Software Interface
CCURDSCC (WC-AD3224-DS)

PCIe 32-Channel Delta Sigma Converter Card (DSCC)

Driver	ccurdscc (WC-AD3224-DS)	
OS	RedHawk	
Vendor	Concurrent Real-Time, Inc.	
Hardware	PCIe 32-Channel Delta Sigma Converter Card (CP-AD3224-DS) (CP-AD3224-DS-10)	
Date	August 23, 2018	rev 2018.1

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 Real-Time PCI Express 32-Channel Delta Sigma Converter Card (DSCC). For additional information on programming, please refer to the *Concurrent Real-Time 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 Real-Time.
- PCIe 32-Channel Delta Sigma Converter Card (DSCC) Design Specification (No. 0610099) by Concurrent Real-Time.

2. Software Support

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

2.1 Direct Driver Access

2.1.1 open(2) system call

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

```
int   fp;
fp = open("/dev/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. Basically, the driver only allows one application to open a board at a time. The reason for this is that the application can have full access to the card, even at the board and API level. If another application were to communicate with the same card concurrently, the results would be unpredictable unless proper synchronization is performed. This synchronization would be external to the driver, between the two applications so as not to affect each other. This driver allows multiple applications to open the same board by specifying the additional *oflag O_APPEND*. It is then the

responsibility of the user to ensure that the various applications communicating with the same cards are properly synchronized. Various tests supplied in this package has the *O_APPEND* flags enabled, however, it is strongly recommended that only one application be used with a single card at a time, unless the user is well aware of how the applications are going to interact with each other and accept any unpredictable results.

2.1.2 ioctl(2) system call

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

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

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

Driver IOCTL command:

```
IOCTL_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_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 support for MSI interrupts are configured, they will be enabled. Normally, there is no need to call this *ioctl* as the interrupt handler is already added when the driver is loaded. This *ioctl* 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* or *CCURDSCC_DEMAND_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* or *CCURDSCC_DEMAND_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_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 bypass the driver and library. Care must be taken when performing user level DMA, as incorrect programming could lead to unpredictable results, including but not limited to corrupting the kernel and any device connected to the system.

IOCTL_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 *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 application.

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* or *CCURDSCC_DEMAND_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 the 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.

CCURDSCC_DEMAND_DMA_FIFO: This is identical to *CCURDSCC_DMA_FIFO,* except that the DMA is being performed in the DEMAND mode, as opposed to the NON-DEMAND mode.

CCURDSCC_DEMAND_DMA_CONTINUOUS: This is identical to *CCURDSCC_DMA_CONTINUOUS,* except that the DMA is being performed in the DEMAND mode, as opposed to the NON-DEMAND mode.

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_AutoCal_Status()
ccurDSCC_Clear_Driver_Error()
ccurDSCC_Clear_Lib_Error()
ccurDSCC_Close()
ccurDSCC_Compute_PLL_Clock()
ccurDSCC_Configure_Channels()
ccurDSCC_Configure_Channels_Info()
ccurDSCC_Create_Factory_Calibration()
ccurDSCC_Create_User_Checkpoint()
ccurDSCC_Data_To_Volts()
ccurDSCC_Disable_Pci_Interrupts()
ccurDSCC_Enable_Pci_Interrupts()
ccurDSCC_Fast_Memcpy()
ccurDSCC_Fast_Memcpy_Unlocked()
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_TestBus_Control()
ccurDSCC_Get_Value()
ccurDSCC_Hex_To_Fraction()
ccurDSCC_Initialize_Board()
ccurDSCC_Initialize_PLL_Input_Struct()
ccurDSCC_MMap_Physical_Memory()
ccurDSCC_Munmap_Physical_Memory()
ccurDSCC_NanoDelay()
ccurDSCC_Open()
```

```
ccurDSCC_Perform_Auto_Calibration()
ccurDSCC_Perform_Auto_Calibration_Async()
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_Read_Serial_Prom()
ccurDSCC_Read_Serial_Prom_Item()
ccurDSCC_Remove_DMA_Continuous_Buffers()
ccurDSCC_Remove_Irq()
ccurDSCC_Reset_Board()
ccurDSCC_Reset_Converter()
ccurDSCC_Reset_DMA_Continuous_Buffers()
ccurDSCC_Reset_Fifo()
ccurDSCC_Restore_Factory_Calibration()
ccurDSCC_Restore_User_Checkpoint()
ccurDSCC_Select_Driver_Read_Mode()
ccurDSCC_Serial_Prom_Write_Override()
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_TestBus_Control()
ccurDSCC_Set_Value()
ccurDSCC_Shutdown_PLL_Clock()
ccurDSCC_Start_PLL_Clock()
ccurDSCC_Stop_PLL_Clock()
ccurDSCC_View_Factory_Calibration()
ccurDSCC_View_User_Checkpoint()
ccurDSCC_Volts_To_Data()
ccurDSCC_Wait_For_Interrupt()
ccurDSCC_Write()
ccurDSCC_Write_Channels_Calibration()
ccurDSCC_Write_Serial_Prom()
ccurDSCC_Write_Serial_Prom_Item()
```

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* or *CCURDSCC_DEMAND_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* or *CCURDSCC_DEMAND_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_AutoCal_Status()

This call returns the current status information on a previously full automatic calibration performed by either the *ccurDSCC_Perform_Auto_Calibration()* or *ccurDSCC_Perform_Auto_Calibration_Async()* call.

```
/***************************************************************************
   int ccurDSCC_AutoCal_Status(void *Handle, ccurdscc_autocal_status_t
                               *autocal_status)

   Description: Return AutoCal Status

   Input:       void *Handle                                (handle pointer)
   Output:      ccurdscc_autocal_status_t *autocal_status (pointer to struct)
                - uint    autocal_state;
                - uint    start_time;
                - uint    end_time;
                - uint    duration;
                - int     time_remaining;
   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)
 ***************************************************************************/
```

The following is the information supplied to the call:

```
/* auto calibration state */
typedef enum {
    CCURDSCC_AUTOCAL_INACTIVE   = 0,
    CCURDSCC_AUTOCAL_STARTED    = 1,
    CCURDSCC_AUTOCAL_SUCCESSFUL = 2,
    CCURDSCC_AUTOCAL_FAILED     = 3,
} CCURDSCC_AUTOCAL_STATE;

/* auto calibration status struct */
typedef struct
{
    uint           autocal_state;  /* autocal state */
    time_t         start_time;     /* seconds since Epoch (00:00:00 UTC,
                                      January 1, 1970) */
    time_t         end_time;       /* seconds since Epoch (00:00:00 UTC,
                                      January 1, 1970)*/
    uint           duration;       /* duration in seconds */
    int            time_remaining; /* time remaining in seconds */
    unsigned long  process_id;     /* current process id */
} ccurdscc_autocal_status_t;

// autocal_state
- CCURDSCC_AUTOCAL_INACTIVE: Unknown/initial state. Recommend that users perform an auto-
                             calibration prior to testing.
```

- CCURDSCC_AUTOCAL_STARTED: User has initiated an auto-calibration. The board should not be accessed until the auto-calibration is complete. Auto-calibration state can be monitored using this call.

- CCURDSCC_AUTOCAL_SUCCESSFUL: Auto-calibration has completed successfully.

- CCURDSCC_AUTOCAL_FAILED: Last auto-calibration operation failed. Make sure that clocks are running and board is properly configured prior to restarting the auto-calibration.

// start_time: Time when auto-calibration was last initiated

// end_time: Time when auto-calibration last completed

// duration: Time in seconds it is taking for auto-calibration

// time_remaining: Approximate time in seconds remaining before auto-calibration completes. A value of '-1' with a *CCURDSCC_AUTOCAL_STARTED* state, indicates that the time has exceeded the expected duration for the card. Users can monitor the *time_remaining* field to determine when the auto-calibration completes by waiting for it to get to '0'. User can also monitor *end_time* to determine the auto-calibration completion once it has started.

// process_id: ID of the process that started the auto calibration. This ID is cleared when the calibration has completed, whether it was successful or not. This process ID field is used internally to correct the calibration state if the process is abnormally terminated. Normally, the ID will display a '0' when auto calibration is not active.

2.2.5 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)
 *************************************************************************/
```

2.2.6 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)
 ****************************************************************************
```

2.2.7 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)
 ****************************************************************************/
```

2.2.8 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;
```

2.2.9 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.

2.2.10 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 {
    u_int32_t CPM_csr;                /* converter status/control register */
    u_int32_t CPM_read_1;             /* converter CPM Read Register 1 */
    u_int32_t CPM_read_2;             /* converter CPM Read Register 2 */
} _CPM_Checkpoint_t;

typedef struct {
    u_int32_t PLL_read_1;             /* PLL Read Register 1 */
    u_int32_t PLL_read_2;             /* PLL Read Register 2 */
} _PLL_Checkpoint_t;

typedef struct
{
    uint           autocal_state;  /* autocal state */
    time_t         start_time;     /* seconds since Epoch (00:00:00 UTC,
                                      January 1, 1970) */
    time_t         end_time;       /* seconds since Epoch (00:00:00 UTC,
                                      January 1, 1970)*/
    uint           duration;       /* duration in seconds */
    int            time_remaining; /* time remaining in seconds */
    unsigned long  process_id;     /* current process id */
} ccurdscc_autocal_status_t;

typedef struct {
    double                      last_specified_fRef;
    _programmed_pll_t           programmed_PLL[CCURDSCC_PLL_MAX_WITH_EXTERNAL];
                                      /* +1 for external clock */
    _CPM_Checkpoint_t           CPM_Checkpoint[CCURDSCC_MAX_CONVERTERS];
```

```
    _PLL_Checkpoint_t            PLL_Checkpoint[CCURDSCC_MAX_PLLS];
    u_int                        PLL_Checkpoint_sync;
    ccurdscc_autocal_status_t    autocal_status;
} _ccurdscc_preserve_t;
```

2.2.11 ccurDSCC_Create_Factory_Calibration()

This routine is used by Concurrent Real-Time to program factory calibration into the serial prom for the voltage range. These settings are non-volatile and preserved through a power cycle. Users should refrain from using this API, as it will no longer reflect the factory calibration shipped with the card.

Prior to using this call, the user will need to issue the *ccurDSOCC_Serial_Prom_Write_Override()* to allowing writing to the serial prom. The supporting calls for this API are *ccurDSCC_View_Factory_Calibration()* and *ccurDSCC_Restore_Factory_Calibration().*

```
/*************************************************************************
   ccurDSCC_Create_Factory_Calibration()

   Description: Create a Factory Calibration from user specified file

   Input:       void                  *Handle   (handlepointer)
                char                  *filename (pointer to filename)
                ccurdscc_bool         force     (force programming)
                  -- CCURDSCC_TRUE
                  -- CCURDSCC_FALSE
   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_CANNOT_OPEN_FILE    (file not readable)
                CCURDSCC_LIB_NO_LOCAL_REGION     (error)
                CCURDSCC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
                CCURDSCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
                CCURDSCC_LIB_INVALID_CRC         (invalid CRC)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)

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

The *filename* contains the *negative, offset* and *positive* floating point values for each channel. Additionally, the file contains the various programming information for each of the four clocks along with the channel configuration information. This file can be created with the *ccurDSCC_Write_Channels_Calibration()* API, once the card has been calibrated for all channels with a specific sample speed. The *ccurdscc_calibrate* utility can be used to create this file (*./ccurdscc_calibrate –A –c00 –c11 –c22 –c33 –oCalOut*).
Sample file for all channels configured using all four PLL clocks.

```
#Date           : Tue May  6 09:41:14 2014
#Board Serial No: 4294967295 (0xffffffff)

#PLL#: PLL_read_1 PLL_read_2
 PLL0: 0xbd361020 0x00033380
 PLL1: 0xbd361020 0x00033380
 PLL2: 0xbd361020 0x00033380
 PLL3: 0xbd361020 0x00033380

 PLL_sync: 0x0000000f
 PLL_fRef: 65.536000
 ExtClk_Freq: 0.000000
```

```
#CPM#: CPM_csr    CPM_read_1 CPM_read_2
 CPM0: 0x00000000 0xffff8681 0x000000ff
 CPM1: 0x00000001 0xffff8681 0x000000ff
 CPM2: 0x00000002 0xffff8681 0x000000ff
 CPM3: 0x00000003 0xffff8681 0x000000ff

#Chan   Negative    Offset      Positive
#====   ========    ======      ========
 ch00:  1.13817756  0.00062346  1.13801234
 ch01:  1.13798389  0.00150204  1.13819617
 ch02:  1.13831496  0.00120997  1.13837921
 ch03:  1.13845147 -0.00010490  1.13879659
 ch04:  1.13844664 -0.00023127  1.13849963
 ch05:  1.13726768  0.00049591  1.13734001
 ch06:  1.13817720  0.00073552  1.13847080
 ch07:  1.13744774  0.00022411  1.13748805
 ch08:  1.13740669  0.00098467  1.13741457
 ch09:  1.13705322  0.00057578  1.13730226
 ch10:  1.13762401  0.00078082  1.13753394
 ch11:  1.13752466  0.00105739  1.13785791
 ch12:  1.13686886  0.00154376  1.13671683
 ch13:  1.13768282 -0.00008225  1.13752552
 ch14:  1.13702987  0.00007272  1.13759905
 ch15:  1.13728815  0.00023723  1.13729101
 ch16:  1.13857680  0.00011683  1.13840102
 ch17:  1.13813042 -0.00044107  1.13833145
 ch18:  1.13823962  0.00194907  1.13788610
 ch19:  1.13829327  0.00137329  1.13824026
 ch20:  1.13850332  0.00036001  1.13853591
 ch21:  1.13792067  0.00034928  1.13806604
 ch22:  1.13850390 -0.00054002  1.13857194
 ch23:  1.13796982  0.00003457  1.13769358
 ch24:  1.13748415  0.00031710  1.13751006
 ch25:  1.13714905 -0.00077724  1.13708145
 ch26:  1.13738782  0.00061631  1.13714448
 ch27:  1.13729314 -0.00051141  1.13764725
 ch28:  1.13671837  0.00062704  1.13680499
 ch29:  1.13711190  0.00035882  1.13709227
 ch30:  1.13692011 -0.00025511  1.13711870
 ch31:  1.13708468  0.00056863  1.13709323
```

The *force* variable can be set to either *CCURDSCC_TRUE* or *CCURDSCC_FALSE*. This API validates the CRC read from the serial prom against what it was expecting and if there is a mismatch and the *force* variable is set to *CCURDSCC_FALSE,* the call will fail.

2.2.12 ccurDSCC_Create_User_Checkpoint()

This routine allows the user to program channel configuration and calibration information into the serial prom for all the channels. These settings are non-volatile and preserved through a power cycle.

The user supplied input can be in the form of an input calibration file previously created with the *ccurDSCC_View_User_Checkpoint()* API that contains negative, offset and positive and channel configuration for each channel to be programmed, or alternately, if the input file is *NULL,* capture a snapshot of the current board settings. Normally, the user could, prior to specific test runs, ensure that the surrounding environment (e.g. temperature) represents the same as the environment during the actual run, and then perform an auto-calibration of all the channels. Once the calibration is

complete, this API can store the current settings in the serial prom for later restore with the *ccurDSCC_Restore_User_Checkpoint() API*.

Prior to using this call, the user will need to issue the *ccurDSCC_Serial_Prom_Write_Override()* to allowing writing to the serial prom. The supporting calls for this API are *ccurDSCC_View_User_Checkpoint()* and *ccurDSOCC_Restore_User_Checkpoint().*

```
/*************************************************************************
   ccurDSCC_Create_User_Checkpoint()

   Description: Create a User Checkpoint from user specified file

   Input:       void                    *Handle   (handle pointer)
                _ccurdscc_sprom_access_t item     (select item)
                  -- CCURDSCC_SPROM_USER_CHECKPOINT_1
                  -- CCURDSCC_SPROM_USER_CHECKPOINT_2
                char                    *filename (pointer to filename or NULL)
                ccurdscc_bool           force     (force programming)
                  -- CCURDSCC_TRUE
                  -- CCURDSCC_FALSE
   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_CANNOT_OPEN_FILE    (file not readable)
                CCURDSCC_LIB_NO_LOCAL_REGION     (error)
                CCURDSCC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
                CCURDSCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
                CCURDSCC_LIB_INVALID_CRC         (invalid CRC)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
 *************************************************************************/

typedef enum {
    CCURDSCC_SPROM_HEADER=1,
    CCURDSOCC_SPROM_FACTORY,
    CCURDSCC_SPROM_USER_CHECKPOINT_1,
    CCURDSCC_SPROM_USER_CHECKPOINT_2,
} _ccurdscc_sprom_access_t;
```

The *filename* contains the *converter CSR*, *negative, offset* and *positive* calibration in floating point for each channel. Additionally, the file contains the various programming information for each of the four clocks along with the channel configuration information. This file can be created with the *ccurDSOCC_View_User_Checkpoint()* API, once the card has been calibrated and information stored in the serial PROM with this *ccurDSCC_Create_User_Checkpoint()* and filename set to *NULL*.
Below is a sample file for all channels configured. User needs to refer to the hardware programming manual to get information on the converter CSR register.

```
# User Checkpoint from serial prom
#            Date: Tue May 06 10:00:37 EDT 2014
#      Checkpoint: User Checkpoint 1
# Board Serial No: 4294967295 (0xffffffff)
#             CRC: 44D0
#
#PLL#: PLL_read_1 PLL_read_2
 PLL0: 0xbd361020 0x00033380
 PLL1: 0xbd361020 0x00033380
 PLL2: 0xbd361020 0x00033380
 PLL3: 0xbd361020 0x00033380
```

```
 PLL_sync: 0x0000000f
 PLL_fRef: 65.536000
 ExtClk_Freq: 0.000000

#CPM#: CPM_csr    CPM_read_1 CPM_read_2
 CPM0: 0x00000000 0xffff8681 0x000000ff
 CPM1: 0x00000001 0xffff8681 0x000000ff
 CPM2: 0x00000002 0xffff8681 0x000000ff
 CPM3: 0x00000003 0xffff8681 0x000000ff

#Chan   Negative    Offset      Positive
#====   ========    ======      ========
 ch00:  1.13817756  0.00062346  1.13801234
 ch01:  1.13798389  0.00150204  1.13819617
 ch02:  1.13831496  0.00120997  1.13837921
 ch03:  1.13845147 -0.00010490  1.13879659
 ch04:  1.13844664 -0.00023127  1.13849963
 ch05:  1.13726768  0.00049591  1.13734001
 ch06:  1.13817720  0.00073552  1.13847080
 ch07:  1.13744774  0.00022411  1.13748805
 ch08:  1.13740669  0.00098467  1.13741457
 ch09:  1.13705322  0.00057578  1.13730226
 ch10:  1.13762401  0.00078082  1.13753394
 ch11:  1.13752466  0.00105739  1.13785791
 ch12:  1.13686886  0.00154376  1.13671683
 ch13:  1.13768282 -0.00008225  1.13752552
 ch14:  1.13702987  0.00007272  1.13759905
 ch15:  1.13728815  0.00023723  1.13729101
 ch16:  1.13857680  0.00011683  1.13840102
 ch17:  1.13813042 -0.00044107  1.13833145
 ch18:  1.13823962  0.00194907  1.13788610
 ch19:  1.13829327  0.00137329  1.13824026
 ch20:  1.13850332  0.00036001  1.13853591
 ch21:  1.13792067  0.00034928  1.13806604
 ch22:  1.13850390 -0.00054002  1.13857194
 ch23:  1.13796982  0.00003457  1.13769358
 ch24:  1.13748415  0.00031710  1.13751006
 ch25:  1.13714905 -0.00077724  1.13708145
 ch26:  1.13738782  0.00061631  1.13714448
 ch27:  1.13729314 -0.00051141  1.13764725
 ch28:  1.13671837  0.00062704  1.13680499
 ch29:  1.13711190  0.00035882  1.13709227
 ch30:  1.13692011 -0.00025511  1.13711870
 ch31:  1.13708468  0.00056863  1.13709323
```

The *force* variable can be set to either *CCURDSCC_TRUE* or *CCURDSCC_FALSE*. This API validates the CRC read from the serial prom against what it was expecting and if there is a mismatch and the *force* variable is set to *CCURDSOCC_FALSE,* the call will fail.

2.2.13 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)
 ****************************************************************************/
```

2.2.14 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)
 ****************************************************************************/
```

2.2.15 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)
 ****************************************************************************/
```

2.2.16 ccurDSCC_Fast_Memcpy()

The purpose of this call is to provide a fast mechanism to copy between hardware and memory using programmed I/O. The library performs appropriate locking while the copying is taking place.

```
/****************************************************************************
   ccurDSCC_Fast_Memcpy()

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

   Input:      void         *Handle       (handle pointer)
               volatile void *Source      (pointer to source buffer)
               int            SizeInBytes (transfer size in bytes)
   Output:     volatile void *Destination (pointer to destination buffer)
```

```
   Return:      _ccurdscc_lib_error_number_t
                  - CCURDSCC_LIB_NO_ERROR     (successful)
                  - CCURDSCC_LIB_BAD_HANDLE   (no/bad handler supplied)
                  - CCURDSCC_LIB_NOT_OPEN     (device not open)
 ********************************************************************/
```

2.2.17 ccurDSCC_Fast_Memcpy_Unlocked()

The purpose of this call is to provide a fast mechanism to copy between hardware and memory using programmed I/O. The library does not perform any locking. User needs to provide external locking instead.

```
/********************************************************************
   ccurDSCC_Fast_Memcpy_Unlocked()

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

   Input:     volatile void *Source      (pointer to source buffer)
              int            SizeInBytes (transfer size in bytes)
   Output:    volatile void *Destination (pointer to destination buffer)
   Return:    None
 ********************************************************************/
```

2.2.18 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)
 *********************************************************************/
```

2.2.19 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 external_clock_detected;/* external clock detected */
```

```
    int data_format;            /* data format selection */
    int external_clock_output;  /* external clock selection */
    int identify_board;         /* identify board */
} ccurdscc_board_csr_t;

// external_clock_detected
- CCURDSCC_BCSR_EXTCLK_NOT_DETECTED
- CCURDSCC_BCSR_EXTCLK_DETECTED

// 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

// identify_board
- CCURDSCC_BCSR_IDENTIFY_BOARD_DISABLE
- CCURDSCC_BCSR_IDENTIFY_BOARD_ENABLE
```

2.2.20 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* or *0x9278* with a board type of *0x2*.

```
/*************************************************************************
   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 {
    u_int32_t   board_serial_number;   /* 0x000 - 0x003 - serial number */
    u_short     sprom_revision;        /* 0x004 - 0x005 - serial prom revision */
    u_short     spare_006_03F[0x3A/2]; /* 0x006 - 0x03F - spare */
} ccurdscc_sprom_header_t;

typedef struct {
    uint    board_id;                   /* board id */
    uint    board_type;                 /* board type */
    uint    firmware_rev;               /* firmware revision */
    ccurdscc_sprom_header_t sprom_header;/* sprom header */
    int     number_of_channels;         /* number of channels in this board */
    int     number_of_converters;       /* number of converters in this board */
    double  input_voltage_range;        /* input voltage range */
    double  cal_ref_voltage;            /* calibration reference voltage */
    double  MinSampleFreq;              /* minimum sample frequency */
    double  MaxSampleFreq;              /* maximum sample frequency */
    double  MasterClock;                /* master clock */
} ccurdscc_board_info_t;
```

2.2.21 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)

2.2.22 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_converter_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
```

2.2.23 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;
```

2.2.24 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;
```

2.2.25 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;
```

2.2.26 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;
```

2.2.27 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,
};
```

2.2.28 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]
                -- ccurdscc_sprom_header_t sprom_header;

   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;
```

```
typedef struct {
    u_int32_t   board_serial_number;   /* 0x000 - 0x003 - serial number */
    u_short     sprom_revision;        /* 0x004 - 0x005 - serial prom revision */
    u_short     spare_006_03F[0x3A/2]; /* 0x006 - 0x03F - spare */
} ccurdscc_sprom_header_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_sprom_header_t sprom_header;

} ccurdscc_driver_info_t;
```

2.2.29 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_DEMAND_DMA_FIFO,
    CCURDSCC_DEMAND_DMA_CONTINUOUS,
}  CCURDSCC_DRIVER_READ_MODE;
#define _CCURDSCC_READ_MODE_LIST_FIRST  CCURDSCC_PIO_CHANNEL
#define _CCURDSCC_READ_MODE_LIST_LAST   CCURDSCC_DEMAND_DMA_CONTINUOUS
```

2.2.30 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)
 *****************************************************************************/
```

2.2.31 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_fifo_info_t   *fifo    (pointer to fifo 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 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.

2.2.32 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
```

2.2.33 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
```

2.2.34 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)
 *****************************************************************************/
```

2.2.35 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;
```

Possible library errors:

```
CCURDSCC_LIB_NO_ERROR                   0  /* successful */
CCURDSCC_LIB_INVALID_ARG               -1  /* invalid argument */
CCURDSCC_LIB_ALREADY_OPEN              -2  /* already open */
CCURDSCC_LIB_OPEN_FAILED               -3  /* open failed */
CCURDSCC_LIB_BAD_HANDLE                -4  /* bad handle */
CCURDSCC_LIB_NOT_OPEN                  -5  /* device not opened */
CCURDSCC_LIB_MMAP_SELECT_FAILED        -6  /* mmap selection failed */
CCURDSCC_LIB_MMAP_FAILED               -7  /* mmap failed */
CCURDSCC_LIB_MUNMAP_FAILED             -8  /* munmap failed */
CCURDSCC_LIB_NOT_MAPPED                -9  /* not mapped */
CCURDSCC_LIB_ALREADY_MAPPED           -10 /* already mapped */
CCURDSCC_LIB_IOCTL_FAILED             -11 /* driver ioctl failed */
CCURDSCC_LIB_IO_ERROR                 -12 /* i/o error */
CCURDSCC_LIB_INTERNAL_ERROR           -13 /* internal library error */
CCURDSCC_LIB_NOT_IMPLEMENTED          -14 /* call not implemented */
CCURDSCC_LIB_LOCK_FAILED              -15 /* failed to get lib lock */
CCURDSCC_LIB_NO_LOCAL_REGION          -16 /* local region not present */
CCURDSCC_LIB_NO_CONFIG_REGION         -17 /* config region not present */
CCURDSCC_LIB_NO_SOLUTION_FOUND        -18 /* no solution found */
CCURDSCC_LIB_CONVERTER_RESET          -19 /* converter not active */
CCURDSCC_LIB_NO_RESOURCE              -20 /* resource not available */
CCURDSCC_LIB_CALIBRATION_RANGE_ERROR  -21 /* calibration voltage out of range */
CCURDSCC_LIB_FIFO_OVERFLOW            -22 /* fifo overflow */
CCURDSCC_LIB_CANNOT_OPEN_FILE         -23 /* cannot open file */
CCURDSCC_LIB_BAD_DATA_IN_CAL_FILE     -24 /* bad date in calibration file */
CCURDSCC_LIB_SERIAL_PROM_BUSY         -25 /* serial prom busy */
CCURDSCC_LIB_SERIAL_PROM_FAILURE      -26 /* serial prom failure - malfunction
                                             or not present */
CCURDSCC_LIB_INVALID_CRC              -27 /* invalid CRC read */
CCURDSCC_LIB_SERIAL_PROM_WRITE_PROTECTED -28 /* serial prom is write protected */
CCURDSCC_LIB_PTHREAD_CREATE_FAILURE   -29 /* pthread() create failure */
CCURDSCC_LIB_AUTOCAL_IN_PROGRESS      -30 /* autocal in progress */
```

2.2.36 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)
 ************************************************************************/
```

2.2.37 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)
 ************************************************************************/
```

2.2.38 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* or *CCURDSCC_DEMAND_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)
 ***************************************************************************/
```

2.2.39 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)
 ***************************************************************************/
```

2.2.40 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;
```

2.2.41 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;
```

2.2.42 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
```

2.2.43 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
```

2.2.44 ccurDSCC_TestBus_Control()

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

```
/*************************************************************************
   ccurDSCC_Get_TestBus_Control()

   Description: Return the value of the Test Bus control information

   Input:       void                        *Handle       (handle pointer)
   Output:      _ccurdscc_testbus_control_t *test_control (pointer to pll sync
                                                           struct)
   Return:      CCURDSCC_LIB_NO_ERROR                     (successful)
                CCURDSCC_LIB_NO_LOCAL_REGION              (local region error)
                CCURDSCC_LIB_BAD_HANDLE                   (no/bad handler
                                                           supplied)
                CCURDSCC_LIB_NOT_OPEN                     (device not open)
 *************************************************************************/

typedef enum
{
    CCURDSCC_TBUS_CONTROL_OPEN = (0),
    CCURDSCC_TBUS_CONTROL_CAL_BUS = (1),
    CCURDSCC_TBUS_CONTROL_5V_REF = (2),
} _ccurdscc_testbus_control_t;
```

2.2.45 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_TEST_BUS_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_CHANNEL_DATA_0,                /* R/W */
    CCURDSCC_CHANNEL_DATA_1,                /* R/W */
    CCURDSCC_CHANNEL_DATA_2,                /* R/W */
    CCURDSCC_CHANNEL_DATA_3,                /* R/W */
    CCURDSCC_CHANNEL_DATA_4,                /* R/W */
    CCURDSCC_CHANNEL_DATA_5,                /* R/W */
    CCURDSCC_CHANNEL_DATA_6,                /* R/W */
    CCURDSCC_CHANNEL_DATA_7,                /* R/W */
    CCURDSCC_CHANNEL_DATA_8,                /* R/W */
    CCURDSCC_CHANNEL_DATA_9,                /* R/W */
    CCURDSCC_CHANNEL_DATA_10,               /* R/W */
    CCURDSCC_CHANNEL_DATA_11,               /* R/W */
    CCURDSCC_CHANNEL_DATA_12,               /* R/W */
    CCURDSCC_CHANNEL_DATA_13,               /* R/W */
    CCURDSCC_CHANNEL_DATA_14,               /* R/W */
    CCURDSCC_CHANNEL_DATA_15,               /* R/W */
    CCURDSCC_CHANNEL_DATA_16,               /* R/W */
    CCURDSCC_CHANNEL_DATA_17,               /* R/W */
    CCURDSCC_CHANNEL_DATA_18,               /* R/W */
    CCURDSCC_CHANNEL_DATA_19,               /* R/W */
    CCURDSCC_CHANNEL_DATA_20,               /* R/W */
    CCURDSCC_CHANNEL_DATA_21,               /* R/W */
    CCURDSCC_CHANNEL_DATA_22,               /* R/W */
    CCURDSCC_CHANNEL_DATA_23,               /* R/W */
    CCURDSCC_CHANNEL_DATA_24,               /* R/W */
    CCURDSCC_CHANNEL_DATA_25,               /* R/W */
    CCURDSCC_CHANNEL_DATA_26,               /* R/W */
    CCURDSCC_CHANNEL_DATA_27,               /* R/W */
    CCURDSCC_CHANNEL_DATA_28,               /* R/W */
    CCURDSCC_CHANNEL_DATA_29,               /* R/W */
    CCURDSCC_CHANNEL_DATA_30,               /* R/W */
    CCURDSCC_CHANNEL_DATA_31,               /* R/W */

    CCURDSCC_FIFO_DATA,                     /* R Only */

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

    CCURDSCC_OFFSET_CALIBRATION,            /* R/W */

    CCURDSCC_SPROM_STAT_ADDR_WRITE_DATA,    /* R/W */
```

```
    CCURDSCC_SPROM_READ_DATA,               /* R Only */

    CCURDSCC_SPI_RAM,                       /* R/W */

} CCURDSCC_CONTROL;
```

2.2.46 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)
 *****************************************************************************/
```

2.2.47 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)
 *****************************************************************************/
```

2.2.48 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;
```

2.2.49 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* or *CCURDSCC_DEMAND_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)
 ************************************************************************/
```

2.2.50 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)
 ************************************************************************/
```

2.2.51 ccurDSCC_NanoDelay()

This call simply delays (loops) for user specified nano–seconds. .

```
/************************************************************************
   void ccurDSCC_NanoDelay(unsigned long long NanoDelay)

   Description: Delay )loop for user specified nano-seconds.

   Input:       unsigned long long NanoDelay      (number of nano-secs to delay)
   Output:      None
   Return:      None

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

2.2.52 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.

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

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

2.2.53 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 15 to 60 seconds *(depending on the board)* 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)
                CCURDSCC_LIB_AUTOCAL_IN_PROGRESS (auto calibration in progress)
 ************************************************************************/
```

2.2.54 ccurDSCC_Perform_Auto_Calibration_Async()

This call is identical to the *ccurDSCC_Perform_Auto_Calibration()* call, except that is run in the background. Its progress can be monitored with the *ccurDSCC_AutoCal_Status()* call. Until this call is complete, the user should not make any attempt to access the cards, otherwise, the call could fail.

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

   Description: Perform Auto Calibration Asynchronously

   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)
                CCURDSCC_LIB_AUTOCAL_IN_PROGRESS (auto calibration in progress)
 ************************************************************************/
```

2.2.55 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)
 ************************************************************************/
```

2.2.56 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)
 ************************************************************************/
```

2.2.57 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)
 ************************************************************************/
```

2.2.58 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)
 ************************************************************************/
```

2.2.59 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)
 ************************************************************************/
```

2.2.60 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)
 ************************************************************************/
```

2.2.61 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;
```

2.2.62 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;
```

2.2.63 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;
```

2.2.64 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)
 ************************************************************************/
```

2.2.65 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;
```

2.2.66 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
```

2.2.67 ccurDSCC_Read_Serial_Prom()

This is a basic call to read short word entries from the serial prom. The user specifies a word offset within the serial prom and a word count, and the call returns the data read in a pointer to short words.

```
/*****************************************************************************
   ccurDSCC_Read_Serial_Prom()

   Description: Read Serial Prom for specified number of words

   Input:       void                    *Handle   (handle pointer)
                ccurdscc_sprom_rw_t     *spr      (pointer to struct)
                  -- u_short word_offset
                  -- u_short num_words
   Output:      ccurdscc_sprom_rw_t     *spr      (pointer to struct)
                  -- u_short *data_ptr
   Return:      CCURDSCC_LIB_NO_ERROR             (successful)
                CCURDSCC_LIB_NO_LOCAL_REGION      (error)
                CCURDSCC_LIB_INVALID_ARG          (invalid argument)
                CCURDSCC_LIB_SERIAL_PROM_BUSY     (serial prom busy)
                CCURDSCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
 *****************************************************************************/

typedef struct
{
    u_short word_offset;    /* word offset */
    u_short num_words;      /* number of words */
    u_short *data_ptr;      /* data pointer */
} ccurdscc_sprom_rw_t;
```

2.2.68 ccurDSCC_Read_Serial_Prom_Item()

This call is used to read well defined sections in the serial prom. The user supplies the serial prom section that needs to be read and the data is returned in a section specific structure.

```
/*****************************************************************************
   ccurDSCC_Read_Serial_Prom_Item()

   Description: Read Serial Prom for specified item

   Input:       void                    *Handle   (handle pointer)
                _ccurdscc_sprom_access_t item     (select item)
                  -- CCURDSCC_SPROM_HEADER
                  -- CCURDSCC_SPROM_FACTORY
                  -- CCURDSCC_SPROM_USER_CHECKPOINT_1
                  -- CCURDSCC_SPROM_USER_CHECKPOINT_2
   Output:      void                    *item_ptr (pointer to item struct)
                  -- *ccurdscc_sprom_header_t
                  -- *ccurdscc_sprom_factory_t
                  -- *ccurdscc_sprom_user_checkpoint_t
   Return:      CCURDSCC_LIB_NO_ERROR             (successful)
                CCURDSCC_LIB_NO_LOCAL_REGION      (error)
                CCURDSCC_LIB_INVALID_ARG          (invalid argument)
                CCURDSCC_LIB_SERIAL_PROM_BUSY     (serial prom busy)
                CCURDSCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
 *****************************************************************************/

typedef enum {
    CCURDSCC_SPROM_HEADER=1,
    CCURDSCC_SPROM_FACTORY,
    CCURDSCC_SPROM_USER_CHECKPOINT_1,
```

```
    CCURDSCC_SPROM_USER_CHECKPOINT_2,
} _ccurdscc_sprom_access_t;

typedef struct {
    u_int32_t   board_serial_number;   /* 0x000 - 0x003 - serial number */
    u_short     sprom_revision;        /* 0x004 - 0x005 - serial prom revision */
    u_short     spare_006_03F[0x3A/2]; /* 0x006 - 0x03F - spare */
} ccurdscc_sprom_header_t;

#define ccurdscc_sprom_factory_t            __ccurdscc_sprom_common_t
#define ccurdscc_sprom_user_checkpoint_t    __ccurdscc_sprom_common_t

typedef struct {
    u_int32_t CPM_csr;                 /* converter status/control register */
    u_int32_t CPM_read_1;              /* converter CPM Read Register 1 */
    u_int32_t CPM_read_2;              /* converter CPM Read Register 2 */
} _CPM_Checkpoint_t;

typedef struct {
    u_int32_t PLL_read_1;              /* PLL Read Register 1 */
    u_int32_t PLL_read_2;              /* PLL Read Register 2 */
} _PLL_Checkpoint_t;

typedef struct {
    u_short             crc;                                    /* 0x000 - 0x001 - CRC */
    u_short             spare_002_007[0x6/2];                   /* 0x002 - 0x007 - spare */
    union {
        time_t          date;                                   /* 0x008 - 0x00F - date */
        u_int32_t       date_storage[2]; /* for 32/64bit m/c */ /* 0x008 - 0x00F - date */
    };
    u_int32_t           spare_010_01B[0xC/4];                   /* 0x010 - 0x01B - spare */
    u_int32_t           PLL_Checkpoint_sync;                    /* 0x01C - 0x01F - PLL_sync
                                                                                        */
    double              last_specified_fRef;                    /* 0x020 - 0x027 - Reference
                                                                                  Freq */
    double              ExternalClock_actual_freq;              /* 0x028 - 0x02F - Ext Clock
                                                                             Actual Freq */
    _CPM_Checkpoint_t   CPM_Checkpoint[CCURDSCC_MAX_CONVERTERS];/* 0x030 - 0x05F - Converter
                                                                                        */
    _PLL_Checkpoint_t   PLL_Checkpoint[CCURDSCC_MAX_PLLS];      /* 0x060 - 0x07F - PLL */
    u_int32_t           offset[CCURDSCC_MAX_CHANNELS];          /* 0x080 - 0x0FF - offset */
    u_int32_t           positive[CCURDSCC_MAX_CHANNELS];        /* 0x100 - 0x17F - positive
                                                                                        */
    u_int32_t           negative[CCURDSCC_MAX_CHANNELS];        /* 0x180 - 0x1FF - offset */
} __ccurdscc_sprom_common_t;
```

2.2.69 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* or *CCURDSCC_DEMAND_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)
 *****************************************************************************/
```

2.2.70 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)
 ************************************************************************/
```

2.2.71 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)
 ************************************************************************/
```

2.2.72 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
```

2.2.73 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)
 ***************************************************************************/
```

2.2.74 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)
 ***************************************************************************/
```

2.2.75 ccurDSCC_Restore_Factory_Calibration()

This API allows the user to reset the board to factory calibration values, located in the serial prom, for all the channels. The API selects the corresponding factory calibration based on the converter and clock settings that was previously configured by the user. It provides a useful way to make sure that each channel is working with the factory calibration without the need to perform an auto-calibration.

```
/***************************************************************************
   ccurDSCC_Restore_Factory_Calibration()

   Description: Restore Factory board calibration from serial prom
```

```
   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    (local region not present)
                CCURDSCC_LIB_INVALID_CRC        (invalid CRC)
 ****************************************************************************/
```

2.2.76 ccurDSCC_Restore_User_Checkpoint()

This API allows the user to reset the board to previously created checkpoint values, located in the serial prom, for all the channels. The API sets the converter and clock configurations and calibration information for all the channels that were previously created by the user. It provides a useful way to make sure that each channel is working with user defined channel configuration and calibration without the need to perform an auto-calibration. The user can select any of two checkpoints to create and restore.

```
/****************************************************************************
ccurDSCC_Restore_User_Checkpoint()

   Description: Restore User Checkpoint from serial prom

   Input:       void                    *Handle   (handle pointer)
                _ccurdscc_sprom_access_t item     (select item)
                  -- CCURDSCC_SPROM_USER_CHECKPOINT_1
                  -- CCURDSCC_SPROM_USER_CHECKPOINT_2
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_NO_LOCAL_REGION     (error)
                CCURDSCC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
                CCURDSCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
                CCURDSCC_LIB_INVALID_CRC         (invalid CRC)
 ****************************************************************************/

typedef enum {
    CCURDSCC_SPROM_HEADER=1,
    CCURDSOCC_SPROM_FACTORY,
    CCURDSCC_SPROM_USER_CHECKPOINT_1,
    CCURDSCC_SPROM_USER_CHECKPOINT_2,
} _ccurdscc_sprom_access_t;
```

2.2.77 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_DEMAND_DMA_FIFO,
    CCURDSCC_DEMAND_DMA_CONTINUOUS,
}  CCURDSCC_DRIVER_READ_MODE;
#define _CCURDSCC_READ_MODE_LIST_FIRST  CCURDSCC_PIO_CHANNEL
#define _CCURDSCC_READ_MODE_LIST_LAST   CCURDSCC_DEMAND_DMA_CONTINUOUS
```

2.2.78 ccurDSCC_Serial_Prom_Write_Override()

The serial prom is non-volatile and its information is preserved during a power cycle. It contains useful information and settings that the customer could lose if they were to inadvertently overwrite. For this reason, all calls that write to the serial proms will fail with a write protect error, unless this write protect override API is invoked prior to writing to the serial proms. Once the Write Override is enabled, it will stay in effect until the user closes the device or re-issues this call to disable writes to the serial prom.

The calls that will fail unless the write protect is disabled are:

- ccurDSCC_Create_Factory_Calibration()
- ccurDSCC_Create_User_Checkpoint()
- ccurDSCC_Write_Serial_Prom()
- ccurDSCC_Write_Serial_Prom_Item()

```
/***************************************************************************
   ccurDSCC_Serial_Prom_Write_Override()

   Description: Set Serial Prom Write Override

   Input:       void            *Handle (handle pointer)
                int             action; (override action)
                  -- CCURDSCC_TRUE
                  -- CCURDSCC_FALSE
   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)
 ***************************************************************************/
```

2.2.79 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 external_clock_detected;/* external clock detected */
    int data_format;            /* data format selection */
    int external_clock_output;  /* external clock selection */
    int identify_board;         /* identify board */
} ccurdscc_board_csr_t;

// external_clock_detected
- CCURDSCC_BCSR_EXTCLK_NOT_DETECTED
- CCURDSCC_BCSR_EXTCLK_DETECTED

// 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

// identify_board
- CCURDSCC_BCSR_IDENTIFY_BOARD_DISABLE
- CCURDSCC_BCSR_IDENTIFY_BOARD_ENABLE
```

2.2.80 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
```

2.2.81 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
```

2.2.82 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;
```

2.2.83 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;
```

2.2.84 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;
```

2.2.85 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)
 ****************************************************************************/
```

2.2.86 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)
```

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

2.2.87 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
```

2.2.88 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
```

2.2.89 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)
 *****************************************************************************/
```

2.2.90 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
```

2.2.91 ccurDSCC_Set_TestBus_Control()

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

```
/*************************************************************************
   ccurDSCC_Set_TestBus_Control()

   Description: Set Test Bus Control Selection

   Input:       void                          *Handle      (handle pointer)
                _ccurdscc_testbus_control_t test_control (pointer to test bus
                                                           control)
   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_TBUS_CONTROL_OPEN = (0),
    CCURDSCC_TBUS_CONTROL_CAL_BUS = (1),
    CCURDSCC_TBUS_CONTROL_5V_REF = (2),
} _ccurdscc_testbus_control_t;
```

2.2.92 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)
                void             *value         (pointer to value 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)
 *************************************************************************/

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_TEST_BUS_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_CHANNEL_DATA_0,                /* R/W */
CCURDSCC_CHANNEL_DATA_1,                /* R/W */
CCURDSCC_CHANNEL_DATA_2,                /* R/W */
CCURDSCC_CHANNEL_DATA_3,                /* R/W */
CCURDSCC_CHANNEL_DATA_4,                /* R/W */
CCURDSCC_CHANNEL_DATA_5,                /* R/W */
CCURDSCC_CHANNEL_DATA_6,                /* R/W */
```

```
    CCURDSCC_CHANNEL_DATA_7,                /* R/W */
    CCURDSCC_CHANNEL_DATA_8,                /* R/W */
    CCURDSCC_CHANNEL_DATA_9,                /* R/W */
    CCURDSCC_CHANNEL_DATA_10,               /* R/W */
    CCURDSCC_CHANNEL_DATA_11,               /* R/W */
    CCURDSCC_CHANNEL_DATA_12,               /* R/W */
    CCURDSCC_CHANNEL_DATA_13,               /* R/W */
    CCURDSCC_CHANNEL_DATA_14,               /* R/W */
    CCURDSCC_CHANNEL_DATA_15,               /* R/W */
    CCURDSCC_CHANNEL_DATA_16,               /* R/W */
    CCURDSCC_CHANNEL_DATA_17,               /* R/W */
    CCURDSCC_CHANNEL_DATA_18,               /* R/W */
    CCURDSCC_CHANNEL_DATA_19,               /* R/W */
    CCURDSCC_CHANNEL_DATA_20,               /* R/W */
    CCURDSCC_CHANNEL_DATA_21,               /* R/W */
    CCURDSCC_CHANNEL_DATA_22,               /* R/W */
    CCURDSCC_CHANNEL_DATA_23,               /* R/W */
    CCURDSCC_CHANNEL_DATA_24,               /* R/W */
    CCURDSCC_CHANNEL_DATA_25,               /* R/W */
    CCURDSCC_CHANNEL_DATA_26,               /* R/W */
    CCURDSCC_CHANNEL_DATA_27,               /* R/W */
    CCURDSCC_CHANNEL_DATA_28,               /* R/W */
    CCURDSCC_CHANNEL_DATA_29,               /* R/W */
    CCURDSCC_CHANNEL_DATA_30,               /* R/W */
    CCURDSCC_CHANNEL_DATA_31,               /* R/W */

    CCURDSCC_FIFO_DATA,                     /* R Only */

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

    CCURDSCC_OFFSET_CALIBRATION,            /* R/W */

    CCURDSCC_SPROM_STAT_ADDR_WRITE_DATA,    /* R/W */
    CCURDSCC_SPROM_READ_DATA,               /* R Only */

    CCURDSCC_SPI_RAM,                       /* R/W */

} CCURDSCC_CONTROL;
```

2.2.93 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)
 *****************************************************************************/
```

2.2.94 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
```

2.2.95 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)
 ************************************************************************/
```

2.2.96 ccurDSCC_View_Factory_Calibration()

This API extracts the factory serial prom calibration information for the selected voltage range and writes it to a user specified file.

```
/************************************************************************
   ccurDSCC_View_Factory_Calibration()

   Description: Read Factory calibration from serial prom and write to user
                output file

   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_CANNOT_OPEN_FILE   (file not readable)
```

```
                CCURDSCC_LIB_NO_LOCAL_REGION     (error)
                CCURDSCC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
                CCURDSCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
 ***************************************************************************/
```

2.2.97 ccurDSCC_View_User_Checkpoint()

This API extracts the user serial prom configuration and calibration information for the selected user checkpoint and writes it to a user specified file.

```
/***************************************************************************
   ccurDSCC_View_User_Checkpoint()

   Description: Read User Checkpoint from serial prom and write to user output
                file

   Input:       void                    *Handle   (handle pointer)
                _ccurdscc_sprom_access_t item     (select item)
                  -- CCURDSCC_SPROM_USER_CHECKPOINT_1
                  -- CCURDSCC_SPROM_USER_CHECKPOINT_2
   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_CANNOT_OPEN_FILE    (file not readable)
                CCURDSCC_LIB_NO_LOCAL_REGION     (error)
                CCURDSCC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
                CCURDSCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
 ***************************************************************************/

typedef enum {
    CCURDSCC_SPROM_HEADER=1,
    CCURDSCC_SPROM_FACTORY,
    CCURDSCC_SPROM_USER_CHECKPOINT_1,
    CCURDSCC_SPROM_USER_CHECKPOINT_2,
} _ccurdscc_sprom_access_t;
```

2.2.98 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
```

2.2.99 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
```

2.2.100 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)
 ************************************************************************/
```

2.2.101 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
```

2.2.102 ccurDSCC_Write_Serial_Prom()

This is a basic call to write short word entries to the serial prom. The user specifies a word offset within the serial prom and a word count, and the call writes the data pointed to by the *spw* pointer, in short words.

Prior to using this call, the user will need to issue the *ccurDSCC_Serial_Prom_Write_Override()* to allowing writing to the serial prom.

```
/****************************************************************************
   ccurDSCC_Write_Serial_Prom()

   Description: Write data to Serial Prom for specified number of words

   Input:       void                   *Handle   (handle pointer)
                ccurdscc_sprom_rw_t    *spw      (pointer to struct)
                  -- u_short word_offset
                  -- u_short num_words
                  -- u_short *data_ptr
   Output:      none
   Return:      CCURDSCC_LIB_NO_ERROR             (successful)
                CCURDSCC_LIB_NO_LOCAL_REGION      (error)
                CCURDSCC_LIB_INVALID_ARG          (invalid argument)
                CCURDSCC_LIB_SERIAL_PROM_BUSY     (serial prom busy)
                CCURDSCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
 ****************************************************************************/

typedef struct
{
    u_short word_offset;    /* word offset */
    u_short num_words;      /* number of words */
    u_short *data_ptr;      /* data pointer */
} ccurdscc_sprom_rw_t;
```

2.2.103 ccurDSCC_Write_Serial_Prom _Item()

This call is used to write well defined sections in the serial prom. The user supplies the serial prom section that needs to be written and the data points to the section specific structure. In the case of factory calibration or user checkpoint writes, the user needs to make sure that the time stamp and crc are setup correctly, otherwise, there will be problems in viewing the section. This call should normally not be used by the user.

Prior to using this call, the user will need to issue the *ccurDSCC_Serial_Prom_Write_Override()* to allowing writing to the serial prom.

```
/*************************************************************************
   ccurDSCC_Write_Serial_Prom_Item()

   Description: Write Serial Prom with specified item

   Input:       void                    *Handle   (handle pointer)
                _ccurdscc_sprom_access_t item     (select item)
                  -- CCURDSCC_SPROM_HEADER
                  -- CCURDSCC_SPROM_FACTORY
                  -- CCURDSCC_SPROM_USER_CHECKPOINT_1
                  -- CCURDSCC_SPROM_USER_CHECKPOINT_2
   Output:      void                    *item_ptr (pointer to item struct)
                  -- *ccurdscc_sprom_header_t
                  -- *ccurdscc_sprom_factory_t
                  -- *ccurdscc_sprom_user_checkpoint_t
   Return:      CCURDSCC_LIB_NO_ERROR            (successful)
                CCURDSCC_LIB_NO_LOCAL_REGION     (error)
                CCURDSCC_LIB_INVALID_ARG         (invalid argument)
                CCURDSCC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
                CCURDSCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
 *************************************************************************/

typedef enum {
    CCURDSCC_SPROM_HEADER=1,
    CCURDSCC_SPROM_FACTORY,
    CCURDSCC_SPROM_USER_CHECKPOINT_1,
    CCURDSCC_SPROM_USER_CHECKPOINT_2,
} _ccurdscc_sprom_access_t;

typedef struct {
    u_int32_t   board_serial_number;   /* 0x000 - 0x003 - serial number */
    u_short     sprom_revision;        /* 0x004 - 0x005 - serial prom revision */
    u_short     spare_006_03F[0x3A/2]; /* 0x006 - 0x03F - spare */
} ccurdscc_sprom_header_t;

#define ccurdscc_sprom_factory_t            __ccurdscc_sprom_common_t
#define ccurdscc_sprom_user_checkpoint_t    __ccurdscc_sprom_common_t

typedef struct {
    u_int32_t CPM_csr;                 /* converter status/control register */
    u_int32_t CPM_read_1;              /* converter CPM Read Register 1 */
    u_int32_t CPM_read_2;              /* converter CPM Read Register 2 */
} _CPM_Checkpoint_t;

typedef struct {
    u_int32_t PLL_read_1;              /* PLL Read Register 1 */
    u_int32_t PLL_read_2;              /* PLL Read Register 2 */
} _PLL_Checkpoint_t;

typedef struct {
    u_short            crc;                                  /* 0x000 - 0x001 - CRC */
    u_short            spare_002_007[0x6/2];                 /* 0x002 - 0x007 - spare */
```

```
    union {
        time_t          date;                                   /* 0x008 - 0x00F - date */
        u_int32_t       date_storage[2]; /* for 32/64bit m/c */ /* 0x008 - 0x00F - date */
    };
    u_int32_t           spare_010_01B[0xC/4];                   /* 0x010 - 0x01B - spare */
    u_int32_t           PLL_Checkpoint_sync;                    /* 0x01C - 0x01F - PLL_sync
                                                                                        */
    double              last_specified_fRef;                    /* 0x020 - 0x027 - Reference
                                                                                  Freq */
    double              ExternalClock_actual_freq;              /* 0x028 - 0x02F - Ext Clock
                                                                             Actual Freq */
    _CPM_Checkpoint_t   CPM_Checkpoint[CCURDSCC_MAX_CONVERTERS];/* 0x030 - 0x05F - Converter
                                                                                        */
    _PLL_Checkpoint_t   PLL_Checkpoint[CCURDSCC_MAX_PLLS];      /* 0x060 - 0x07F - PLL */
    u_int32_t           offset[CCURDSCC_MAX_CHANNELS];          /* 0x080 - 0x0FF - offset */
    u_int32_t           positive[CCURDSCC_MAX_CHANNELS];        /* 0x100 - 0x17F - positive
                                                                                        */
    u_int32_t           negative[CCURDSCC_MAX_CHANNELS];        /* 0x180 - 0x1FF - offset */
} __ccurdscc_sprom_common_t;
```

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])
 -mW                    (Driver Demand DMA read mode [DEMAND FIFO])
 -N                     (Open device with O_NONBLOCK flag)
 -p                     (Program board to max clock first)
```

Example display:

./ccurdscc_disp *(9277- 5volt board)*

```
Board Number       [-b]: 0  ==> '/dev/ccurdscc0'
Board Serial Number    : 4294967295 (0xffffffff)
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
```

./ccurdscc_disp *(9278- 10volt board)*

```
Board Number       [-b]: 0  ==> '/dev/ccurdscc0'
Board Serial Number    : 4294967295 (0xffffffff)
Delay              [-d]: 0 milli-seconds
Data Format        [-f]: 'Offset Binary'
Read Mode          [-m]: 'User PIO (Channel Data)'
Program Board      [-p]: 'No'
Input Voltage Range    : +/-10.0 Volts
Read Error?            : '=== no ==='

Scan count:      77165, Delta:  36.9 usec (min= 36.8,max= 45.6,av= 37.1)

         ##### Raw Data #####
         [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]
         ======   ======   ======   ======   ======   ======   ======   ======
Conv[0]  7ffe07   7ffe88   7fff80   7ffea7   800064   7fff2b   80001d   8000b0

Conv[1]  8002b8   7fff84   7ffbf0   80018b   7ffd69   80000c   800141   7fffb2

Conv[2]  8002a5   7ffeec   80018a   80027d   80022d   7ffeba   7ffdec   800045

Conv[3]  7ffb98   8000dc   7fff1b   80013b   800375   7ffeff   8001ba   7fffc6



         ##### Volts #####
         [0]       [1]       [2]       [3]       [4]       [5]       [6]       [7]
         ========  ========  ========  ========  ========  ========  ========  ========
Conv[0]  -0.00060  -0.00045  -0.00015  -0.00041  +0.00012  -0.00025  +0.00003  +0.00021

Conv[1]  +0.00083  -0.00015  -0.00124  +0.00047  -0.00079  +0.00001  +0.00038  -0.00009

Conv[2]  +0.00081  -0.00033  +0.00047  +0.00076  +0.00066  -0.00039  -0.00063  +0.00008

Conv[3]  -0.00134  +0.00026  -0.00027  +0.00038  +0.00106  -0.00031  +0.00053  -0.00007
```

3.1.2 ccurdscc_dump

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

Usage: ccurdscc_dump [-b board]

Example display:

```
Device Name    : /dev/ccurdscc0
Board Serial No: 4294967295 (0xffffffff)

LOCAL Register 0x7ffff7ffb000 Offset=0x0
CONFIG Register 0x7ffff7ffa000 Offset=0x0

======= LOCAL BOARD REGISTERS =========
LBR: @0x0000 --> 0x92780201
LBR: @0x0004 --> 0x00000300
LBR: @0x0008 --> 0x00000000
LBR: @0x000c --> 0x00000000
LBR: @0x0010 --> 0x00000001
LBR: @0x0014 --> 0x00000001
```

```
LBR: @0x0018 --> 0x00000001
LBR: @0x001c --> 0x00000001
LBR: @0x0020 --> 0x00000008
LBR: @0x0024 --> 0x000004ff
LBR: @0x0028 --> 0xffff8681
LBR: @0x002c --> 0x000000ff
LBR: @0x0030 --> 0x00000001
LBR: @0x0034 --> 0x00000001
LBR: @0x0038 --> 0x00000001
LBR: @0x003c --> 0x00000001
LBR: @0x0040 --> 0x00000008
LBR: @0x0044 --> 0x000004ff
LBR: @0x0048 --> 0xffff8681
LBR: @0x004c --> 0x000000ff
LBR: @0x0050 --> 0x00000001
LBR: @0x0054 --> 0x00000001
LBR: @0x0058 --> 0x00000001
LBR: @0x005c --> 0x00000001
LBR: @0x0060 --> 0x00000008
LBR: @0x0064 --> 0x000004ff
LBR: @0x0068 --> 0xffff8681
LBR: @0x006c --> 0x000000ff
LBR: @0x0070 --> 0x00000001
LBR: @0x0074 --> 0x00000001
LBR: @0x0078 --> 0x00000001
LBR: @0x007c --> 0x00000001
LBR: @0x0080 --> 0x00000008
LBR: @0x0084 --> 0x000004ff
LBR: @0x0088 --> 0xffff8681
LBR: @0x008c --> 0x000000ff
LBR: @0x0090 --> 0x00000001
LBR: @0x0094 --> 0x00000001
LBR: @0x0098 --> 0x00000001
LBR: @0x009c --> 0x00000001
LBR: @0x00a0 --> 0x00000000
LBR: @0x00a4 --> 0x00000001
LBR: @0x00a8 --> 0x00000001
LBR: @0x00ac --> 0x00000001
LBR: @0x00b0 --> 0x00000000
LBR: @0x00b4 --> 0x00000000
LBR: @0x00b8 --> 0x00000001
LBR: @0x00bc --> 0x00000001
LBR: @0x00c0 --> 0x81000000
LBR: @0x00c4 --> 0x0000c000
LBR: @0x00c8 --> 0xffffffff
LBR: @0x00cc --> 0x00000001
LBR: @0x00d0 --> 0x00000001
LBR: @0x00d4 --> 0x00000001
LBR: @0x00d8 --> 0x00000001
LBR: @0x00dc --> 0x00000001
LBR: @0x00e0 --> 0x00000001
LBR: @0x00e4 --> 0x00000001
LBR: @0x00e8 --> 0x00000001
LBR: @0x00ec --> 0x00000001
LBR: @0x00f0 --> 0x00000000
LBR: @0x00f4 --> 0x00000000
LBR: @0x00f8 --> 0x00000001
```

```
LBR: @0x00fc --> 0x00000001
LBR: @0x0100 --> 0x00800000
LBR: @0x0104 --> 0x00800000
LBR: @0x0108 --> 0x00800000
LBR: @0x010c --> 0x00800000
LBR: @0x0110 --> 0x00800000
LBR: @0x0114 --> 0x00800000
LBR: @0x0118 --> 0x00800000
LBR: @0x011c --> 0x00800000
LBR: @0x0120 --> 0x00800000
LBR: @0x0124 --> 0x00800000
LBR: @0x0128 --> 0x00800000
LBR: @0x012c --> 0x00800000
LBR: @0x0130 --> 0x00800000
LBR: @0x0134 --> 0x00800000
LBR: @0x0138 --> 0x00800000
LBR: @0x013c --> 0x00800000
LBR: @0x0140 --> 0x00800000
LBR: @0x0144 --> 0x00800000
LBR: @0x0148 --> 0x00800000
LBR: @0x014c --> 0x00800000
LBR: @0x0150 --> 0x00800000
LBR: @0x0154 --> 0x00800000
LBR: @0x0158 --> 0x00800000
LBR: @0x015c --> 0x00800000
LBR: @0x0160 --> 0x00800000
LBR: @0x0164 --> 0x00800000
LBR: @0x0168 --> 0x00800000
LBR: @0x016c --> 0x00800000
LBR: @0x0170 --> 0x00800000
LBR: @0x0174 --> 0x00800000
LBR: @0x0178 --> 0x00800000
LBR: @0x017c --> 0x00800000
LBR: @0x0180 --> 0x00000001
LBR: @0x0184 --> 0x00000001
LBR: @0x0188 --> 0x00000001
LBR: @0x018c --> 0x00000001
LBR: @0x0190 --> 0x00800000
LBR: @0x0194 --> 0x00000001
LBR: @0x0198 --> 0x00000001
LBR: @0x019c --> 0x00000001
LBR: @0x01a0 --> 0x00000000
LBR: @0x01a4 --> 0x00000600
LBR: @0x01a8 --> 0x00000000
LBR: @0x01ac --> 0x00000000
LBR: @0x01b0 --> 0x00000000
LBR: @0x01b4 --> 0x00000600
LBR: @0x01b8 --> 0x00000000
LBR: @0x01bc --> 0x00000000
LBR: @0x01c0 --> 0x00000000
LBR: @0x01c4 --> 0x00000600
LBR: @0x01c8 --> 0x00000000
LBR: @0x01cc --> 0x00000000
LBR: @0x01d0 --> 0x00000000
LBR: @0x01d4 --> 0x00000600
LBR: @0x01d8 --> 0x00000000
LBR: @0x01dc --> 0x00000000
```

```
LBR: @0x01e0 --> 0x00000001
LBR: @0x01e4 --> 0x00000001
LBR: @0x01e8 --> 0x00000001
LBR: @0x01ec --> 0x00000001
LBR: @0x01f0 --> 0x00000001
LBR: @0x01f4 --> 0x00000001
LBR: @0x01f8 --> 0x00000001
LBR: @0x01fc --> 0x00000001
LBR: @0x0200 --> 0x9197f195
LBR: @0x0204 --> 0x9192a27b
LBR: @0x0208 --> 0x919cdee3
LBR: @0x020c --> 0x91a797f9
LBR: @0x0210 --> 0x919c8355
LBR: @0x0214 --> 0x9198bbe5
LBR: @0x0218 --> 0x919863b0
LBR: @0x021c --> 0x919e26c2
LBR: @0x0220 --> 0x919b0f32
LBR: @0x0224 --> 0x91b9b055
LBR: @0x0228 --> 0x91b51630
LBR: @0x022c --> 0x9191cd0d
LBR: @0x0230 --> 0x91b27c6f
LBR: @0x0234 --> 0x919a9f1b
LBR: @0x0238 --> 0x91999662
LBR: @0x023c --> 0x91b9b34e
LBR: @0x0240 --> 0x91b7bd5f
LBR: @0x0244 --> 0x91b3a76b
LBR: @0x0248 --> 0x91ab416e
LBR: @0x024c --> 0x91ab9144
LBR: @0x0250 --> 0x91a79dee
LBR: @0x0254 --> 0x91a0f9e7
LBR: @0x0258 --> 0x91989bb7
LBR: @0x025c --> 0x91b033b3
LBR: @0x0260 --> 0x91a058b6
LBR: @0x0264 --> 0x917e978b
LBR: @0x0268 --> 0x91a1c3ac
LBR: @0x026c --> 0x91bd1e4c
LBR: @0x0270 --> 0x9194f584
LBR: @0x0274 --> 0x91a9f651
LBR: @0x0278 --> 0x91b1d8ea
LBR: @0x027c --> 0x919d3c94
LBR: @0x0280 --> 0x9199fe82
LBR: @0x0284 --> 0x9185f85d
LBR: @0x0288 --> 0x91a3e936
LBR: @0x028c --> 0x91a266b1
LBR: @0x0290 --> 0x919c1768
LBR: @0x0294 --> 0x918e576c
LBR: @0x0298 --> 0x9199f4d2
LBR: @0x029c --> 0x919c145a
LBR: @0x02a0 --> 0x919d5bdd
LBR: @0x02a4 --> 0x91b5a3d8
LBR: @0x02a8 --> 0x91a9da5d
LBR: @0x02ac --> 0x91910149
LBR: @0x02b0 --> 0x91b22e35
LBR: @0x02b4 --> 0x9193cac9
LBR: @0x02b8 --> 0x9195b172
LBR: @0x02bc --> 0x91bda00e
LBR: @0x02c0 --> 0x91ad3bbb
```

```
LBR: @0x02c4 --> 0x91aee49a
LBR: @0x02c8 --> 0x91ad6ee6
LBR: @0x02cc --> 0x91a48664
LBR: @0x02d0 --> 0x91a57c8a
LBR: @0x02d4 --> 0x91a3b0bc
LBR: @0x02d8 --> 0x919c8ae5
LBR: @0x02dc --> 0x91aca4da
LBR: @0x02e0 --> 0x91aac496
LBR: @0x02e4 --> 0x916dc5e4
LBR: @0x02e8 --> 0x919ef3da
LBR: @0x02ec --> 0x91bb513a
LBR: @0x02f0 --> 0x918f125c
LBR: @0x02f4 --> 0x91a1f7e2
LBR: @0x02f8 --> 0x91a9b880
LBR: @0x02fc --> 0x91a17521
LBR: @0x0300 --> 0x00000348
LBR: @0x0304 --> 0x00fffdcf
LBR: @0x0308 --> 0x000000d0
LBR: @0x030c --> 0x000001f0
LBR: @0x0310 --> 0x0000013c
LBR: @0x0314 --> 0x00fffed5
LBR: @0x0318 --> 0x00ffffcf
LBR: @0x031c --> 0x000001c2
LBR: @0x0320 --> 0x0000021d
LBR: @0x0324 --> 0x00ffff17
LBR: @0x0328 --> 0x0000009b
LBR: @0x032c --> 0x00ffffdf
LBR: @0x0330 --> 0x00ffffd4
LBR: @0x0334 --> 0x00fffed5
LBR: @0x0338 --> 0x00000364
LBR: @0x033c --> 0x000001b5
LBR: @0x0340 --> 0x00ffff8d
LBR: @0x0344 --> 0x00fffbed
LBR: @0x0348 --> 0x000000d2
LBR: @0x034c --> 0x00000179
LBR: @0x0350 --> 0x00ffff8a
LBR: @0x0354 --> 0x0000024a
LBR: @0x0358 --> 0x00000332
LBR: @0x035c --> 0x00ffff21
LBR: @0x0360 --> 0x000004d4
LBR: @0x0364 --> 0x00fffc16
LBR: @0x0368 --> 0x000001d7
LBR: @0x036c --> 0x000000cb
LBR: @0x0370 --> 0x00ffff5b
LBR: @0x0374 --> 0x00000167
LBR: @0x0378 --> 0x00000060
LBR: @0x037c --> 0x0000029a
LBR: @0x0380 --> 0x00000001
LBR: @0x0384 --> 0x00000001
LBR: @0x0388 --> 0x00000001
LBR: @0x038c --> 0x00000001
LBR: @0x0390 --> 0x00000001
LBR: @0x0394 --> 0x00000001
LBR: @0x0398 --> 0x00000001
LBR: @0x039c --> 0x00000001
LBR: @0x03a0 --> 0x00000001
LBR: @0x03a4 --> 0x00000001
```

```
LBR: @0x03a8 --> 0x00000001
LBR: @0x03ac --> 0x00000001
LBR: @0x03b0 --> 0x00000001
LBR: @0x03b4 --> 0x00000001
LBR: @0x03b8 --> 0x00000001
LBR: @0x03bc --> 0x00000001
LBR: @0x03c0 --> 0x00000001
LBR: @0x03c4 --> 0x00000001
LBR: @0x03c8 --> 0x00000001
LBR: @0x03cc --> 0x00000001
LBR: @0x03d0 --> 0x00000001
LBR: @0x03d4 --> 0x00000001
LBR: @0x03d8 --> 0x00000001
LBR: @0x03dc --> 0x00000001
LBR: @0x03e0 --> 0x00000001
LBR: @0x03e4 --> 0x00000001
LBR: @0x03e8 --> 0x00000001
LBR: @0x03ec --> 0x00000001
LBR: @0x03f0 --> 0x00000001
LBR: @0x03f4 --> 0x00000001
LBR: @0x03f8 --> 0x00000001
LBR: @0x03fc --> 0x00000001
LBR: @0x0400 --> 0x00000001
LBR: @0x0404 --> 0x00000001
LBR: @0x0408 --> 0x00000001
LBR: @0x040c --> 0x00000001
LBR: @0x0410 --> 0x00000001
LBR: @0x0414 --> 0x00000001
LBR: @0x0418 --> 0x00000001
LBR: @0x041c --> 0x00000001
LBR: @0x0420 --> 0x00000001
LBR: @0x0424 --> 0x00000001
LBR: @0x0428 --> 0x00000001
LBR: @0x042c --> 0x00000001
LBR: @0x0430 --> 0x00000001
LBR: @0x0434 --> 0x00000001
LBR: @0x0438 --> 0x00000001
LBR: @0x043c --> 0x00000001
LBR: @0x0440 --> 0x00000001
LBR: @0x0444 --> 0x00000001
LBR: @0x0448 --> 0x00000001
LBR: @0x044c --> 0x00000001
LBR: @0x0450 --> 0x00000001
LBR: @0x0454 --> 0x00000001
LBR: @0x0458 --> 0x00000001
LBR: @0x045c --> 0x00000001
LBR: @0x0460 --> 0x00000001
LBR: @0x0464 --> 0x00000001
LBR: @0x0468 --> 0x00000001
LBR: @0x046c --> 0x00000001
LBR: @0x0470 --> 0x00000001
LBR: @0x0474 --> 0x00000001
LBR: @0x0478 --> 0x00000001
LBR: @0x047c --> 0x00000001
LBR: @0x0480 --> 0x00000001
LBR: @0x0484 --> 0x00000001
LBR: @0x0488 --> 0x00000001
```

```
LBR: @0x048c --> 0x00000001
LBR: @0x0490 --> 0x00000001
LBR: @0x0494 --> 0x00000001
LBR: @0x0498 --> 0x00000001
LBR: @0x049c --> 0x00000001
LBR: @0x04a0 --> 0x00000001
LBR: @0x04a4 --> 0x00000001
LBR: @0x04a8 --> 0x00000001
LBR: @0x04ac --> 0x00000001
LBR: @0x04b0 --> 0x00000001
LBR: @0x04b4 --> 0x00000001
LBR: @0x04b8 --> 0x00000001
LBR: @0x04bc --> 0x00000001
LBR: @0x04c0 --> 0x00000001
LBR: @0x04c4 --> 0x00000001
LBR: @0x04c8 --> 0x00000001
LBR: @0x04cc --> 0x00000001
LBR: @0x04d0 --> 0x00000001
LBR: @0x04d4 --> 0x00000001
LBR: @0x04d8 --> 0x00000001
LBR: @0x04dc --> 0x00000001
LBR: @0x04e0 --> 0x00000001
LBR: @0x04e4 --> 0x00000001
LBR: @0x04e8 --> 0x00000001
LBR: @0x04ec --> 0x00000001
LBR: @0x04f0 --> 0x00000001
LBR: @0x04f4 --> 0x00000001
LBR: @0x04f8 --> 0x00000001
LBR: @0x04fc --> 0x00000001
LBR: @0x0500 --> 0x001f0000
LBR: @0x0504 --> 0x001fffff
LBR: @0x0508 --> 0x00000001
LBR: @0x050c --> 0x00000001
LBR: @0x0510 --> 0x00000001
LBR: @0x0514 --> 0x00000001
LBR: @0x0518 --> 0x00000001
LBR: @0x051c --> 0x00000001
LBR: @0x0520 --> 0x00000001
LBR: @0x0524 --> 0x00000001
LBR: @0x0528 --> 0x00000001
LBR: @0x052c --> 0x00000001
LBR: @0x0530 --> 0x00000001
LBR: @0x0534 --> 0x00000001
LBR: @0x0538 --> 0x00000001
LBR: @0x053c --> 0x00000001
LBR: @0x0540 --> 0x00000001
LBR: @0x0544 --> 0x00000001
LBR: @0x0548 --> 0x00000001
LBR: @0x054c --> 0x00000001
LBR: @0x0550 --> 0x00000001
LBR: @0x0554 --> 0x00000001
LBR: @0x0558 --> 0x00000001
LBR: @0x055c --> 0x00000001
LBR: @0x0560 --> 0x00000001
LBR: @0x0564 --> 0x00000001
LBR: @0x0568 --> 0x00000001
LBR: @0x056c --> 0x00000001
```

```
LBR: @0x0570 --> 0x00000001
LBR: @0x0574 --> 0x00000001
LBR: @0x0578 --> 0x00000001
LBR: @0x057c --> 0x00000001
LBR: @0x0580 --> 0x00000001
LBR: @0x0584 --> 0x00000001
LBR: @0x0588 --> 0x00000001
LBR: @0x058c --> 0x00000001
LBR: @0x0590 --> 0x00000001
LBR: @0x0594 --> 0x00000001
LBR: @0x0598 --> 0x00000001
LBR: @0x059c --> 0x00000001
LBR: @0x05a0 --> 0x00000001
LBR: @0x05a4 --> 0x00000001
LBR: @0x05a8 --> 0x00000001
LBR: @0x05ac --> 0x00000001
LBR: @0x05b0 --> 0x00000001
LBR: @0x05b4 --> 0x00000001
LBR: @0x05b8 --> 0x00000001
LBR: @0x05bc --> 0x00000001
LBR: @0x05c0 --> 0x00000001
LBR: @0x05c4 --> 0x00000001
LBR: @0x05c8 --> 0x00000001
LBR: @0x05cc --> 0x00000001
LBR: @0x05d0 --> 0x00000001
LBR: @0x05d4 --> 0x00000001
LBR: @0x05d8 --> 0x00000001
LBR: @0x05dc --> 0x00000001
LBR: @0x05e0 --> 0x00000001
LBR: @0x05e4 --> 0x00000001
LBR: @0x05e8 --> 0x00000001
LBR: @0x05ec --> 0x00000001
LBR: @0x05f0 --> 0x00000001
LBR: @0x05f4 --> 0x00000001
LBR: @0x05f8 --> 0x00000001
LBR: @0x05fc --> 0x00000001
LBR: @0x0600 --> 0x00000001
LBR: @0x0604 --> 0x00000001
LBR: @0x0608 --> 0x00000001
LBR: @0x060c --> 0x00000001
LBR: @0x0610 --> 0x00000001
LBR: @0x0614 --> 0x00000001
LBR: @0x0618 --> 0x00000001
LBR: @0x061c --> 0x00000001
LBR: @0x0620 --> 0x00000001
LBR: @0x0624 --> 0x00000001
LBR: @0x0628 --> 0x00000001
LBR: @0x062c --> 0x00000001
LBR: @0x0630 --> 0x00000001
LBR: @0x0634 --> 0x00000001
LBR: @0x0638 --> 0x00000001
LBR: @0x063c --> 0x00000001
LBR: @0x0640 --> 0x00000001
LBR: @0x0644 --> 0x00000001
LBR: @0x0648 --> 0x00000001
LBR: @0x064c --> 0x00000001
LBR: @0x0650 --> 0x00000001
```

```
LBR: @0x0654 --> 0x00000001
LBR: @0x0658 --> 0x00000001
LBR: @0x065c --> 0x00000001
LBR: @0x0660 --> 0x00000001
LBR: @0x0664 --> 0x00000001
LBR: @0x0668 --> 0x00000001
LBR: @0x066c --> 0x00000001
LBR: @0x0670 --> 0x00000001
LBR: @0x0674 --> 0x00000001
LBR: @0x0678 --> 0x00000001
LBR: @0x067c --> 0x00000001
LBR: @0x0680 --> 0x00000001
LBR: @0x0684 --> 0x00000001
LBR: @0x0688 --> 0x00000001
LBR: @0x068c --> 0x00000001
LBR: @0x0690 --> 0x00000001
LBR: @0x0694 --> 0x00000001
LBR: @0x0698 --> 0x00000001
LBR: @0x069c --> 0x00000001
LBR: @0x06a0 --> 0x00000001
LBR: @0x06a4 --> 0x00000001
LBR: @0x06a8 --> 0x00000001
LBR: @0x06ac --> 0x00000001
LBR: @0x06b0 --> 0x00000001
LBR: @0x06b4 --> 0x00000001
LBR: @0x06b8 --> 0x00000001
LBR: @0x06bc --> 0x00000001
LBR: @0x06c0 --> 0x00000001
LBR: @0x06c4 --> 0x00000001
LBR: @0x06c8 --> 0x00000001
LBR: @0x06cc --> 0x00000001
LBR: @0x06d0 --> 0x00000001
LBR: @0x06d4 --> 0x00000001
LBR: @0x06d8 --> 0x00000001
LBR: @0x06dc --> 0x00000001
LBR: @0x06e0 --> 0x00000001
LBR: @0x06e4 --> 0x00000001
LBR: @0x06e8 --> 0x00000001
LBR: @0x06ec --> 0x00000001
LBR: @0x06f0 --> 0x00000001
LBR: @0x06f4 --> 0x00000001
LBR: @0x06f8 --> 0x00000001
LBR: @0x06fc --> 0x00000001
LBR: @0x0700 --> 0x00000000
LBR: @0x0704 --> 0x00000000
LBR: @0x0708 --> 0x00000000
LBR: @0x070c --> 0x00000000
LBR: @0x0710 --> 0x00000000
LBR: @0x0714 --> 0x00000000
LBR: @0x0718 --> 0x00000000
LBR: @0x071c --> 0x00000000
LBR: @0x0720 --> 0x00000000
LBR: @0x0724 --> 0x00000000
LBR: @0x0728 --> 0x00000000
LBR: @0x072c --> 0x00000000
LBR: @0x0730 --> 0x00000000
LBR: @0x0734 --> 0x00000000
```

```
LBR: @0x0738 --> 0x00000000
LBR: @0x073c --> 0x00000000
LBR: @0x0740 --> 0x00000000
LBR: @0x0744 --> 0x00000000
LBR: @0x0748 --> 0x00000000
LBR: @0x074c --> 0x00000000
LBR: @0x0750 --> 0x00000000
LBR: @0x0754 --> 0x00000000
LBR: @0x0758 --> 0x00000000
LBR: @0x075c --> 0x00000000
LBR: @0x0760 --> 0x00000000
LBR: @0x0764 --> 0x00000000
LBR: @0x0768 --> 0x00000000
LBR: @0x076c --> 0x00000000
LBR: @0x0770 --> 0x00000000
LBR: @0x0774 --> 0x00000000
LBR: @0x0778 --> 0x00000000
LBR: @0x077c --> 0x00000000
LBR: @0x0780 --> 0x00000000
LBR: @0x0784 --> 0x00000000
LBR: @0x0788 --> 0x00000000
LBR: @0x078c --> 0x00000000
LBR: @0x0790 --> 0x00000000
LBR: @0x0794 --> 0x00000000
LBR: @0x0798 --> 0x00000000
LBR: @0x079c --> 0x00000000
LBR: @0x07a0 --> 0x00000000
LBR: @0x07a4 --> 0x00000000
LBR: @0x07a8 --> 0x00000000
LBR: @0x07ac --> 0x00000000
LBR: @0x07b0 --> 0x00000000
LBR: @0x07b4 --> 0x00000000
LBR: @0x07b8 --> 0x00000000
LBR: @0x07bc --> 0x00000000
LBR: @0x07c0 --> 0x00000000
LBR: @0x07c4 --> 0x00000000
LBR: @0x07c8 --> 0x00000000
LBR: @0x07cc --> 0x00000000
LBR: @0x07d0 --> 0x00000000
LBR: @0x07d4 --> 0x00000000
LBR: @0x07d8 --> 0x00000000
LBR: @0x07dc --> 0x00000000
LBR: @0x07e0 --> 0x00000000
LBR: @0x07e4 --> 0x00000000
LBR: @0x07e8 --> 0x00000000
LBR: @0x07ec --> 0x00000000
LBR: @0x07f0 --> 0x00000000
LBR: @0x07f4 --> 0x00000000
LBR: @0x07f8 --> 0x00000000
LBR: @0x07fc --> 0x00000000

======= LOCAL CONFIG REGISTERS =========
LCR: @0x0000 --> 0xfffff800
LCR: @0x0004 --> 0x00000001
LCR: @0x0008 --> 0x00200000
LCR: @0x000c --> 0x00300400
LCR: @0x0010 --> 0x00000000
```

```
LCR: @0x0014 --> 0x00000000
LCR: @0x0018 --> 0x42430343
LCR: @0x001c --> 0x00000000
LCR: @0x0020 --> 0x00000000
LCR: @0x0024 --> 0x00000000
LCR: @0x0028 --> 0x00000000
LCR: @0x002c --> 0x00000000
LCR: @0x0030 --> 0x00000000
LCR: @0x0034 --> 0x00000008
LCR: @0x0038 --> 0x00000000
LCR: @0x003c --> 0x00000000
LCR: @0x0040 --> 0x00000000
LCR: @0x0044 --> 0x00000000
LCR: @0x0048 --> 0x00000000
LCR: @0x004c --> 0x00000000
LCR: @0x0050 --> 0x00000000
LCR: @0x0054 --> 0x00000000
LCR: @0x0058 --> 0x00000000
LCR: @0x005c --> 0x00000000
LCR: @0x0060 --> 0x00000000
LCR: @0x0064 --> 0x00000000
LCR: @0x0068 --> 0x0f000080
LCR: @0x006c --> 0x100f767e
LCR: @0x0070 --> 0x905610b5
LCR: @0x0074 --> 0x000000ba
LCR: @0x0078 --> 0x00000000
LCR: @0x007c --> 0x00000000
LCR: @0x0080 --> 0x00000003
LCR: @0x0084 --> 0x37880000
LCR: @0x0088 --> 0x00000190
LCR: @0x008c --> 0x00030000
LCR: @0x0090 --> 0x0000000e
LCR: @0x0094 --> 0x00000003
LCR: @0x0098 --> 0x00000000
LCR: @0x009c --> 0x00000000
LCR: @0x00a0 --> 0x00000000
LCR: @0x00a4 --> 0x00000000
LCR: @0x00a8 --> 0x00001010
LCR: @0x00ac --> 0x00200000
LCR: @0x00b0 --> 0x00000000
LCR: @0x00b4 --> 0x00000000
LCR: @0x00b8 --> 0x00000000
LCR: @0x00bc --> 0x00000000
LCR: @0x00c0 --> 0x00000002
LCR: @0x00c4 --> 0x00000000
LCR: @0x00c8 --> 0x00000000
LCR: @0x00cc --> 0x00000000
LCR: @0x00d0 --> 0x00000000
LCR: @0x00d4 --> 0x00000000
LCR: @0x00d8 --> 0x00000000
LCR: @0x00dc --> 0x00000000
LCR: @0x00e0 --> 0x00000000
LCR: @0x00e4 --> 0x00000000
LCR: @0x00e8 --> 0x00000050
LCR: @0x00ec --> 0x00000000
LCR: @0x00f0 --> 0x00000000
LCR: @0x00f4 --> 0x00000000
```

```
LCR: @0x00f8 --> 0x00000043
LCR: @0x00fc --> 0x00000000
LCR: @0x0100 --> 0x00000000
LCR: @0x0104 --> 0x378afff0

======= PCI CONFIG REG ADDR MAPPING =========
PCR: @0x0000 --> 0x92781542
PCR: @0x0004 --> 0x02b00017
PCR: @0x0008 --> 0x08800001
PCR: @0x000c --> 0x00006008
PCR: @0x0010 --> 0xc7301000
PCR: @0x0014 --> 0x00000000
PCR: @0x0018 --> 0xc7300000
PCR: @0x001c --> 0x00000000
PCR: @0x0020 --> 0x00000000
PCR: @0x0024 --> 0x00000000
PCR: @0x0028 --> 0x00000000
PCR: @0x002c --> 0x905610b5
PCR: @0x0030 --> 0x00000000
PCR: @0x0034 --> 0x00000040
PCR: @0x0038 --> 0x00000000
PCR: @0x003c --> 0x0000010b
PCR: @0x0040 --> 0x00024801
PCR: @0x0044 --> 0x00000000
PCR: @0x0048 --> 0x00004c00
PCR: @0x004c --> 0x00000003
PCR: @0x0050 --> 0x00000000

======= PCI BRIDGE REGISTERS =========
PBR: @0x0000 --> 0x811110b5
PBR: @0x0004 --> 0x00100417
PBR: @0x0008 --> 0x06040021
PBR: @0x000c --> 0x00010010
PBR: @0x0010 --> 0xffe0000c
PBR: @0x0014 --> 0x0000383f
PBR: @0x0018 --> 0x20020201
PBR: @0x001c --> 0x220000f0
PBR: @0x0020 --> 0xc730c730
PBR: @0x0024 --> 0x0000fff0
PBR: @0x0028 --> 0x00000000
PBR: @0x002c --> 0x00000000
PBR: @0x0030 --> 0x00000000
PBR: @0x0034 --> 0x00000040
PBR: @0x0038 --> 0x00000000
PBR: @0x003c --> 0x0000010b
PBR: @0x0040 --> 0x5a025001
PBR: @0x0044 --> 0x00000000
PBR: @0x0048 --> 0x000e2012
PBR: @0x004c --> 0x00000000
PBR: @0x0050 --> 0x00816005
PBR: @0x0054 --> 0xfee00000
PBR: @0x0058 --> 0x00000000
PBR: @0x005c --> 0x00004048
PBR: @0x0060 --> 0x00710010
PBR: @0x0064 --> 0x00000000
PBR: @0x0068 --> 0x00002000
PBR: @0x006c --> 0x00024c11
```

```
PBR: @0x0070 --> 0x00110000
PBR: @0x0074 --> 0x00000c80
PBR: @0x0078 --> 0x00400000
PBR: @0x007c --> 0x00000000
PBR: @0x0080 --> 0x00000000
PBR: @0x0084 --> 0x00000000
PBR: @0x0088 --> 0x00000033
PBR: @0x008c --> 0x00000000
PBR: @0x0090 --> 0x00000000
PBR: @0x0094 --> 0x00000000
PBR: @0x0098 --> 0x00000000
PBR: @0x009c --> 0x00000000
PBR: @0x00a0 --> 0x00000000
PBR: @0x00a4 --> 0x00000000
PBR: @0x00a8 --> 0x00000000
PBR: @0x00ac --> 0x00000000
PBR: @0x00b0 --> 0x00000000
PBR: @0x00b4 --> 0x00000000
PBR: @0x00b8 --> 0x00000000
PBR: @0x00bc --> 0x00000000
PBR: @0x00c0 --> 0x00000000
PBR: @0x00c4 --> 0x00000000
PBR: @0x00c8 --> 0x00000000
PBR: @0x00cc --> 0x00000000
PBR: @0x00d0 --> 0x00000000
PBR: @0x00d4 --> 0x00000000
PBR: @0x00d8 --> 0x00000000
PBR: @0x00dc --> 0x00000000
PBR: @0x00e0 --> 0x00000000
PBR: @0x00e4 --> 0x00000000
PBR: @0x00e8 --> 0x00000000
PBR: @0x00ec --> 0x00000000
PBR: @0x00f0 --> 0x00000000
PBR: @0x00f4 --> 0x00000000
PBR: @0x00f8 --> 0x00000000
PBR: @0x00fc --> 0x00000000
PBR: @0x0100 --> 0x00010004
PBR: @0x0104 --> 0x00000000
PBR: @0x0108 --> 0x00000000
PBR: @0x010c --> 0x00000000
PBR: @0x0110 --> 0x00000000
PBR: @0x0114 --> 0x00000000
PBR: @0x0118 --> 0x00000000


======= MAIN CONTROL REGISTERS =========
MCR: @0x0000 --> 0x00000033
MCR: @0x0004 --> 0x8000ff00
MCR: @0x0008 --> 0x00000000
MCR: @0x000c --> 0x03008090
MCR: @0x0010 --> 0x80000002
MCR: @0x0014 --> 0x00000000
MCR: @0x0018 --> 0x00000000
MCR: @0x001c --> 0x00000000
MCR: @0x0020 --> 0x0000141b
MCR: @0x0024 --> 0x00000000
MCR: @0x0028 --> 0x00000000
```

```
MCR: @0x002c --> 0x00000000
MCR: @0x0030 --> 0xfeedface
MCR: @0x0034 --> 0x00000000
MCR: @0x0038 --> 0x00000000
MCR: @0x003c --> 0x00000000
MCR: @0x0040 --> 0x00000201
MCR: @0x0044 --> 0x00000000
MCR: @0x0048 --> 0x00810a20
MCR: @0x004c --> 0x000000d4
MCR: @0x0050 --> 0x00010100
MCR: @0x0054 --> 0x00000000
MCR: @0x0058 --> 0x080a2c2a
MCR: @0x005c --> 0x0000029a
MCR: @0x0060 --> 0x00000019
MCR: @0x0064 --> 0x00000000
```

3.1.3 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:

./ccurdscc_get_sps –b0 *(display all channels)*

```
Device Name            : /dev/ccurdscc0
Board Serial Number    : 4294967295 (0xffffffff)

#### 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)
```

./ccurdscc_get_sps –b0 –c17 *(display channel '17' only)*

```
Device Name            : /dev/ccurdscc0
Board Serial Number    : 4294967295 (0xffffffff)

#### Active Channel Found: 32 #### (Channel Mask: 0xffffffff)
#### Only Channel 17 tracked  ####
delta= 9462.611 usec, samples= 2044 rate=216.0080 Ksps (215.915/216.042/215.987)
```

3.1.4 ccurdscc_rdreg

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

```
Usage: ./ccurdscc_rdreg [-b board] [-o offset] [-s size]
 -b board: Board number -- default board is 0
 -o offset: Hex offset to read from -- default offset is 0x0
 -s size: number of bytes to read -- default size is 0x4
```

Example display:

./ccurdscc_rdreg *(9277- 5volt board)*

```
Device Name    : /dev/ccurdscc0
Board Serial No: 4294967295 (0xffffffff)

Read at offset 0x0000: 0x92770102
```

./ccurdscc_rdreg *(9278- 10volt board)*

```
Device Name    : /dev/ccurdscc0
Board Serial No: 4294967295 (0xffffffff)

Read at offset 0x0000: 0x92780201
```

3.1.5 ccurdscc_reg

This call displays all the boards local and configuration registers.

```
Usage: ./ccurdscc_reg [-b board]
 -b board: Board number -- default board is 0
```

Example display:

./ccurdscc_reg *(9278- 10volt board)*

```
Device Name    : /dev/ccurdscc0
Board Serial No: 4294967295 (0xffffffff)

LOCAL Register 0x7ffff7ffc000 Offset=0x0

#### LOCAL REGS #### (length=2048)
+LCL+        0   92780201  00000300  00000000  00000000 *.x.!............*
+LCL+     0x10   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+     0x20   00000000  000004ff  ffff8681  000000ff *................*
+LCL+     0x30   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+     0x40   00000000  000004ff  ffff8681  000000ff *................*
+LCL+     0x50   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+     0x60   00000000  000004ff  ffff8681  000000ff *................*
+LCL+     0x70   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+     0x80   00000000  000004ff  ffff8681  000000ff *................*
+LCL+     0x90   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+     0xa0   00000001  00000021  00000021  00000021 *.......!...!...!*
+LCL+     0xb0   00000000  00000000  00000021  00000021 *...........!...!*
+LCL+     0xc0   2400fff6  00000000  ffffffff  00000021 *$.............!*
+LCL+     0xd0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+     0xe0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+     0xf0   00000000  00000000  00000021  00000021 *...........!...!*
+LCL+    0x100   007f001c  008000db  007f02c3  007f004b *..............K*
+LCL+    0x110   008000d9  00800257  008001ad  007fff7e *.......W.......~*
+LCL+    0x120   007ffc9c  0080ffc6  007ffe93  007ffddc *................*
+LCL+    0x130   0080ffcd  008000cd  0080008d  007ffdb0 *................*
+LCL+    0x140   0080fcc6  007ffff1  007f033f  008002e0 *...........?....*
+LCL+    0x150   007f00a9  007f0110  008002a4  0080fd59 *..............Y*
+LCL+    0x160   007f0014  007f0172  007ff935  008000ad *.......r...5....*
+LCL+    0x170   007fff37  0080fff4  008003b8  007ffd06 *...7............*
+LCL+    0x180   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x190   0a7ffef4  00000021  00000021  00000021 *.......!...!...!*
+LCL+    0x1a0   00000000  00000533  bd361020  00033380 *.......3.6. ..3.*
+LCL+    0x1b0   00000000  00000120  00002000  00000000 *....... .. .....*
+LCL+    0x1c0   00000000  00000120  00002000  00000000 *....... .. .....*
+LCL+    0x1d0   00000000  00000120  00002000  00000000 *....... .. .....*
+LCL+    0x1e0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x1f0   00000021  00000021  00000021  00000021 *...!...!...!...!*
```

```
+LCL+    0x200   91a9957f  91b09dde  91b5a8e1  91c3b621 *..............!*
+LCL+    0x210   91b99910  91943c54  91b872c6  91986136 *......<T..r...a6*
+LCL+    0x220   9196a3f8  9193209d  9199d397  91a52729 *...... .......')*
+LCL+    0x230   917f6e28  919a5a58  919c8756  91925f50 *..n(..ZX...V.._P*
+LCL+    0x240   91b58ccf  91b33694  91a4bb5e  91af57d4 *......6....^..W.*
+LCL+    0x250   91b9b25f  91aa815c  91bb9091  919db17a *..._...\.......z*
+LCL+    0x260   91977af9  91893523  918b18f5  919b5d04 *..z...5#......].*
+LCL+    0x270   9180c0ab  918a4ed0  918ab0ad  918a33e7 *......N.......3.*
+LCL+    0x280   91aff664  91a9a757  91b38b73  91b80569 *...d...W...s...i*
+LCL+    0x290   91b81174  91922e1e  91afa6ff  91989566 *...t...........f*
+LCL+    0x2a0   9197012d  918a480a  919dabeb  919a9a3a *...-..H........:*
+LCL+    0x2b0   9184cc71  919fc0d9  918a115a  9192e49b *...q.......Z....*
+LCL+    0x2c0   91bbd51e  91ac86b2  91b03f8c  91b13d8d *..........?...=.*
+LCL+    0x2d0   91b987a3  91a5e161  91b996a1  91a7857f *.......a........*
+LCL+    0x2e0   91969b0d  918b6c55  91932646  918fc6a1 *......lU..&F....*
+LCL+    0x2f0   917e31b3  918aa3e8  9183df74  9189e85f *.~1........t..._*
+LCL+    0x300   00000102  000001e9  00000377  00fffe8c *...........w....*
+LCL+    0x310   00000026  00000172  0000011a  00000109 *...&...r........*
+LCL+    0x320   0000035f  00000024  00000329  000002c6 *..._...$...)....*
+LCL+    0x330   00000396  00000042  00ffff99  00000045 *.......B.......E*
+LCL+    0x340   000000a4  00fffe4f  000005de  00000441 *.......O.......A*
+LCL+    0x350   0000012c  00000206  00fffe9a  00000046 *...,...........F*
+LCL+    0x360   000000ca  00fffdb8  000001df  00fffe77 *................w*
+LCL+    0x370   00000292  000000d2  00ffffa9  000001f1 *................*
+LCL+    0x380   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x390   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x3a0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x3b0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x3c0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x3d0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x3e0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x3f0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x400   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x410   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x420   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x430   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x440   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x450   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x460   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x470   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x480   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x490   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x4a0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x4b0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x4c0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x4d0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x4e0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x4f0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x500   00000000  00000000  00000021  00000021 *...........!...!*
+LCL+    0x510   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x520   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x530   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x540   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x550   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x560   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x570   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x580   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x590   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x5a0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x5b0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x5c0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x5d0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x5e0   00000021  00000021  00000021  00000021 *...!...!...!...!*
```

```
+LCL+    0x5f0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x600   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x610   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x620   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x630   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x640   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x650   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x660   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x670   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x680   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x690   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x6a0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x6b0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x6c0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x6d0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x6e0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x6f0   00000021  00000021  00000021  00000021 *...!...!...!...!*
+LCL+    0x700   00000000  00000000  00000000  00000000 *................*
+LCL+    0x710   00000000  00000000  00000000  00000000 *................*
+LCL+    0x720   00000000  00000000  00000000  00000000 *................*
+LCL+    0x730   00000000  00000000  00000000  00000000 *................*
+LCL+    0x740   00000000  00000000  00000000  00000000 *................*
+LCL+    0x750   00000000  00000000  00000000  00000000 *................*
+LCL+    0x760   00000000  00000000  00000000  00000000 *................*
+LCL+    0x770   00000000  00000000  00000000  00000000 *................*
+LCL+    0x780   00000000  00000000  00000000  00000000 *................*
+LCL+    0x790   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7a0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7b0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7c0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7d0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7e0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7f0   00000000  00000000  00000000  00000000 *................*

CONFIG Register 0x7ffff7ffb800 Offset=0x800

#### CONFIG REGS #### (length=264)
+CFG+       0   fffff800  00000001  00200000  00300400 *........ ...0..*
+CFG+    0x10   00000000  00000000  42430343  00000000 *........BC.C....*
+CFG+    0x20   00000000  00000000  00000000  00000000 *................*
+CFG+    0x30   00000000  00000008  00000000  00000000 *................*
+CFG+    0x40   00000000  00000000  00000000  00000000 *................*
+CFG+    0x50   00000000  00000000  00000000  00000000 *................*
+CFG+    0x60   00000000  00000000  0f000080  100f767e *..............v~*
+CFG+    0x70   905610b5  000000ba  00000000  00000000 *.V..............*
+CFG+    0x80   00000043  79b80000  00000100  00000080 *...Cy...........*
+CFG+    0x90   0000000a  00000003  00000000  00000000 *................*
+CFG+    0xa0   00000000  00000000  00001011  00200000 *.............  ..*
+CFG+    0xb0   00000000  00000000  00000000  00000000 *................*
+CFG+    0xc0   00000002  00000000  00000000  00000000 *................*
+CFG+    0xd0   00000000  00000000  00000000  00000000 *................*
+CFG+    0xe0   00000000  00000000  00000050  00000000 *...........P....*
+CFG+    0xf0   00000000  00000000  00000043  00000000 *...........C....*
+CFG+   0x100   00000000  79b80078                     *....y..x        *

======= LOCAL REGISTERS =========
    board_info                =0x92780201      @0x00000000
    board_csr                 =0x00000300      @0x00000004
    interrupt_control         =0x00000000      @0x00000008
    interrupt_status          =0x00000000      @0x0000000c
    converter[C0].CPM_csr     =0x00000000      @0x00000020
    converter[C0].CPM_access  =0x000004ff      @0x00000024
    converter[C0].CPM_read_1  =0xffff8681      @0x00000028
```

```
    converter[C0].CPM_read_2   =0x000000ff      @0x0000002c
    converter[C1].CPM_csr      =0x00000000      @0x00000040
    converter[C1].CPM_access   =0x000004ff      @0x00000044
    converter[C1].CPM_read_1   =0xffff8681      @0x00000048
    converter[C1].CPM_read_2   =0x000000ff      @0x0000004c
    converter[C2].CPM_csr      =0x00000000      @0x00000060
    converter[C2].CPM_access   =0x000004ff      @0x00000064
    converter[C2].CPM_read_1   =0xffff8681      @0x00000068
    converter[C2].CPM_read_2   =0x000000ff      @0x0000006c
    converter[C3].CPM_csr      =0x00000000      @0x00000080
    converter[C3].CPM_access   =0x000004ff      @0x00000084
    converter[C3].CPM_read_1   =0xffff8681      @0x00000088
    converter[C3].CPM_read_2   =0x000000ff      @0x0000008c
    PLL_sync                   =0x00000001      @0x000000a0
    calib_voltage_control      =0x00000000      @0x000000b0
    test_bus_control           =0x00000000      @0x000000b4
    fifo_csr                   =0x2400fff2      @0x000000c0
    fifo_threshold             =0x00000000      @0x000000c4
    fifo_channel_select        =0xffffffff      @0x000000c8
    spi_counter_status         =0x00000000      @0x000000f0

        channel_data[0..31]
@0x0100 007fff60 007fff15 00800036 008000e2 007fff7d 007ffbdf 008003f9 007ffc4b
@0x0120 007ffe54 008001c5 008004c7 007fffd3 007ffae9 007fff9d 008000bd 007ffe6c
@0x0140 007fff94 007fff81 007fff12 007fff43 007ffeff 00800161 007fffd4 00800307
@0x0160 007ffefc 00800151 007ffd9a 007ffffb 00800117 0080021c 007fff81 007ffc19
    fifo_data                  =0x0e7ffcfa      @0x00000190
    pll[P0].PLL_status         =0x00000000      @0x000001a0
    pll[P0].PLL_access         =0x00000533      @0x000001a4
    pll[P0].PLL_read_1         =0xbd361020      @0x000001a8
    pll[P0].PLL_read_2         =0x00033380      @0x000001ac
    pll[P1].PLL_status         =0x00000000      @0x000001b0
    pll[P1].PLL_access         =0x00000120      @0x000001b4
    pll[P1].PLL_read_1         =0x00002000      @0x000001b8
    pll[P1].PLL_read_2         =0x00000000      @0x000001bc
    pll[P2].PLL_status         =0x00000000      @0x000001c0
    pll[P2].PLL_access         =0x00000120      @0x000001c4
    pll[P2].PLL_read_1         =0x00002000      @0x000001c8
    pll[P2].PLL_read_2         =0x00000000      @0x000001cc
    pll[P3].PLL_status         =0x00000000      @0x000001d0
    pll[P3].PLL_access         =0x00000120      @0x000001d4
    pll[P3].PLL_read_1         =0x00002000      @0x000001d8
    pll[P3].PLL_read_2         =0x00000000      @0x000001dc

        positive_calibration[0..31]
@0x0200 91a9957f 91b09dde 91b5a8e1 91c3b621 91b99910 91943c54 91b872c6 91986136
@0x0220 9196a3f8 9193209d 9199d397 91a52729 917f6e28 919a5a58 919c8756 91925f50
@0x0240 91b58ccf 91b33694 91a4bb5e 91af57d4 91b9b25f 91aa815c 91bb9091 919db17a
@0x0260 91977af9 91893523 918b18f5 919b5d04 9180c0ab 918a4ed0 918ab0ad 918a33e7

        negative_calibration[0..31]
@0x0280 91aff664 91a9a757 91b38b73 91b80569 91b81174 91922e1e 91afa6ff 91989566
@0x02a0 9197012d 918a480a 919dabeb 919a9a3a 9184cc71 919fc0d9 918a115a 9192e49b
@0x02c0 91bbd51e 91ac86b2 91b03f8c 91b13d8d 91b987a3 91a5e161 91b996a1 91a7857f
@0x02e0 91969b0d 918b6c55 91932646 918fc6a1 917e31b3 918aa3e8 9183df74 9189e85f

        offset_calibration[0..31]
@0x0300 00000102 000001e9 00000377 00fffe8c 00000026 00000172 0000011a 00000109
@0x0320 0000035f 00000024 00000329 000002c6 00000396 00000042 00ffff99 00000045
@0x0340 000000a4 00fffe4f 000005de 00000441 0000012c 00000206 00fffe9a 00000046
@0x0360 000000ca 00fffdb8 000001df 00fffe77 00000292 000000d2 00ffffa9 000001f1

    sprom_stat_addr_write_data =0x021f0000      @0x00000500
```

```
    sprom_read_data            =0x021f918b      @0x00000504

        spi_ram[0..63]
@0x0700 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x0720 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x0740 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x0760 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x0780 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x07a0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x07c0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x07e0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

======= CONFIG REGISTERS =========
    las0rr                     =0xfffff800      @0x00000000
    las0ba                     =0x00000001      @0x00000004
    marbr                      =0x00200000      @0x00000008
    bigend                     =0x00300400      @0x0000000c
    eromrr                     =0x00000000      @0x00000010
    eromba                     =0x00000000      @0x00000014
    lbrd0                      =0x42430343      @0x00000018
    dmrr                       =0x00000000      @0x0000001c
    dmlbam                     =0x00000000      @0x00000020
    dmlbai                     =0x00000000      @0x00000024
    dmpbam                     =0x00000000      @0x00000028
    dmcfga                     =0x00000000      @0x0000002c
    oplfis                     =0x00000000      @0x00000030
    oplfim                     =0x00000008      @0x00000034
    mbox0                      =0x00000000      @0x00000040
    mbox1                      =0x00000000      @0x00000044
    mbox2                      =0x00000000      @0x00000048
    mbox3                      =0x00000000      @0x0000004c
    mbox4                      =0x00000000      @0x00000050
    mbox5                      =0x00000000      @0x00000054
    mbox6                      =0x00000000      @0x00000058
    mbox7                      =0x00000000      @0x0000005c
    p2ldbell                   =0x00000000      @0x00000060
    l2pdbell                   =0x00000000      @0x00000064
    intcsr                     =0x0f000080      @0x00000068
    cntrl                      =0x100f767e      @0x0000006c
    pcihidr                    =0x905610b5      @0x00000070
    pcihrev                    =0x000000ba      @0x00000074
    dmamode0                   =0x00000043      @0x00000080
    dmapadr0                   =0x79b80000      @0x00000084
    dmaladr0                   =0x00000100      @0x00000088
    dmasiz0                    =0x00000080      @0x0000008c
    dmadpr0                    =0x0000000a      @0x00000090
    dmamode1                   =0x00000003      @0x00000094
    dmapadr1                   =0x00000000      @0x00000098
    dmaladr1                   =0x00000000      @0x0000009c
    dmasiz1                    =0x00000000      @0x000000a0
    dmadpr1                    =0x00000000      @0x000000a4
    dmacsr0                    =0x00000011      @0x000000a8
    dmacsr1                    =0x00000010      @0x000000a9
    dmaaarb                    =0x00200000      @0x000000ac
    dmathr                     =0x00000000      @0x000000b0
    dmadac0                    =0x00000000      @0x000000b4
    dmadac1                    =0x00000000      @0x000000b8
    las1rr                     =0x00000000      @0x000000f0
    las1ba                     =0x00000000      @0x000000f4
    lbrd1                      =0x00000043      @0x000000f8
    dmdac                      =0x00000000      @0x000000fc
    pciarb                     =0x00000000      @0x00000100
    pabtadr                    =0x79b80078      @0x00000104
```

3.1.6 ccurdscc_regedit

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

Usage: ./ccurdscc_regedit [-b board]
 -b board: Board number -- default board is 0

Example display:

./ccurdscc_regedit

```
Device Name     : /dev/ccurdscc0
Board Serial No : 4294967295 (0xffffffff)
Initialize_Board: Firmware Rev. 0x1 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)->
```

3.1.7 ccurdscc_tst

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

Usage: ./ccurdscc_tst [-b board]
 -b board: Board number -- default board is 0

Example display:

./ccurdscc_tst

```
Device Name     : /dev/ccurdscc0
Board Serial No : 4294967295 (0xffffffff)
Initialize_Board: Firmware Rev. 0x1 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)->
```

3.1.8 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] [-s size] [-v value]
 -b board : Board selection --  default board is 0
 -o offset: Hex offset to write to -- default offset is 0x0
 -s size: number of bytes to write -- default size is 0x4
 -v value: Hex value to write at offset -- default value is 0x0
```

Example display:

./ccurdscc_wreg *(9277- 5volt board)*

```
Device Name    : /dev/ccurdscc0
Board Serial No: 4294967295 (0xffffffff)

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

./ccurdscc_wreg *(9278- 10volt board)*

```
Device Name    : /dev/ccurdscc0
Board Serial No: 4294967295 (0xffffffff)

Writing 0x00000000 to offset 0x0000
Read at offset 0x0000: 0x92780201
```

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 lib/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 automatically calibrated using internal voltage reference. 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.

Another point to note is that prior to performing a calibration, the board needs to be running (programmed) for some time so that the temperatures reflect the run state. At this point, the calibration can be performed in order to provide more accurate results.

```
Usage: ./ccurdscc_calibrate [-A] [-b board] [-c PLL] [-C mask] [-E extcal]
                            [-f format] [-F] [-i inCalFile] [-o outCalFile] [-p]
                            [-s sample_rate] [-T TestBus] [-v Cal] [-X ExtClock]
 -A                     (Perform Auto Calibration and exit)
 -b <board>             (Board #, default = 0)
 -c <P#C#>              (Assign PLL clock to Converter: P#=0..3,e, C#=0..3,a)
 -C <chan sel mask>     (Channel selection mask)
 -E <w>,<st>,<en>,<v>   (Perform External Calibration
                        (w='+|-|o', st=start ch, en=end ch, v=ext volt)
 -E+                    (Perform External Positive Cal. st=0,en=31,v=+Ref Volt)
 -E-                    (Perform External Negative Cal. st=0,en=31,v=-Ref Volt)
 -Eo                    (Perform External Offset Cal. st=0,en=31,v=zero)
 -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)
 -T <TestBus>           (Default  = No Change
                           'b'    - Calibration Bus Control
                           'o'    - Open
                           'r'    - 5 Volt Reference
 -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)
                           '0..3' - PLL clock
                           'e'    - External clock
```

Example display:

./ccurdscc_calibrate

```
Device Name    : /dev/ccurdscc0
Board Serial No: 4294967295 (0xffffffff)

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)
```

```
===> Dump to 'stdout'
#Date           : Tue May  6 16:58:20 2014
#Board Serial No: 4294967295 (0xffffffff)

#PLL#: PLL_read_1 PLL_read_2
 PLL0: 0x00000000 0x00000000
 PLL1: 0x00000000 0x00000000
 PLL2: 0x00000000 0x00000000
 PLL3: 0x00000000 0x00000000

 PLL_sync: 0x00000000
 PLL_fRef: 65.536000
 ExtClk_Freq: 0.000000

#CPM#: CPM_csr    CPM_read_1 CPM_read_2
 CPM0: 0x00000008 0x00000000 0x00000000
 CPM1: 0x00000008 0x00000000 0x00000000
 CPM2: 0x00000008 0x00000000 0x00000000
 CPM3: 0x00000008 0x00000000 0x00000000

#Chan   Negative    Offset      Positive
#====   ========    ======      ========
 ch00:  1.13817756  0.00062346  1.13801234
 ch01:  1.13798389  0.00150204  1.13819617
 ch02:  1.13831496  0.00120997  1.13837921
 ch03:  1.13845147 -0.00010490  1.13879659
 ch04:  1.13844664 -0.00023127  1.13849963
 ch05:  1.13726768  0.00049591  1.13734001
 ch06:  1.13817720  0.00073552  1.13847080
 ch07:  1.13744774  0.00022411  1.13748805
 ch08:  1.13740669  0.00098467  1.13741457
 ch09:  1.13705322  0.00057578  1.13730226
 ch10:  1.13762401  0.00078082  1.13753394
 ch11:  1.13752466  0.00105739  1.13785791
 ch12:  1.13686886  0.00154376  1.13671683
 ch13:  1.13768282 -0.00008225  1.13752552
 ch14:  1.13702987  0.00007272  1.13759905
 ch15:  1.13728815  0.00023723  1.13729101
 ch16:  1.13857680  0.00011683  1.13840102
 ch17:  1.13813042 -0.00044107  1.13833145
 ch18:  1.13823962  0.00194907  1.13788610
 ch19:  1.13829327  0.00137329  1.13824026
 ch20:  1.13850332  0.00036001  1.13853591
 ch21:  1.13792067  0.00034928  1.13806604
 ch22:  1.13850390 -0.00054002  1.13857194
 ch23:  1.13796982  0.00003457  1.13769358
 ch24:  1.13748415  0.00031710  1.13751006
 ch25:  1.13714905 -0.00077724  1.13708145
 ch26:  1.13738782  0.00061631  1.13714448
 ch27:  1.13729314 -0.00051141  1.13764725
 ch28:  1.13671837  0.00062704  1.13680499
 ch29:  1.13711190  0.00035882  1.13709227
 ch30:  1.13692011 -0.00025511  1.13711870
 ch31:  1.13708468  0.00056863  1.13709323
```

3.2.2 lib/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:

./ccurdscc_compute_pll_clock

```
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 lib/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] [-a#] [-b board] [-c PLL] [-C mask] [-d delay] [-D
debugfile] [-E ExpInpVolt] [-f format] [-F] [-l loopcnt] [-m mode] [-N] [-o
outfile] [-p] [-s sample_rate] [-v Cal] [-X ExtClock]
 -A                      (Perform Auto Calibration)
 -a <#>                  (Display rolling average of # values.)
 -b <board>              (Default = 0)
 -c <P#C#>               (Assign PLL clock to Converter: P#=0..3,e, C#=0..3,a)
 -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)
    <+/-99>@<Tol>        (Calibration Ref Volt@Tolerance)
 -f <format 'b', '2'>    (Default = 'b' Offset Binary)
 -F                      (Enable High-Pass Filter)
 -l <#>                  (Specify loop count)
 -mC                     (Driver DMA read mode [CONTINUOUS FIFO])
 -md                     (User DMA read mode [FIFO])
 -mD                     (Driver DMA read mode [FIFO])
 -me                     (User Demand DMA read mode [DEMAND 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])
 -mW                      (Driver Demand DMA read mode [DEMAND FIFO])
 -mx                      (User DMA read mode [CHANNEL])
 -mX                      (Driver DMA read mode [CHANNEL])
 -mZ                      (Driver Demand DMA read mode [DEMAND CONTINUOUS FIFO])
 -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)
                             '0..3' - PLL clock
                             'e'    - External clock
```

Example display:

./ccurdscc_disp *(9277 – 5Volt board)*

```
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            : **** Not Detected ****
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
```

./ccurdscc_disp *(9278 –10Volt board)*

```
Rolling Average Count [-a]: 10000
Board Number          [-b]: 0  ==> '/dev/ccurdscc0'
Board Serial Number       : 4294967295 (0xffffffff)
Clock Used            [-c]: P0->C0, P1->C1, P2->C2, P3->C3
Channel Sel Mask      [-C]: 0xffffffff
Delay                 [-d]: 0 milli-seconds
Expected Input Volts  [-E]: === Not Specified ===
Data Format           [-f]: Offset Binary
High Pass Filter      [-F]: 'Last set state'
Loop Count            [-l]: ***Forever***
Read Mode             [-m]: Driver DMA (Channel Data)
Program Board         [-p]: No
Calibration Sel       [-v]: Input Signal
External Clock            : **** Not Detected ****
External Clock Output [-X]: PLL 0
Test Bus Control          : Open
Input Voltage Range       : +/-10.0 Volts
Calibration Ref Voltage   : 9.910 Volts
Read Error?               : ===== no ====

Scan count:      260541, Total Delta:  12.1 usec (min= 10.5,max=207.5,av= 11.5)

         ##### Raw Data (Rolling Average Count [10000/10000]) #####
         [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]
         ======   ======   ======   ======   ======   ======   ======   ======
Conv[0]  7ffff0   7fffb6   7fffeb   7fffd1   7ffff9   80001d   80002d   7ffff9

Conv[1]  7ffffe   7fffc3   80000a   7fffe5   80001b   80000b   800014   80000f

Conv[2]  800019   7fffe7   7ffffd   800002   800006   7ffff6   7ffff9   800003

Conv[3]  7ffffc   7ffff7   7ffff9   7ffff3   800046   7ffff4   800025   800009


         ##### Volts (Rolling Average Count [10000/10000]) #####
         [0]       [1]       [2]       [3]       [4]       [5]       [6]       [7]
         ========  ========  ========  ========  ========  ========  ========  ========
Conv[0] -0.00002  -0.00009  -0.00003  -0.00006  -0.00001  +0.00003  +0.00005  -0.00001

Conv[1] -0.00000  -0.00007  +0.00001  -0.00003  +0.00003  +0.00001  +0.00002  +0.00002

Conv[2] +0.00003  -0.00003  -0.00000  +0.00000  +0.00001  -0.00001  -0.00001  +0.00000

Conv[3] -0.00000  -0.00001  -0.00001  -0.00002  +0.00008  -0.00001  +0.00004  +0.00001
```

3.2.4 lib/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 PPL] [-C mask] [-d
debugfile] [-E ExpInpVolt] [-f format] [-F] [-l count] [-m mode] [-N] [-o] [-p]
[-r size] [-s sample_rate] [-S] [-v Cal] [-W] [-X ExtClock] [-Z]
 -A                      (Perform Auto Calibration and exit)
 -b <board>              (Board #, default = 0)
 -B <DMA Cont Bufs>      (DMA Continuous Buffers)
 -c <P#C#>               (Assign PLL clock to Converter: P#=0..3,e, C#=0..3,a)
 -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)
    <+/-99>@<Tol>        (Calibration Ref Volt@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])
 -me                     (User Demand DMA read mode [DEMAND FIFO])
 -mf                     (User PIO read mode [FIFO])
 -mF                     (Driver PIO read mode [FIFO])
 -mW                     (Driver Demand DMA read mode [DEMAND FIFO])
 -mZ                     (Driver Demand DMA read mode [DEMAND CONTINUOUS FIFO])
 -N                      (Open device with O_NONBLOCK flag)
 -o                      (Abort on overflow)
 -p                      (Program board to max clock first)
 -r <read size>          (Sample to read: 1 - 65535)
 -s <sample rate>        (Sample rate: 2000 - 216000 sps)
 -S                      (Display Auto Calibration Status)
 -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)
 -W                      (A/C input signal - Sine Wave)
 -X[0..3,e]              (Board External Clock Output Selection)
                            '0..3' - PLL clock
                            'e'    - External clock
 -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
Board Serial No: 4294967295 (0xffffffff)

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=42464 t=5.06ms (4.73/11.16/4.99) 38.86MB/s
tol=0 overflow=0

Total Tolerance Exceed Count=0




./ccurdscc_fifo  -vi -l50 -W -E5.0@0.023 -Cff000000

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

```
Board Serial No: 327685 (0x00050005)

Clock Used                [-c]: P0->C0, P0->C1, P0->C2, P0->C3
External Clock Output     [-X]: PLL 0
Channels Selected Mask    [-C]: 0xff000000 (Number of Channels: 8)
Expected Input Volts      [-E]: 5.000000 volts (Tolerance 0.023000 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=8, sample_rate=216000.00
SPS...done
000050: Samples Read=49152 Remaining=10520 t=26.16ms (26.00/32.52/26.30) 7.51MB/s
tol=2 overflow=0

Total Tolerance Exceed Count=2 (AC Input Signal - Sine Wave 10.00 Volts P-P)
```

Chan	MinVolts	MaxVolts	MinPpAmp	MaxPpAmp	MinError	MaxError	dB	TolerExcedCnt	
24	-4.998290	+4.984020	+9.978015	+9.981706	-0.018294	-0.021985	-0.01912	-	
25	-4.997522	+4.984452	+9.976949	+9.981325	-0.018675	-0.023051	-0.02004	1	<====
26	-4.998883	+4.984026	+9.978133	+9.982238	-0.017762	-0.021867	-0.01901	-	
27	-4.997643	+4.983948	+9.977010	+9.981166	-0.018834	-0.022990	-0.01999	-	
28	-4.997557	+4.984485	+9.976912	+9.981024	-0.018976	-0.023088	-0.02008	1	<====
29	-4.998701	+4.983884	+9.977325	+9.981657	-0.018343	-0.022675	-0.01972	-	
30	-4.998961	+4.983773	+9.978577	+9.982203	-0.017797	-0.021423	-0.01863	-	
31	-4.998147	+4.983711	+9.977086	+9.981267	-0.018733	-0.022914	-0.01993	-	

3.2.5 lib/ccurdscc_identify

This test is useful in identifying a particular board from a number of installed boards, by flashing the LED for a period of time.

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

Example display:

./ccurdscc_identify

```
Device Name    : /dev/ccurdscc0
Board ID       : 9278
Board Serial No: 12345678 (0x00bc614e)

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

3.2.6 lib/ccurdscc_tst_lib

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

```
Usage: ./ccurdscc_tst_lib [-b board]
 -b board: board number -- default board is 0
```

Example display:

./ccurdscc_tst_lib

```
Device Name    : /dev/ccurdscc0
Board Serial No: 4294967295 (0xffffffff)

Configured Channels Information...
       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 = Allocate DMA Cont. Bufs
  03 = Clear Driver Error               04 = Clear Library Error
  05 = Display BOARD Registers          06 = Display CONFIG Registers
  07 = Get Board CSR                    08 = Get Board Information
  09 = Get Driver Error                 10 = Get Driver Information
  11 = Get Driver Read Mode             12 = Get Fifo Channel Select
  13 = Get Fifo Information             14 = Get Library Error
  15 = Get Mapped Config Pointer        16 = Get Mapped Local Pointer
  17 = Get Number of DMA Cont. Buffers 18 = Get Physical Memory
  19 = Get Test Bus Control             20 = Get Value
  21 = Initialize Board                 22 = MMap Physical Memory
  23 = Munmap Physical Memory           24 = One Shot Test
  25 = Read Operation                   26 = Read Channels
  27 = Remove DMA Cont. Buffers         28 = Reset Board
  29 = Reset DMA Continuous Buffers     30 = Reset Fifo
  31 = Select Driver Read Mode          32 = Set Board CSR
  33 = Set Fifo Channel Select          34 = Set Fifo Threshold
  35 = Set Test Bus Control             36 = Set Value
  37 = Write Operation                  38 = ### CALIBRATION MENU ###
  39 = ### CONVERTER MENU ###           40 = ### INTERRUPT MENU ###
  41 = ### PLL MENU ###                 42 = ### SERIAL PROM MENU ###

Main Selection ('h'=display menu, 'q'=quit)->
_____________________________________________________________________________
Main Selection ('h'=display menu, 'q'=quit)-> 38
   Command: calibration_menu()
  01 = Auto Calibration Status          02 = Get Converter Calibration
  03 = Perform Auto Calibration         04 = Perform Auto Calibration Async
  05 = Perform Neg Calib (External)     06 = Perform Offset Calib (External)
  07 = Perform Pos Calib (External)     08 = Perform Negative Calibration
  09 = Perform Offset Calibration       10 = Perform Positive Calibration
  11 = Read Channels Calibration        12 = Reset Calibration
  13 = Write Channels Calibration

Calibration Selection ('h'=display menu, 'q'=quit)->
_____________________________________________________________________________
Main Selection ('h'=display menu, 'q'=quit)-> 39
   Command: converter_menu()
  01 = Configure Channels               02 = Configure Channels Info
  03 = Get Converter Cal CSR            04 = Get Converter CSR
  05 = Get Converter Information        06 = Program CPM (Advanced)
  07 = Reset Converter                  08 = Set Converter Cal CSR
  09 = Set Converter Clock Source       10 = Set Converter Negative Cal
  11 = Set Converter Offset Cal         12 = Set Converter Positive Cal

Converter Selection ('h'=display menu, 'q'=quit)->
_____________________________________________________________________________
Main Selection ('h'=display menu, 'q'=quit)-> 40
   Command: interrupt_menu()
  01 = Add Irq                          02 = Disable Pci Interrupts
  03 = Enable Pci Interrupts            04 = Get Interrupt Control
```

```
  05 = Get Interrupt Status            06 = Get Interrupt Timeout
  07 = Remove Irq                      08 = Set Interrupt Control
  09 = Set Interrupt Status            10 = Set Interrupt Timeout

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

```
Main Selection ('h'=display menu, 'q'=quit)-> 41
   Command: pll_menu()
  01 = Get PLL Information             02 = Get PLL Status
  03 = Get PLL Synchronization         04 = Program PLL (Advanced)
  05 = Program PLL Clock               06 = Set PLL Synchronization
  07 = Shutdown PLL Clock              08 = Start PLL Clock
  09 = Stop PLL Clock

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

```
Main Selection ('h'=display menu, 'q'=quit)-> 42
   Command: serial_prom_menu()
  01 = Clear Serial Prom               02 = Create Factory Calibration
  03 = Create User Checkpoint          04 = Read Serial PROM
  05 = Read Serial PROM Item           06 = Restore Factory Calibration
  07 = Restore User Checkpoint         08 = Serial PROM Write Override
  09 = View Factory Calibration        10 = View User Checkpoint
  11 = Write Serial PROM               12 = Write Serial PROM Item

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

3.2.7 lib/sprom/ccurdscc_sprom

This utility is available to the user to control the viewing and editing of the non-volatile serial prom information on the board. Once again, this utility should only be used by users that are aware that incorrect usage could result in useful information being permanently lost.

```
Usage: ./ccurdscc_sprom [-b board] [-C] [-D] [-F] [-i inCalFile] [-o outCalFile]
                        [-R] [-S serialNo] [-U num]
 -b <board>            (Board #, default = 0)
 -C                    (Clear ENTIRE serial PROM first)
 -D                    (Dump entire serial prom)
 -F                    (Select factory calibration)
 -i <inCalFile>        (Input calibration file [input->factory])
                       (                      [input->user_checkpoint])
 -i.                   (Create user checkpoint using board reg as input)
 -o <outCalFile>       (Output calibration file [factory->output])
                       (                        [user_checkpoint->output])
 -R                    (Perform Factory or User Checkpoint restore)
 -S <serialNo>         (Program board serial number)
 -U <num>              (Select user checkpoint. <num> is 1 or 2)

Cannot use '-F' and '-U#' in same command line
   e.g.  ./ccurdscc_sprom -F -o CalOut -> Dump Factory calibration to CalOut
         ./ccurdscc_sprom -F -i CalIn  -> Program Factory calibration sprom using
                                          CalIn file
         ./ccurdscc_sprom -U1 -i CalIn -> Create user checkpoint 1 using CalIn
                                          file
         ./ccurdscc_sprom -U 2 -i.     -> Create user checkpoint 2 using memory
                                          register
         ./ccurdscc_sprom -U2 -o CalOut-> Dump user checkpoint 2 to CalOut
         ./ccurdscc_sprom -F -R        -> Restore memory registers using factory
                                          settings
         ./ccurdscc_sprom -U 1 -R      -> Restore memory registers using user
                                          checkpoint 1
```

This page intentionally left blank