLL clock to Converter: C#=0..3,a P#=0..3,e)
 -C <chan sel mask>     (channel selection mask)
 -d <delay - msec)      (delay between screen refresh)
 -D <Debug File>        (write to debug file)
 -E <ExpInpVolts>@<Tol> (Expected Input Volts@Tolerance)
 -f <format 'b', '2'>   (default = 'b' Offset Binary)
 -F                     (Enable High-Pass Filter)
```

```
 -l <#>                   (specify loop count)
 -md                      (User DMA read mode [FIFO])
 -mD                      (Driver DMA read mode [FIFO])
 -mf                      (User PIO read mode [FIFO])
 -mF                      (Driver PIO read mode [FIFO])
 -mp                      (User PIO read mode [CHANNEL])
 -mP                      (Driver PIO read mode [CHANNEL])
 -mx                      (User DMA read mode [CHANNEL])
 -mX                      (Driver DMA read mode [CHANNEL])
 -N                       (open device with O_NONBLOCK flag)
 -o <#>@<Output File>     (average # count, write to output file)
 -p                       (program board to max clock first)
 -s <sample rate>         (sample rate: 2000 - 216000 sps)
 -vi                      (enable input signal)
 -vg                      (enable [All Converters] ground calibration)
 -v+                      (enable [All Converters] +Ref Volt calibration)
 -v-                      (enable [All Converters] -Ref Volt calibration)
 -vg[0..3]                (enable [Converter 0..3] ground calibration)
 -v+[0..3]                (enable [Converter 0..3] +Ref Volt calibration)
 -v-[0..3]                (enable [Converter 0..3] -Ref Volt calibration)
 -X[0..3,e]               (Board External Clock Output Selection)
```

Example display:

```
Rolling Average Count [-A]: 10000
Board Number          [-b]: 0  ==> '/dev/ccurdscc0'
Clock Used            [-c]: P0->C0, P0->C1, P0->C2, P0->C3
Channel Sel Mask      [-C]: 0xffffffff
Delay                 [-d]: 0 milli-seconds
Expected Input Volts  [-E]: 0.000000 volts (Tolerance 0.005000 volts)
Data Format           [-f]: Offset Binary
High Pass Filter      [-F]: 'Last set state'
Loop Count            [-l]: ***Forever***
Read Mode             [-m]: Driver DMA (Channel Data)
Output File (Calib)   [-o]: 'outfile' (Rolling Average Count = 10000/10000)
Program Board         [-p]: No
Calibration Sel       [-v]: Ground (All Converters)
External Clock Output [-X]: PLL 0
Input Voltage Range       : +/-5.0 Volts
Calibration Ref Voltage   : 4.955 Volts
Read Error?               : ===== no ====
Tolerance Exceeded Count  : 0

Scan count:       27217, Total Delta:  10.4 usec (min= 10.0,max= 34.1,av= 10.3)

         ##### Raw Data (Rolling Average Count [10000/10000]) #####
         [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]
         ======   ======   ======   ======   ======   ======   ======   ======
Conv[0]  80005e   800096   80000e   80001b   7fff68   800092   7fffed   800057

Conv[1]  7ffdd4   800091   7fff6f   7fffc4   800027   7fffe1   7fffe5   7fffa5

Conv[2]  8000cc   7ffe2d   7ffd15   80009e   800232   7fff5b   800015   7fffd7

Conv[3]  80001b   8000c2   7fff3b   8002ea   800079   8000e5   7ffefa   8001b0


         ##### Volts (Rolling Average Count [10000/10000]) #####
         [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]
         =======  =======  =======  =======  =======  =======  =======  =======
Conv[0] +0.00006 +0.00009 +0.00001 +0.00002 -0.00009 +0.00009 -0.00001 +0.00005
```

```
Conv[1] -0.00033 +0.00009 -0.00009 -0.00004 +0.00002 -0.00002 -0.00002 -0.00005

Conv[2] +0.00012 -0.00028 -0.00045 +0.00009 +0.00033 -0.00010 +0.00001 -0.00002

Conv[3] +0.00002 +0.00012 -0.00012 +0.00044 +0.00007 +0.00014 -0.00016 +0.00026
```

3.2.4 ccurdscc_fifo

This is a powerful test program that exercises the FIFO capabilities of the board under various reading modes.

```
Usage: ./ccurdscc_fifo [-A] [-b board] [-B DMA bufs] [-c] [-d debugfile] [-E
ExpInpVolt] [-f format] [-F] [-l count] [-m mode] [-N] [-p] [-r size] [-s
sample_rate] [-v] [-X clock]
 -A                      (perform Auto Calibration and exit)
 -b <board>              (board #, default = 0)
 -B <DMA Cont Bufs>      (DMA Continuous Buffers)
 -c <C#P#>               (Assign PLL clock to Converter: C#=0..3,a P#=0..3,e)
 -C <chan sel mask>      (channel selection mask)
 -d <Debug File>         (write to debug file - standard format)
 -d +<Debug File>        (write to debug file - for gunzip plot format)
 -E <ExpInpVolts>@<Tol>  (Expected Input Volts@Tolerance)
 -f <format 'b', '2'>    (default = 'b' Offset Binary)
 -F                      (Enable High-Pass Filter)
 -l <loop count>         (Loop count (def=1000))
 -mC                     (Driver DMA read mode [CONTINUOUS FIFO])
 -md                     (User DMA read mode [FIFO])
 -mD                     (Driver DMA read mode [FIFO])
 -mf                     (User PIO read mode [FIFO])
 -mF                     (Driver PIO read mode [FIFO])
 -N                      (open device with O_NONBLOCK flag)
 -p                      (program board to max clock first)
 -r <read size>          (sample to read: 1 - 65535)
 -s <sample rate>        (sample rate: 2000 - 216000 sps)
 -vi                     (enable input signal)
 -vg                     (enable [All Converters] ground calibration)
 -v+                     (enable [All Converters] +Ref Volt calibration 0.000)
 -v-                     (enable [All Converters] -Ref Volt calibration 0.000)
 -vg[0..3]               (enable [Converter 0..3] ground calibration)
 -v+[0..3]               (enable [Converter 0..3] +Ref Volt calibration 0.000)
 -v-[0..3]               (enable [Converter 0..3] -Ref Volt calibration 0.000)
 -X[0..3,e]              (Board External Clock Output Selection)
 -Z                      (do not display channel mismatch message)
```

Example display:

```
./ccurdscc_fifo -vg -E0@0.025

Read Mode: 'Driver DMA (FIFO Data)'
Device Name: /dev/ccurdscc0

Clock Used                [-c]: P0->C0, P0->C1, P0->C2, P0->C3
External Clock Output     [-X]: PLL 0
Channels Selected Mask    [-C]: 0xffffffff (Number of Channels: 32)
Expected Input Volts      [-E]: 0.000000 volts (Tolerance 0.025000 volts)
Channel Mismatch Messages [-Z]: ENABLED
Driver Interrupt Timeout=30 seconds

Clock settling delay (2 seconds)...done

Read Issued In BLOCK mode.
Waiting for 49152 FIFO samples: Num. active channels=32, sample_rate=216000.00
SPS...done
001000: Samples Read=49152 Remaining=31904 t=6.26ms (6.15/10.95/6.30) 31.39MB/s
tol=0 overflow=0

Total Tolerance Exceed Count=0
```

3.2.5 ccurdscc_tst_lib

This is an interactive test that accesses the various supported API calls.

```
Usage: ccurdscc_tst_lib <device number>
```

Example display:

```
Configured Channels Information...
       Last Action    : Restore Library Variables
       Last Specified Reference Frequency: 65.536000 MHz
       PLL_0: Actual Freq= 0.00000000, C0=  0 C1=  0 C2=  0 C3=  0
       PLL_1: Actual Freq= 0.00000000, C0=  0 C1=  0 C2=  0 C3=  0
       PLL_2: Actual Freq= 0.00000000, C0=  0 C1=  0 C2=  0 C3=  0
       PLL_3: Actual Freq= 0.00000000, C0=  0 C1=  0 C2=  0 C3=  0
     Ext Clk:  Clock Freq= 0.00000000, C0=  0 C1=  0 C2=  0 C3=  0
  01 = Abort DMA                        02 = Add Irq
  03 = Allocate DMA Cont. Bufs          04 = Clear Driver Error
  05 = Clear Library Error              06 = Configure Channels
  07 = Configure Channels Info          08 = Disable Pci Interrupts
  09 = Display BOARD Registers          10 = Display CONFIG Registers
  11 = Enable Pci Interrupts            12 = Get Board CSR
  13 = Get Board Information            14 = Get Converter Cal CSR
  15 = Get Converter CSR                16 = Get Converter Information
  17 = Get Converter Calibration        18 = Get Driver Error
  19 = Get Driver Information           20 = Get Driver Read Mode
  21 = Get Fifo Channel Select          22 = Get Fifo Information
  23 = Get Interrupt Control            24 = Get Interrupt Status
  25 = Get Library Error                26 = Get Mapped Config Pointer
  27 = Get Mapped Local Pointer         28 = Get Number of DMA Cont. Buffers
  29 = Get Physical Memory              30 = Get PLL Information
  31 = Get PLL Status                   32 = Get PLL Synchronization
  33 = Get Value                        34 = Initialize Board
  35 = MMap Physical Memory             36 = Munmap Physical Memory
  37 = One Shot Test                    38 = Perform Auto Calibration
  39 = Perform Neg Calib (External)     40 = Perform Offset Calib (External)
  41 = Perform Pos Calib (External)     42 = Perform Negative Calibration
  43 = Perform Offset Calibration       44 = Perform Positive Calibration
  45 = Program CPM (Advanced)           46 = Program PLL (Advanced)
  47 = Program PLL Clock                48 = Read Operation
  49 = Read Channels                    50 = Read Channels Calibration
  51 = Remove Irq                       52 = Remove DMA Cont. Buffers
  53 = Reset Board                      54 = Reset Calibration
  55 = Reset Converter                  56 = Reset DMA Continuous Buffers
  57 = Reset Fifo                       58 = Select Driver Read Mode
  59 = Set Converter Cal CSR            60 = Set Converter Clock Source
  61 = Set Converter Negative Cal       62 = Set Converter Offset Cal
  63 = Set Converter Positive Cal       64 = Set Board CSR
  65 = Set Fifo Channel Select          66 = Set Fifo Threshold
  67 = Set Interrupt Control            68 = Set Interrupt Status
  69 = Set PLL Synchronization          70 = Set Value
  71 = Shutdown PLL Clock               72 = Start PLL Clock
  73 = Stop PLL Clock                   74 = Write Operation
  75 = Write Channels Calibration

Main Selection ('h'=display menu, 'q'=quit)->
```

This page intentionally left blank