Software Interface
CCURAOCC (WC-DA3218)

PCIe 32-Channel Digital to Analog Output Converter Card (AOCC)

Driver	ccuraocc (WC-DA3218)	
Platform	RedHawk Linux® *(CentOS/Rocky/RHEL & Ubuntu)*, Native Ubuntu® and Native Red Hat Enterprise Linux®[1]	
Vendor	Concurrent Real-Time	
Hardware	PCIe 8-Channel (CP-DA0818) or 32-Channel (CP-DA3218) Digital to Analog Output Converter Card	
Author	Darius Dubash	
Date	July 11th, 2025	Rev 2025.1

---

[1] All trademarks are the property of their respective owners

This page intentionally left blank

Table of Contents

This page intentionally left blank

1. Introduction

This document provides the software interface to the ***ccuraocc*** driver which communicates with the Concurrent Real-Time PCI Express 8-Channel or 32-Channel Digital to Analog Output Converter Card (AOCC). Low-level programming information is contained in *Concurrent Real-Time PCIe 8-Channel or 32-Channel Digital to Analog Output Converter Cards (AOCC) Design Specification (No. 0610102)*) which is a CCRT internal document that is not supplied to the customer.

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

Additionally, the software package is accompanied by an extensive set of application programming interface (API) calls that allow the user to access all capabilities of the board. The API also allows the user the ability to communicate directly with the board through the *ioctl(2)* and *mmap(2)* system calls. In this case, there is a risk of 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 Output Driver Installation on RedHawk Release Notes by Concurrent Real-TIme.
- PCIe 8-Channel or 32-Channel Digital to Analog Output Converter Card (AOCC) Design Specification (No. 0610102) by Concurrent Real-Time *(internal document)*.

2. Software Support

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

2.1 Direct Driver Access

2.1.1 open(2) system call

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

```
int    fp;
fp = open("/dev/ccuraocc0", 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 writes to complete. In that case, if the write is not satisfied, only partial write will occur. The device name specified is of the format "/dev/ccuraocc<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.

The driver creates a duplicate set of device names in the following format: *"/dev/ccuraocc_wave<num>".* The optional wave generation API uses this name when opening this device.

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_CCURAOCC_ABORT_DMA
IOCTL_CCURAOCC_ADD_IRQ
IOCTL_CCURAOCC_DISABLE_PCI_INTERRUPTS
IOCTL_CCURAOCC_ENABLE_PCI_INTERRUPTS
IOCTL_CCURAOCC_GET_DRIVER_ERROR
IOCTL_CCURAOCC_GET_DRIVER_INFO
IOCTL_CCURAOCC_GET_PHYSICAL_MEMORY
IOCTL_CCURAOCC_GET_READ_MODE
IOCTL_CCURAOCC_GET_WRITE_MODE
IOCTL_CCURAOCC_INIT_BOARD
IOCTL_CCURAOCC_INTERRUPT_TIMEOUT_SECONDS
IOCTL_CCURAOCC_MAIN_CONTROL_REGISTERS
IOCTL_CCURAOCC_MMAP_SELECT
IOCTL_CCURAOCC_NO_COMMAND
IOCTL_CCURAOCC_PCI_BRIDGE_REGISTERS
IOCTL_CCURAOCC_PCI_CONFIG_REGISTERS
IOCTL_CCURAOCC_READ_EEPROM
IOCTL_CCURAOCC_REMOVE_IRQ
IOCTL_CCURAOCC_RESET_BOARD
IOCTL_CCURAOCC_SELECT_READ_MODE
IOCTL_CCURAOCC_SELECT_WRITE_MODE
IOCTL_CCURAOCC_WAIT_FOR_INTERRUPT
IOCTL_CCURAOCC_WRITE_EEPROM
```

IOCTL_CCURAOCC_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.

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

IOCTL *CCURAOCC_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_CCURAOCC_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_CCURAOCC_GET_DRIVER_ERROR: The argument supplied to this *ioctl* is a pointer to the *ccuraocc_user_error_t* structure. Information on the structure is located in the *ccuraocc_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 *CCURAOCC_SUCCESS.*

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

IOCTL_CCURAOCC_GET_PHYSICAL_MEMORY: The argument supplied to this *ioctl* is a pointer to the *ccuraocc_phys_mem_t* structure. Information on the structure is located in the *ccuraocc_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_CCURAOCC_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 _ccuraocc_driver_rw_mode_t* located in the *ccuraocc_user.h* include file. Though this is an analog output card, the user can read last values of the channel registers that were written to. If user is writing data to the board using the on-board FIFO, then the channel registers would reflect the most recent FIFO data that was output by the board. FIFO operation is not supported by the read mode as the FIFO is a write only register.

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

IOCTL_CCURAOCC_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_CCURAOCC_RESET_BOARD* call.

IOCTL_CCURAOCC_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 write call will wait before it times out. The call could time out if either the FIFO fails to drain or a DMA fails to complete. The device should have been opened in the block mode (*O_NONBLOCK* not set) for writes to wait for an operation to complete.

IOCTL_CCURAOCC_MAIN_CONTROL_REGISTERS: This *ioctl* dumps all the PCI Main Control registers and is mainly used for debug purpose. The argument to this *ioctl* is a pointer to the *ccuraocc_main_control_register_t* structure. Raw 32-bit data values are read from the board and loaded into this structure.

IOCTL_CCURAOCC_MMAP_SELECT: The argument to this *ioctl* is a pointer to the *ccuraocc_mmap_select_t* structure. Information on the structure is located in the *ccuraocc_user.h* include file. This call needs to be made prior to the *mmap(2)* system call so as to direct the following *mmap(2)* call to perform the requested mapping specified by this *ioctl*. The four possible mappings that are performed by the driver are to *mmap* the local register space *(CCURAOCC_SELECT_LOCAL_MMAP)*, the configuration register space *(CCURAOCC_SELECT_CONFIG_MMAP),* the physical memory *(CCURAOCC_SELECT_PHYS_MEM_MMAP)* and the *(CCURAOCC_SELECT_DRIVER_LIBRARY_MMAP)* that is created by the *mmap(2)* system call.

IOCTL_CCURAOCC_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_CCURAOCC_PCI_BRIDGE_REGISTERS: This *ioctl* dumps all the PCI bridge registers and is mainly used for debug purpose. The argument to this *ioctl* is a pointer to the *ccuraocc_pci_bridge_register_t* structure. Raw 32-bit data values are read from the board and loaded into this structure.

IOCTL_CCURAOCC_PCI_CONFIG_REGISTERS: This *ioctl* dumps all the PCI configuration registers and is mainly used for debug purpose. The argument to this *ioctl* is a pointer to the *ccuraocc_pci_config_reg_addr_mapping_t* structure. Raw 32-bit data values are read from the board and loaded into this structure.

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

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

IOCTL_CCURAOCC_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 _ccuraocc_driver_rw_mode_t* located in the *ccuraocc_user.h* include file. FIFO operation is not supported by the read mode as the FIFO is a write only register.

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

IOCTL_CCURAOCC_WAIT_FOR_INTERRUPT: The argument to this *ioctl* is a pointer to the *ccuraocc_driver_int_t* structure. Information on the structure is located in the *ccuraocc_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.

IOCTL_CCURAOCC_WRITE_EEPROM: The argument to this *ioctl* is a pointer to the *ccuraocc_eeprom_t* structure. Information on the structure is located in the *ccuraocc_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_CCURAOCC_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;
ccuraocc_local_ctrl_data_t  *local_ptr;
ccuraocc_mmap_select_t  mmap_select;
unsigned long mmap_local_size;

mmap_select.select = CCURAOCC_SELECT_LOCAL_MMAP;
mmap_select.offset=0;
```

```
        mmap_select.size=0;

        ioctl(fp, IOCTL_CCURAOCC_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 = (ccuraocc_local_ctrl_data_t *)munmap_local_ptr;
        local_ptr = (ccuraocc_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 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_CCURAOCC_SELECT_READ_MODE* command. The following are the possible read modes:

CCURAOCC_PIO_CHANNEL: This mode returns the data that was last written to the FIFO or the channel registers 1 to 32 for the 32-Channel card or 1 to 8 for the 8-Channel card. The relative offset within the returned buffer determines the channel number. The data content is an 18-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 output by the hardware.

CCURAOCC_DMA_CHANNEL: This mode of operation is identical to the *CCURAOCC_PIO_CHANNEL* mode with the exception that the driver performs a DMA operation instead of Programmed I/O to complete the operation.

2.1.5 write(2) system call

Prior to issuing this call to write, the user needs to select the type of write operation they would like to perform. The only reason for providing various write 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 write mode is specified by the *ioctl* call with the *IOCTL_CCURAOCC_SELECT_WRITE_MODE* command. The following are the possible write modes:

CCURAOCC_PIO_CHANNEL: This mode writes from 1 to 32 channels for the 32-Channel card and 1 to 8 for the 8-Channel card raw data to the channel registers.. The relative offset within the write buffer determines the channel number. The data content is an 18-bit analog output raw value. The driver uses Programmed I/O to perform this operation. In this mode, samples written are immediately sent out to the channels by the hardware based on the setting of the synchronization flags.

CCURAOCC_DMA_CHANNEL: This mode of operation is identical to the *CCURAOCC_PIO_CHANNEL* mode with the exception that the driver performs a DMA operation instead of Programmed I/O to complete the operation.

CCURAOCC_PIO_FIFO: This mode writes selected channels raw data to the channel registers. The channels to be written are first selected by the *channel_select* register mask. The data content is an 18-bit analog output raw value. The driver uses Programmed I/O to perform this operation. In this mode, samples written to the hardware FIFO register, which are in turn clocked out to the channels by either internal or external clocking.

CCURAOCC_DMA_FIFO: This mode is identical to the *CCURAOCC_PIO_FIFO* mode with the exception that writes are performed using DMA operation.

For both of the above FIFO operations, the following operation is common:
- In order to synchronize channels, the channel *converter_csr* needs to set the synchronized mode, otherwise, the channels will be updated immediately when the data is read from the FIFO.
- The *channel_select* register determines which set of registers are being placed in the FIFO.
- When the user requests a write of sample size, the routine checks to see if there is sufficient room available in the FIFO to perform the complete write. If true, then the write operation is carried out and completed immediately. If there are insufficient open space in the FIFO to completely satisfy the write operation, the write routine then checks whether the user has selected the *O_NONBLOCK* flag during opening the device, then a partial write will take place filling the current available space in the FIFO and returning. If the *O_NONBLOCK* flag is not set during opening the device, the driver will block waiting for enough samples to be available to complete the write. The duration of blocking is a direct function of the number of channels in the FIFO and the sample rate.

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.

ccurAOCC_Abort_DMA()
ccurAOCC_Add_Irq()
ccurAOCC_Clear_Driver_Error()
ccurAOCC_Clear_Lib_Error()
ccurAOCC_Close()
ccurAOCC_Compute_PLL_Clock()
ccurAOCC_Create_Factory_Calibration()
ccurAOCC_Create_User_Checkpoint()
ccurAOCC_DataToVolts()
ccurAOCC_DataToVoltsChanCal()
ccurAOCC_Disable_Pci_Interrupts()
ccurAOCC_Enable_Pci_Interrupts()
ccurAOCC_Fast_Memcpy()
ccurAOCC_Fast_Memcpy_Unlocked()
ccurAOCC_Fraction_To_Hex()
ccurAOCC_Get_Board_CSR()
ccurAOCC_Get_Board_Info()
ccurAOCC_Get_Calibrator_ADC_Control()
ccurAOCC_Get_Calibrator_ADC_Data()
ccurAOCC_Get_Calibrator_ADC_NegativeGainCal()
ccurAOCC_Get_Calibrator_ADC_OffsetCal()
ccurAOCC_Get_Calibrator_ADC_PositiveGainCal()
ccurAOCC_Get_Calibrator_Bus_Control()
ccurAOCC_Get_Calibration_ChannelGain()
ccurAOCC_Get_Calibration_ChannelOffset()
ccurAOCC_Get_Channel_Selection()
ccurAOCC_Get_Converter_Clock_Divider()
ccurAOCC_Get_Converter_CSR()
ccurAOCC_Get_Converter_Update_Selection()
ccurAOCC_Get_Driver_Error()
ccurAOCC_Get_Driver_Info()
ccurAOCC_Get_Driver_Read_Mode()
ccurAOCC_Get_Driver_Write_Mode()
ccurAOCC_Get_Fifo_Driver_Threshold()
ccurAOCC_Get_Fifo_Info()
ccurAOCC_Get_Fifo_Threshold()
ccurAOCC_Get_Interrupt_Control()
ccurAOCC_Get_Interrupt_Status()
ccurAOCC_Get_Interrupt_Timeout_Seconds()
ccurAOCC_Get_Lib_Error_Description()
ccurAOCC_Get_Lib_Error()
ccurAOCC_Get_Mapped_Config_Ptr()
ccurAOCC_Get_Mapped_Driver_Library_Ptr()
ccurAOCC_Get_Mapped_Local_Ptr()
ccurAOCC_Get_Open_File_Descriptor()
ccurAOCC_Get_Physical_Memory()
ccurAOCC_Get_PLL_Info()
ccurAOCC_Get_PLL_Status()
ccurAOCC_Get_PLL_Sync()
ccurAOCC_Get_Sample_Rate()
ccurAOCC_Get_TestBus_Control()

ccurAOCC_Get_Value()
ccurAOCC_Hex_To_Fraction()
ccurAOCC_Identify_Board()
ccurAOCC_Initialize_Board()
ccurAOCC_Initialize_PLL_Input_Struct()
ccurAOCC_MMap_Physical_Memory()
ccurAOCC_Munmap_Physical_Memory()
ccurAOCC_NanoDelay()
ccurAOCC_Open()
ccurAOCC_Open_Wave()
ccurAOCC_Perform_ADC_Calibration()
ccurAOCC_Perform_Channel_Gain_Calibration()
ccurAOCC_Perform_Channel_Offset_Calibration()
ccurAOCC_Perform_Auto_Calibration()
ccurAOCC_Program_PLL_Advanced()
ccurAOCC_Program_PLL_Clock()
ccurAOCC_Program_Sample_Rate()
ccurAOCC_Read()
ccurAOCC_Read_Channels()
ccurAOCC_Read_Channels_Calibration()
ccurAOCC_Read_Serial_Prom()
ccurAOCC_Read_Serial_Prom_Item()
ccurAOCC_Read_Single_Channel()
ccurAOCC_Remove_Irq()
ccurAOCC_Reset_ADC_Calibrator()
ccurAOCC_Reset_Board()
ccurAOCC_Reset_Channel_Calibration()
ccurAOCC_Reset_Fifo()
ccurAOCC_Restore_Factory_Calibration()
ccurAOCC_Restore_User_Checkpoint()
ccurAOCC_Select_Driver_Read_Mode()
ccurAOCC_Select_Driver_Write_Mode()
ccurAOCC_Serial_Prom_Write_Override()
ccurAOCC_Set_Board_CSR()
ccurAOCC_Set_Calibrator_ADC_Control()
ccurAOCC_Set_Calibrator_ADC_NegativeGainCal()
ccurAOCC_Set_Calibrator_ADC_OffsetCal()
ccurAOCC_Set_Calibrator_ADC_PositiveGainCal()
ccurAOCC_Set_Calibrator_Bus_Control()
ccurAOCC_Set_Calibration_ChannelGain()
ccurAOCC_Set_Calibration_ChannelOffset()
ccurAOCC_Set_Channel_Selection()
ccurAOCC_Set_Converter_Clock_Divider()
ccurAOCC_Set_Converter_CSR()
ccurAOCC_Set_Converter_Update_Selection()
ccurAOCC_Set_Fifo_Driver_Threshold()
ccurAOCC_Set_Fifo_Threshold()
ccurAOCC_Set_Interrupt_Control()
ccurAOCC_Set_Interrupt_Status()
ccurAOCC_Set_Interrupt_Timeout_Seconds()
ccurAOCC_Set_PLL_Sync()
ccurAOCC_Set_TestBus_Control()
ccurAOCC_Set_Value()
ccurAOCC_Shutdown_PLL_Clock()
ccurAOCC_Start_PLL_Clock()
ccurAOCC_Stop_PLL_Clock()

ccurAOCC_View_Factory_Calibration()
ccurAOCC_View_User_Checkpoint()
ccurAOCC_VoltsToData()
ccurAOCC_VoltsToDataChanCal()
ccurAOCC_Wait_For_Channel_Idle()
ccurAOCC_Wait_For_Interrupt()
ccurAOCC_Write()
ccurAOCC_Write_Channels()
ccurAOCC_Write_Channels_Calibration()
ccurAOCC_Write_Serial_Prom()
ccurAOCC_Write_Serial_Prom_Item()
ccurAOCC_Write_Single_Channel()

2.2.1 ccurAOCC_Abort_DMA()

This call will abort any DMA operation that is in progress. Normally, the user should not use this call unless they are providing their own DMA handling.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Abort_DMA (void *Handle)

    Description: Abort any DMA in progress

    Input:      void *Handle                    (handle pointer)
    Output:     none
    Return:     _ccuraocc_lib_error_number_t
                - CCURAOCC_LIB_NO_ERROR          (successful)
                - CCURAOCC_LIB_BAD_HANDLE        (no/bad handler supplied)
                - CCURAOCC_LIB_NOT_OPEN          (device not open)
                - CCURAOCC_LIB_NO_LOCAL_REGION   (error)
                - CCURAOCC_LIB_IOCTL_FAILED      (error)
 ****************************************************************************/
```

2.2.2 ccurAOCC_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.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_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 ccurAOCC_Remove_Irq(), then this call adds it back.

    Input:      void *Handle                    (handle pointer)
    Output:     None
    Return:     _ccuraocc_lib_error_number_t
                - CCURAOCC_LIB_NO_ERROR          (successful)
                - CCURAOCC_LIB_BAD_HANDLE        (no/bad handler supplied)
                - CCURAOCC_LIB_NOT_OPEN          (device not open)
                - CCURAOCC_LIB_IOCTL_FAILED      (driver ioctl call failed)
 ****************************************************************************/
```

2.2.3 ccurAOCC_Clear_Driver_Error()

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

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Clear_Driver_Error (void *Handle)

    Description: Clear any previously generated driver related error.

    Input:      void *Handle                    (handle pointer)
    Output:     None
    Return:     _ccuraocc_lib_error_number_t
                - CCURAOCC_LIB_NO_ERROR          (successful)
                - CCURAOCC_LIB_BAD_HANDLE        (no/bad handler supplied)
                - CCURAOCC_LIB_NOT_OPEN          (device not open)
                - CCURAOCC_LIB_IOCTL_FAILED      (driver ioctl call failed)
 ****************************************************************************/
```

2.2.4 ccurAOCC_Clear_Lib_Error()

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

```
/******************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Clear_Lib_Error (void *Handle)

    Description: Clear any previously generated library related error.

    Input:      void *Handle                    (handle pointer)
    Output:     None
    Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR      (successful)
                 - CCURAOCC_LIB_BAD_HANDLE    (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN      (device not open)
 ******************************************************************************
```

2.2.5 ccurAOCC_Close()

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

```
/******************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Close (void *Handle)

    Description: Close a previously opened device.

    Input:      void *Handle                    (handle pointer)
    Output:     None
    Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR      (successful)
                 - CCURAOCC_LIB_BAD_HANDLE    (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN      (device not open)
 ******************************************************************************/
```

2.2.6 ccurAOCC_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 *ccuraocc_pll.c* file located in the *.../test/lib* directory for usage of this call. Refer to the *.../lib/ccuraocc_lib.h* include file for structure definitions.

```
/******************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Compute_PLL_Clock (void                 *Handle,
                                ccuraocc_PLL_setting_t *input,
                                ccuraocc_solution_t    *solution)

    Description: Return the value of the specified PLL information.

    Input:      void                 *Handle    (handle pointer)
                ccuraocc_PLL_setting_t *input    (pll input setting)
    Output:     ccuraocc_solution_t    *solution; (pointer to solution struct)
    Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR      (successful)
                 - CCURAOCC_LIB_BAD_HANDLE    (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN      (device not open)
                 - CCURAOCC_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 */
} ccuraocc_PLL_setting_t;
```

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

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


typedef struct {
    int                        NREF;
    int                        NFBK;
    ccuraocc_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;
    ccuraocc_PLL_struct_t      setup;
} ccuraocc_solution_t;
```

2.2.7 ccurAOCC_Create_Factory_Calibration()

This routine is used by Concurrent Real-Time to program factory calibration into the serial prom for each 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 *ccurAOCC_Serial_Prom_Write_Override()* to allowing writing to the serial prom. The supporting calls for this API are *ccurAOCC_View_Factory_Calibration()* and *ccurAOCC_Restore_Factory_Calibration()*.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Create_Factory_Calibration (void                  *Handle,
                                         _ccuraocc_sprom_access_t item,
                                         char                  *filename,
                                         ccuraocc_bool         force)

    Description: Create a Factory Calibration from user specified file
```

```
Input:          void                    *Handle (handlepointer)
                _ccuraocc_sprom_access_t item    (select item)
                  -- CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V
                  -- CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V
                  -- CCURAOCC_SPROM_FACTORY_BIPOLAR_5V
                  -- CCURAOCC_SPROM_FACTORY_BIPOLAR_10V
                  -- CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V
                char                    *filename (pointer to filename)
                ccuraocc_bool           force    (force programming)
                  -- CCURAOCC_TRUE
                  -- CCURAOCC_FALSE
Output:         none
Return:         _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR           (successful)
                  - CCURAOCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN           (device not open)
                  - CCURAOCC_LIB_CANNOT_OPEN_FILE   (file not readable)
                  - CCURAOCC_LIB_NO_LOCAL_REGION    (error)
                  - CCURAOCC_LIB_SERIAL_PROM_BUSY   (serial prom busy)
                  - CCURAOCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
                  - CCURAOCC_LIB_INVALID_CRC        (invalid CRC)
                  - CCURAOCC_LIB_INVALID_ARG        (invalid argument)
**************************************************************************/
```

The *item* can be one of the following factory voltage ranges:

```
typedef enum {
    CCURAOCC_SPROM_HEADER=1,
    CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V,
    CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V,
    CCURAOCC_SPROM_FACTORY_BIPOLAR_5V,
    CCURAOCC_SPROM_FACTORY_BIPOLAR_10V,
    CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V,
    CCURAOCC_SPROM_USER_CHECKPOINT_1,
    CCURAOCC_SPROM_USER_CHECKPOINT_2,
} _ccuraocc_sprom_access_t;
```

The *filename* contains the *offset* and *gain* in floating point for each channel. This file can be created with the *ccurAOCC_Write_Channels_Calibration()* API, once the card has been calibrated for all channels with a specific voltage range. The *ccuraocc_calibrate* utility can be used to create this file (*./ccuraocc_calibrate –Vb10 –oCalOut_b10*). The third argument *Range* in the calibration file is ignored in this *ccurAOCC_Create_Factory_Calibration()* routine. It is up to the user to ensure that the correct file is supplied for the selected voltage range.

Sample file for all channels configured for bipolar 10 volts:

```
#Date           : Tue Mar 25 12:45:24 2014
#Board Serial No: 12345678 (0x00bc614e)

#Chan   Offset                 Gain                 Range
#====   ==================     ==================   ============
 ch00:  -0.0213623046875000    -0.0119018554687500  BiPolar 10v
 ch01:  -0.0503540039062500    -0.0396728515625000  BiPolar 10v
 ch02:   0.2633666992187500     0.5798339843750000  BiPolar 10v
 ch03:  -0.0027465820312500     0.0497436523437500  BiPolar 10v
 ch04:  -0.1342773437500000    -0.2017211914062500  BiPolar 10v
 ch05:  -0.1959228515625000    -0.3466796875000000  BiPolar 10v
 ch06:  -0.0250244140625000     0.0170898437500000  BiPolar 10v
 ch07:   0.1223754882812500     0.3179931640625000  BiPolar 10v
 ch08:   0.1010131835937500     0.2215576171875000  BiPolar 10v
 ch09:  -0.0607299804687500    -0.0958251953125000  BiPolar 10v
 ch10:   0.0299072265625000     0.0997924804687500  BiPolar 10v
```

```
ch11:    0.0881958007812500    0.2145385742187500  BiPolar 10v
ch12:   -0.0018310546875000    0.0003051757812500  BiPolar 10v
ch13:    0.0851440429687500    0.2136230468750000  BiPolar 10v
ch14:    0.0775146484375000    0.1760864257812500  BiPolar 10v
ch15:    0.0289916992187500    0.0781250000000000  BiPolar 10v
ch16:    0.0024414062500000   -0.0180053710937500  BiPolar 10v
ch17:    0.3225708007812500    0.7015991210937500  BiPolar 10v
ch18:    0.1724243164062500    0.3021240234375000  BiPolar 10v
ch19:    0.0872802734375000    0.1937866210937500  BiPolar 10v
ch20:    0.0973510742187500    0.2261352539062500  BiPolar 10v
ch21:   -0.0057983398437500    0.0051879882812500  BiPolar 10v
ch22:   -0.0097656250000000   -0.0253295898437500  BiPolar 10v
ch23:    0.2059936523437500    0.4101562500000000  BiPolar 10v
ch24:    0.0607299804687500    0.1651000976562500  BiPolar 10v
ch25:    0.1062011718750000    0.2593994140625000  BiPolar 10v
ch26:   -0.1159667968750000   -0.1934814453125000  BiPolar 10v
ch27:    0.0329589843750000    0.1181030273437500  BiPolar 10v
ch28:   -0.0424194335937500   -0.0390625000000000  BiPolar 10v
ch29:   -0.1092529296875000   -0.1565551757812500  BiPolar 10v
ch30:   -0.0247192382812500    0.0076293945312500  BiPolar 10v
ch31:   -0.0567626953125000   -0.0656127929687500  BiPolar 10v
```

The *force* variable can be set to either *CCURAOCC_TRUE* or *CCURAOCC_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 *CCURAOCC_FALSE,* the call will fail.

2.2.8 ccurAOCC_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 *ccurAOCC_View_User_Checkpoint()* API that contains offset, gain 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, disconnect the outputs to the test equipment so as not to cause any damage to it, configure the individual channels for appropriate voltage ranges, 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 *ccurAOCC_Restore_User_Checkpoint() API.*

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

```
/*************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Create_User_Checkpoint (void                     *Handle,
                                     _ccuraocc_sprom_access_t item,
                                     char                     *filename,
                                     ccuraocc_bool            force)

    Description: Create a User Checkpoint from user specified file

    Input:       void                     *Handle   (handle pointer)
                 _ccuraocc_sprom_access_t item     (select item)
                   -- CCURAOCC_SPROM_USER_CHECKPOINT_1
                   -- CCURAOCC_SPROM_USER_CHECKPOINT_2
                 char                     *filename (pointer to filename or NULL)
                 ccuraocc_bool            force    (force programming)
                   -- CCURAOCC_TRUE
                   -- CCURAOCC_FALSE
```

```
        Output:        none
        Return:        _ccuraocc_lib_error_number_t
                          - CCURAOCC_LIB_NO_ERROR            (successful)
                          - CCURAOCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                          - CCURAOCC_LIB_NOT_OPEN            (device not open)
                          - CCURAOCC_LIB_CANNOT_OPEN_FILE    (file not readable)
                          - CCURAOCC_LIB_NO_LOCAL_REGION     (error)
                          - CCURAOCC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
                          - CCURAOCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
                          - CCURAOCC_LIB_INVALID_CRC         (invalid CRC)
                          - CCURAOCC_LIB_INVALID_ARG         (invalid argument)
          ************************************************************************/


        typedef enum {
            CCURAOCC_SPROM_HEADER=1,
            CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V,
            CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V,
            CCURAOCC_SPROM_FACTORY_BIPOLAR_5V,
            CCURAOCC_SPROM_FACTORY_BIPOLAR_10V,
            CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V,
            CCURAOCC_SPROM_USER_CHECKPOINT_1,
            CCURAOCC_SPROM_USER_CHECKPOINT_2,
        } _ccuraocc_sprom_access_t;
```

The *filename* contains the *converter CSR*, *offset* and *gain* in floating point for each channel. This file can be created with the *ccurAOCC_View_User_Checkpoint()* API, once the card has been calibrated and information stored in the serial PROM with this *ccurAOCC_Create_User_Checkpoint()* and filename set to *NULL*.

Below is a sample file for all channels configured for varying voltage ranges. User needs to refer to the hardware programming manual to get information on the converter CSR register.

```
# User Checkpoint from serial prom
#            Date: Tue Mar 25 13:46:02 EDT 2014
#      Checkpoint: User Checkpoint 1
# Board Serial No: 12345678 (0x00bc614e)
#             CRC: 1A64
#
#Chan   Offset                Gain                  Converter Csr
#====   ======                ====                  =============
 ch00: -0.0247192382812500   -0.0198364257812500    0x00000003
 ch01:  0.0198364257812500    0.0057983398437500    0x00000001
 ch02:  0.2603149414062500    0.5737304687500000    0x00000003
 ch03:  0.0234985351562500    0.0814819335937500    0x00000001
 ch04: -0.1391601562500000   -0.2117919921875000    0x00000003
 ch05:  0.0100708007812500   -0.3005981445312500    0x00000001
 ch06: -0.0302124023437500    0.0051879882812500    0x00000003
 ch07:  0.0167846679687500    0.3506469726562500    0x00000001
 ch08:  0.1013183593750000    0.2279663085937500    0x00000003
 ch09: -0.0665283203125000   -0.1065063476562500    0x00000003
 ch10:  0.0112915039062500    0.0625610351562500    0x00000003
 ch11:  0.0903320312500000    0.2209472656250000    0x00000003
 ch12:  0.0057983398437500    0.0015258789062500    0x00000002
 ch13:  0.0775146484375000    0.1983642578125000    0x00000002
 ch14:  0.0833129882812500    0.1864624023437500    0x00000002
 ch15:  0.0292968750000000    0.0659179687500000    0x00000002
 ch16: -0.0042724609375000   -0.0311279296875000    0x00000003
 ch17:  0.3076171875000000    0.6713867187500000    0x00000003
 ch18:  0.1687622070312500    0.2954101562500000    0x00000003
 ch19:  0.0747680664062500    0.1699829101562500    0x00000003
 ch20:  0.0820922851562500    0.1928710937500000    0x00000003
 ch21: -0.0198364257812500   -0.0231933593750000    0x00000003
 ch22: -0.0238037109375000   -0.0509643554687500    0x00000003
 ch23:  0.1971435546875000    0.3942871093750000    0x00000003
```

```
ch24:    0.0732421875000000     0.1361083984375000    0x00000004
ch25:    0.1171875000000000     0.2380371093750000    0x00000004
ch26:   -0.1086425781250000    -0.2108764648437500    0x00000004
ch27:    0.0552368164062500     0.1199340820312500    0x00000004
ch28:   -0.0314331054687500    -0.0656127929687500    0x00000004
ch29:   -0.0958251953125000    -0.1699829101562500    0x00000004
ch30:   -0.0079345703125000     0.0036621093750000    0x00000004
ch31:   -0.0323486328125000    -0.0527954101562500    0x00000004
```

The *force* variable can be set to either *CCURAOCC_TRUE* or *CCURAOCC_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 *CCURAOCC_FALSE,* the call will fail.

2.2.9 ccurAOCC_DataToVolts()

This routine takes a raw analog input data value and converts it to a floating point voltage based on the supplied *format* and *voltage range*.

```
/***************************************************************************
   double
   ccurAOCC_DataToVolts (int us_data,
                         int format,
                         int select_voltage_range)

   Description: Convert Data to volts

   Input:       int     us_data              (data to convert)
                int     format               (conversion format)
                int     select_voltage_range (select voltage range)
   Output:      none
   Return:      double  volts                (returned volts)
   ***************************************************************************/
```

The *format* can be: CCURAOCC_CONVERTER_OFFSET_BINARY
                     CCURAOCC_CONVERTER_TWOS_COMPLEMENT

If an invalid *format* is supplied, the call defaults to CCURAOCC_CONVERTER_OFFSET_BINARY.

The *select_voltage_range* can be: CCURAOCC_CONVERTER_UNIPOLAR_5V
                                   CCURAOCC_CONVERTER_UNIPOLAR_10V
                                   CCURAOCC_CONVERTER_BIPOLAR_5V
                                   CCURAOCC_CONVERTER_BIPOLAR_10V
                                   CCURAOCC_CONVERTER_BIPOLAR_2_5V

If the data to volts conversion is for the on-board Analog to Digital Converter (ADC), nicknamed *"Calibrator"*, then the following parameters to be supplied to the *select_voltage_range.*

    CCURAOCC_CALADC_RANGE_BIPOLAR_5V
    CCURAOCC_CALADC_RANGE_BIPOLAR_10V
    CCURAOCC_CALADC_RANGE_BIPOLAR_20V

If an invalid voltage range is selected, the call defaults to CCURAOCC_CONVERTER_UNIPOLAR_5V.

2.2.10 ccurAOCC_DataToVoltsChanCal()

This call converts raw data to volts for calibration registers.

```
/***************************************************************************
   double
   ccurAOCC_DataToVoltsChanCal (int us_data)
```

```
        Description: Convert Data to Volts (for Channel Calibration Registers)

        Input:      int     us_data         (data to convert)
        Output:     none
        Return:     double  volts           (returned volts)
        **************************************************************************/
```

2.2.11 ccurAOCC_Disable_Pci_Interrupts()

This call disables PCI interrupts. This call shouldn't be used during normal reads as writes could time out. The driver handles enabling and disabling interrupts during its normal course of operation.

```
/**************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Disable_Pci_Interrupts (void *Handle)

    Description: Disable interrupts being generated by the board.

    Input:      void *Handle                    (handle pointer)
    Output:     None
    Return:     _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR     (successful)
                    - CCURAOCC_LIB_BAD_HANDLE   (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN     (device not open)
                    - CCURAOCC_LIB_IOCTL_FAILED (driver ioctl call failed)
    **************************************************************************/
```

2.2.12 ccurAOCC_Enable_Pci_Interrupts()

This call enables 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.

```
/**************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Enable_Pci_Interrupts (void *Handle)

    Description: Enable interrupts being generated by the board.

    Input:      void *Handle                    (handle pointer)
    Output:     None
    Return:     _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR     (successful)
                    - CCURAOCC_LIB_BAD_HANDLE   (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN     (device not open)
                    - CCURAOCC_LIB_IOCTL_FAILED (driver ioctl call failed)
    **************************************************************************/
```

2.2.13 ccurAOCC_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.

```
/**************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Fast_Memcpy(void           *Handle,
                         volatile void *Destination,
                         volatile void *Source,
                         int           SizeInBytes)

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

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

```
                    volatile void *Source       (pointer to source buffer)
                    int           SizeInBytes   (transfer size in bytes)
    Output:         volatile void *Destination  (pointer to destination buffer)
    Return:         _ccuraocc_lib_error_number_t
                      - CCURAOCC_LIB_NO_ERROR    (successful)
                      - CCURAOCC_LIB_BAD_HANDLE  (no/bad handler supplied)
                      - CCURAOCC_LIB_NOT_OPEN    (device not open)
                      - CCURAOCC_LIB_INVALID_ARG (invalid argument)
    ********************************************************************/
```

2.2.14 ccurAOCC_Fast_Memcpy_Unlocked()

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

```
/************************************************************************
    void
    ccurAOCC_Fast_Memcpy_Unlocked(volatile void *Destination,
                                  volatile void *Source,
                                  int           SizeInBytes)

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

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

2.2.15 ccurAOCC_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
    ccurAOCC_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)
    *************************************************************************
```

2.2.16 ccurAOCC_Get_Board_CSR()

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

```
/*************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Board_CSR (void                  *Handle,
                            ccuraocc_board_csr_t *bcsr)

    Description: Get Board Control and Status information

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

```
      Output:       ccuraocc_board_csr_t   *bcsr      (pointer to board csr)
      Return:       _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR        (successful)
                    - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN        (device not open)
                    - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                    - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
   ***************************************************************************


   typedef struct
   {
       int external_clock_detected;     /* external clock detected */
       int all_converter_reset;         /* all converter reset */
       int external_clock_output;       /* external clock selection */
       int identify_board;              /* identify board */
   } ccuraocc_board_csr_t;
```

// external_clock_detected
- CCURAOCC_BCSR_EXTCLK_NOT_DETECTED
- CCURAOCC_BCSR_EXTCLK_DETECTED

// all_converter_reset
- CCURAOCC_BCSR_ALL_CONVERTER_ACTIVE
- CCURAOCC_BCSR_ALL_CONVERTER_RESET

// external_clock_output
- CCURAOCC_BCSR_EXTCLK_OUTPUT_SOFTWARE_FLAG
- CCURAOCC_BCSR_EXTCLK_OUTPUT_PLL_CLOCK
- CCURAOCC_BCSR_EXTCLK_OUTPUT_EXTERNAL_CLOCK

// identify_board
- CCURAOCC_BCSR_IDENTIFY_BOARD_DISABLE
- CCURAOCC_BCSR_IDENTIFY_BOARD_ENABLE

2.2.17 ccurAOCC_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 *0x9287* and board type is *0x1*=Differential, *0x2*=Single-Ended.

```
   /***************************************************************************
      _ccuraocc_lib_error_number_t
      ccurAOCC_Get_Board_Info (void                  *Handle,
                            ccuraocc_board_info_t *binfo)

      Description: Get Board Information

      Input:        void                   *Handle  (handle pointer)
      Output:       ccuraocc_board_info_t   *binfo  (pointer to board info)
      Return:       _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR        (successful)
                    - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN        (device not open)
                    - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                    - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
   ***************************************************************************


   typedef struct
   {
       int     board_id;                /* board id */
       int     board_type;              /* board type */
       int     firmware_rev;            /* firmware revision */
```

```
        ccuraocc_sprom_header_t sprom_header;
                                        /* serial prom header information */
        int     board_wiring;           /* single-ended or differential */
        int     number_of_channels;     /* number of hardware channels */
        int     number_of_converters;   /* number of converters */
        int     all_channels_mask;      /* all channels mask */
        int     all_converters_mask;    /* all converters mask */
        double  cal_ref_voltage;        /* calibration reference voltage */
        double  voltage_range;          /* maximum voltage range */
        double  MinSampleFreq;          /* minimum sample frequency */
        double  MaxSampleFreq;          /* maximum sample frequency */
        double  MasterClock;            /* master clock */
        int     number_of_calbus_channels; /* number of calibration bus channels */
} ccuraocc_board_info_t;
```

2.2.18 ccurAOCC_Get_Calibrator_ADC_Control()

The board has an on-board Analog to Digital Converter (ADC) that is used during calibration of the channels. This call returns the ADC control and range information. Normally, the user does not need this API. It is used internally by the API to calibrate the channels.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Calibrator_ADC_Control (void                      *Handle,
                                   _ccuraocc_calib_adc_control_t *adc_control,
                                   _ccuraocc_calib_adc_range_t   *adc_range)

    Description: Get Calibrator ADC Control Information

    Input:      void          *Handle          (handle pointer)
    Output:     _ccuraocc_calib_adc_control_t
                        *adc_control      (pointer to cal ADC control)
                _ccuraocc_calib_adc_range_t
                        *adc_range        (pointer to cal ADC range)
    Return:     _ccuraocc_lib_error_number_t
                   - CCURAOCC_LIB_NO_ERROR       (successful)
                   - CCURAOCC_LIB_NO_LOCAL_REGION (local region error)
                   - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                   - CCURAOCC_LIB_NOT_OPEN        (device not open)
                   - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                   - CCURAOCC_LIB_CALIBRATION_RANGE_ERROR
                                                  (calibration range error)
 *****************************************************************************/

typedef enum
{
    CCURAOCC_CALADC_CONTROL_BIPOLAR_0_5V   = (0), /*  0V  to  +5V (10V p-p) */
    CCURAOCC_CALADC_CONTROL_BIPOLAR_0_10V  = (1), /*  0V  to +10V (20V p-p) */
    CCURAOCC_CALADC_CONTROL_BIPOLAR_5_5V   = (2), /* -5V  to  +5V (20V p-p) */
    CCURAOCC_CALADC_CONTROL_BIPOLAR_10_10V = (3), /* -10V to +10V (40V p-p) */
} _ccuraocc_calib_adc_control_t;

typedef enum
{
    CCURAOCC_CALADC_RANGE_BIPOLAR_5V  = (CCURAOCC_CONVERTER_BIPOLAR_5V),
    CCURAOCC_CALADC_RANGE_BIPOLAR_10V = (CCURAOCC_CONVERTER_BIPOLAR_10V),
    CCURAOCC_CALADC_RANGE_BIPOLAR_20V = (99),  /* any number not in range 0..3 */
                                               /* for Cal ADC Control Only */
} _ccuraocc_calib_adc_range_t;
```

2.2.19 ccurAOCC_Get_Calibrator_ADC_Data()

The call returns to the user the current ADC data register, both in raw value and floating point volts.

```
/*****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Get_Calibrator_ADC_Data (void   *Handle,
                                     uint   *raw_data,
                                     double *volts)

   Description: Get Calibrator ADC Date Information

   Input:      void                 *Handle     (handle pointer)
   Output:     uint                 *raw_data   (pointer to cal ADC data)
               double               *volts      (pointer to cal ADC data)
   Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR        (successful)
                 - CCURAOCC_LIB_NO_LOCAL_REGION (local region error)
                 - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN        (device not open)
                 - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
 *****************************************************************************/
```

2.2.20 ccurAOCC_Get_Calibrator_ADC_NegativeGainCal()

The call returns to the user the current ADC negative gain calibration register, both in raw value and floating point volts.

```
/*****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Get_Calibrator_ADC_NegativeGainCal (void   *Handle,
                                                uint   *Raw,
                                                double *Float)

   Description: Get Calibrator ADC Negative Gain Data

   Input:      void            *Handle      (handle pointer)
   Output:     uint            *Raw         (pointer to Raw ADC Cal)
               double          *Float       (pointer to Float ADC Cal)
   Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR        (successful)
                 - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                 - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN        (device not open)
                 - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
 *****************************************************************************/
```

2.2.21 ccurAOCC_Get_Calibrator_ADC_OffsetCal()

The call returns to the user the current ADC offset calibration register, both in raw value and floating point volts.

```
/*****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Get_Calibrator_ADC_OffsetCal (void   *Handle,
                                          uint   *Raw,
                                          double *Float)

   Description: Get Calibrator ADC Offset Data

   Input:      void            *Handle      (handle pointer)
   Output:     uint            *Raw         (pointer to Raw ADC Cal)
               double          *Float       (pointer to Float ADC Cal)
   Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR        (successful)
                 - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
```

```
                            - CCURAOCC_LIB_BAD_HANDLE        (no/bad handler supplied)
                            - CCURAOCC_LIB_NOT_OPEN          (device not open)
                            - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
             ******************************************************************************/
```

2.2.22 ccurAOCC_Get_Calibrator_ADC_PositiveGainCal()

The call returns to the user the current ADC positive gain calibration register, both in raw value and floating point volts.

```
/******************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Calibrator_ADC_PositiveGainCal (void    *Handle,
                                                 uint    *Raw,
                                                 double *Float)

    Description: Get Calibrator ADC Positive Gain Data

    Input:      void                *Handle      (handle pointer)
    Output:     uint                *Raw         (pointer to Raw ADC Cal)
                double              *Float       (pointer to Float ADC Cal)
    Return:     _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR       (successful)
                    - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                    - CCURAOCC_LIB_BAD_HANDLE     (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN       (device not open)
                    - CCURAOCC_LIB_INVALID_ARG    (invalid argument)
             ******************************************************************************/
```

2.2.23 ccurAOCC_Get_Calibrator_Bus_Control()

The ADC *(calibrator)* can only return information for one element at a time. Prior to reading the ADC data, the user needs to select the element whose information is to be returned. This call returns to the user the current connection to the calibrator bus.

```
/******************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Calibrator_Bus_Control (void                     *Handle,
                                  _ccuraocc_calib_bus_control_t *adc_bus_control)

    Description: Get Calibration Bus Control Information

    Input:      void        *Handle              (handle pointer)
    Output:     _ccuraocc_calib_bus_control_t
                            *adc_bus_control      (pointer to cal Bus control)
    Return:     _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR       (successful)
                    - CCURAOCC_LIB_NO_LOCAL_REGION (local region error)
                    - CCURAOCC_LIB_BAD_HANDLE     (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN       (device not open)
                    - CCURAOCC_LIB_INVALID_ARG    (invalid argument)
             ******************************************************************************/


    typedef enum
    {
        CCURAOCC_CALBUS_CONTROL_GROUND       = (0),
        CCURAOCC_CALBUS_CONTROL_POSITIVE_REF = (1),
        CCURAOCC_CALBUS_CONTROL_NEGATIVE_REF = (2),
        CCURAOCC_CALBUS_CONTROL_OPEN         = (3),

        CCURAOCC_CALBUS_CONTROL_CHAN_0       = (0x20),  // 8 & 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_1       = (0x21),  // 8 & 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_2       = (0x22),  // 8 & 32-channel card
```

```
        CCURAOCC_CALBUS_CONTROL_CHAN_3        = (0x23),   // 8 & 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_4        = (0x24),   // 8 & 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_5        = (0x25),   // 8 & 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_6        = (0x26),   // 8 & 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_7        = (0x27),   // 8 & 32-channel card

        CCURAOCC_CALBUS_CONTROL_CHAN_8        = (0x28),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_9        = (0x29),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_10       = (0x2A),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_11       = (0x2B),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_12       = (0x2C),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_13       = (0x2D),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_14       = (0x2E),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_15       = (0x2F),   // 32-channel card

        CCURAOCC_CALBUS_CONTROL_CHAN_16       = (0x30),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_17       = (0x31),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_18       = (0x32),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_19       = (0x33),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_20       = (0x34),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_21       = (0x35),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_22       = (0x36),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_23       = (0x37),   // 32-channel card

        CCURAOCC_CALBUS_CONTROL_CHAN_24       = (0x38),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_25       = (0x39),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_26       = (0x3A),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_27       = (0x3B),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_28       = (0x3C),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_29       = (0x3D),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_30       = (0x3E),   // 32-channel card
        CCURAOCC_CALBUS_CONTROL_CHAN_31       = (0x3F),   // 32-channel card

    } _ccuraocc_calib_bus_control_t;
```

2.2.24 ccurAOCC_Get_Calibration_ChannelGain()

This single call can be used to read back the selected channel *gain* raw hardware registers. Additionally, the call returns the floating point value of the register as well.

```
/**************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Calibration_ChannelGain (void                    *Handle,
                                          _ccuraocc_channel_mask_t chan_mask,
                                          ccuraocc_converter_cal_t *gain)

    Description: Get Calibration Channel Gain

    Input:      void                     *Handle   (handle pointer)
                _ccuraocc_channel_mask_t chan_mask (selected channel mask)
    Output:     ccuraocc_converter_cal_t *gain     (gain value)
    Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR        (successful)
                 - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                 - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
 **************************************************************************/

typedef enum
{
    CCURAOCC_CHANNEL_MASK_0  = 0x00000001,      /* chan 0 : 8 & 32-channel card */
    CCURAOCC_CHANNEL_MASK_1  = 0x00000002,      /* chan 1 : 8 & 32-channel card */
    CCURAOCC_CHANNEL_MASK_2  = 0x00000004,      /* chan 2 : 8 & 32-channel card */
    CCURAOCC_CHANNEL_MASK_3  = 0x00000008,      /* chan 3 : 8 & 32-channel card */
```

```
        CCURAOCC_CHANNEL_MASK_4  = 0x00000010,      /* chan 4 : 8 & 32-channel card */
        CCURAOCC_CHANNEL_MASK_5  = 0x00000020,      /* chan 5 : 8 & 32-channel card */
        CCURAOCC_CHANNEL_MASK_6  = 0x00000040,      /* chan 6 : 8 & 32-channel card */
        CCURAOCC_CHANNEL_MASK_7  = 0x00000080,      /* chan 7 : 8 & 32-channel card */

        CCURAOCC_CHANNEL_MASK_8  = 0x00000100,      /* chan 8 : 32-channel card */
        CCURAOCC_CHANNEL_MASK_9  = 0x00000200,      /* chan 9 : 32-channel card */
        CCURAOCC_CHANNEL_MASK_10 = 0x00000400,      /* chan 0 : 32-channel card */
        CCURAOCC_CHANNEL_MASK_11 = 0x00000800,      /* chan 11: 32-channel card */
        CCURAOCC_CHANNEL_MASK_12 = 0x00001000,      /* chan 12: 32-channel card */
        CCURAOCC_CHANNEL_MASK_13 = 0x00002000,      /* chan 13: 32-channel card */
        CCURAOCC_CHANNEL_MASK_14 = 0x00004000,      /* chan 14: 32-channel card */
        CCURAOCC_CHANNEL_MASK_15 = 0x00008000,      /* chan 15: 32-channel card */

        CCURAOCC_CHANNEL_MASK_16 = 0x00010000,      /* chan 16: 32-channel card */
        CCURAOCC_CHANNEL_MASK_17 = 0x00020000,      /* chan 17: 32-channel card */
        CCURAOCC_CHANNEL_MASK_18 = 0x00040000,      /* chan 18: 32-channel card */
        CCURAOCC_CHANNEL_MASK_19 = 0x00080000,      /* chan 19: 32-channel card */
        CCURAOCC_CHANNEL_MASK_20 = 0x00100000,      /* chan 20: 32-channel card */
        CCURAOCC_CHANNEL_MASK_21 = 0x00200000,      /* chan 21: 32-channel card */
        CCURAOCC_CHANNEL_MASK_22 = 0x00400000,      /* chan 22: 32-channel card */
        CCURAOCC_CHANNEL_MASK_23 = 0x00800000,      /* chan 23: 32-channel card */

        CCURAOCC_CHANNEL_MASK_24 = 0x01000000,      /* chan 24: 32-channel card */
        CCURAOCC_CHANNEL_MASK_25 = 0x02000000,      /* chan 25: 32-channel card */
        CCURAOCC_CHANNEL_MASK_26 = 0x04000000,      /* chan 26: 32-channel card */
        CCURAOCC_CHANNEL_MASK_37 = 0x08000000,      /* chan 27: 32-channel card */
        CCURAOCC_CHANNEL_MASK_28 = 0x10000000,      /* chan 28: 32-channel card */
        CCURAOCC_CHANNEL_MASK_29 = 0x20000000,      /* chan 29: 32-channel card */
        CCURAOCC_CHANNEL_MASK_30 = 0x40000000,      /* chan 30: 32-channel card */
        CCURAOCC_CHANNEL_MASK_31 = 0x80000000,      /* chan 31: 32-channel card */

        /* End Channel */
        CCURAOCC_ALL_CHANNEL_MASK = 0xFFFFFFFF,
} _ccuraocc_channel_mask_t;


typedef struct
{
    uint   Raw[CCURAOCC_MAX_CHANNELS];
    double Float[CCURAOCC_MAX_CHANNELS];
} ccuraocc_converter_cal_t;
```

2.2.25 ccurAOCC_Get_Calibration_ChannelOffset()

This single call can be used to read back the selected channel *offset* raw hardware registers. Additionally, the call returns the floating point value of the register as well.

```
/******************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Calibration_ChannelOffset (void                    *Handle,
                                     _ccuraocc_channel_mask_t chan_mask,
                                     ccuraocc_converter_cal_t *offset)

    Description: Get Calibration Channel Offset

    Input:      void                    *Handle   (handle pointer)
                _ccuraocc_channel_mask_t chan_mask (selected channel mask)
    Output:     ccuraocc_converter_cal_t *offset   (offset value)
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR         (successful)
                  - CCURAOCC_LIB_NO_LOCAL_REGION  (local region not present)
                  - CCURAOCC_LIB_INVALID_ARG      (invalid argument)
```

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

        Information on structures are described in the above API ccurAOCC_Get_Calibration_ChannelGain().
```

2.2.26 ccurAOCC_Get_Channel_Selection()

This API returns the current channel selection mask that is used during FIFO write operations.

```
        /****************************************************************************
           _ccuraocc_lib_error_number_t
           ccurAOCC_Get_Channel_Selection (void *Handle,
                                              uint *channel_select)

           Description: Get Channel_Selection

           Input:      void           *Handle         (handle pointer)
           Output:     uint           *channel_select (channel selection mask)
           Return:     _ccuraocc_lib_error_number_t
                          - CCURAOCC_LIB_NO_ERROR       (successful)
                          - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                          - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
         ****************************************************************************/
```

Information on structure is described in the above API ccurAOCC_Get_Calibration_ChannelGain().

2.2.27 ccurAOCC_Get_Converter_Clock_Divider()

This API returns the current clock divider register information.

```
        /****************************************************************************
           _ccuraocc_lib_error_number_t
           ccurAOCC_Get_Converter_Clock_Divider (void *Handle,
                                                     uint *divider)

           Description: Get Converter Clock Divider

           Input:      void           *Handle         (handle pointer)
           Output:     uint           *divider         (pointer to clock divider)
           Return:     _ccuraocc_lib_error_number_t
                          - CCURAOCC_LIB_NO_ERROR       (successful)
                          - CCURAOCC_LIB_BAD_HANDLE       (no/bad handler supplied)
                          - CCURAOCC_LIB_NOT_OPEN         (device not open)
                          - CCURAOCC_LIB_INVALID_ARG      (invalid argument)
                          - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
         ****************************************************************************/
```

2.2.28 ccurAOCC_Get_Converter_CSR()

This call returns control information on the selected converter. The converter cannot be written to while the *CCURAOCC_CONVERTER_BUSY* flag is set in the *converter_interface_busy* field.

```
        /****************************************************************************
           _ccuraocc_lib_error_number_t
           ccurAOCC_Get_Converter_CSR (void                      *Handle,
                                        _ccuraocc_converter_mask_t conv_mask,
                                        ccuraocc_converter_csr_t   ccsr)

           Description: Get Converter Control and Status information

           Input:      void                      *Handle  (handle pointer)
                        _ccuraocc_converter_mask_t conv_mask (selected converter)
           Output:     ccuraocc_converter_csr_t   ccsr       (converter csr)
           Return:     _ccuraocc_lib_error_number_t
```

```
                    - CCURAOCC_LIB_NO_ERROR             (successful)
                    - CCURAOCC_LIB_BAD_HANDLE           (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN             (device not open)
                    - CCURAOCC_LIB_INVALID_ARG          (invalid argument)
                    - CCURAOCC_LIB_NO_LOCAL_REGION      (local region not present)
    ************************************************************************/

typedef enum
{
    CCURAOCC_CONVERTER_MASK_0  = 0x00000001,    /* chan 0 : 8 & 32-channel card */
    CCURAOCC_CONVERTER_MASK_1  = 0x00000002,    /* chan 1 : 8 & 32-channel card */
    CCURAOCC_CONVERTER_MASK_2  = 0x00000004,    /* chan 2 : 8 & 32-channel card */
    CCURAOCC_CONVERTER_MASK_3  = 0x00000008,    /* chan 3 : 8 & 32-channel card */
    CCURAOCC_CONVERTER_MASK_4  = 0x00000010,    /* chan 4 : 8 & 32-channel card */
    CCURAOCC_CONVERTER_MASK_5  = 0x00000020,    /* chan 5 : 8 & 32-channel card */
    CCURAOCC_CONVERTER_MASK_6  = 0x00000040,    /* chan 6 : 8 & 32-channel card */
    CCURAOCC_CONVERTER_MASK_7  = 0x00000080,    /* chan 7 : 8 & 32-channel card */

    CCURAOCC_CONVERTER_MASK_8  = 0x00000100,    /* chan 8 : 32-channel card */
    CCURAOCC_CONVERTER_MASK_9  = 0x00000200,    /* chan 9 : 32-channel card */
    CCURAOCC_CONVERTER_MASK_10 = 0x00000400,    /* chan 10: 32-channel card */
    CCURAOCC_CONVERTER_MASK_11 = 0x00000800,    /* chan 11: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_12 = 0x00001000,    /* chan 12: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_13 = 0x00002000,    /* chan 13: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_14 = 0x00004000,    /* chan 14: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_15 = 0x00008000,    /* chan 15: 32-channel card  */

    CCURAOCC_CONVERTER_MASK_16 = 0x00010000,    /* chan 16: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_17 = 0x00020000,    /* chan 17: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_18 = 0x00040000,    /* chan 18: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_19 = 0x00080000,    /* chan 19: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_20 = 0x00100000,    /* chan 20: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_21 = 0x00200000,    /* chan 21: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_22 = 0x00400000,    /* chan 22: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_23 = 0x00800000,    /* chan 23: 32-channel card  */

    CCURAOCC_CONVERTER_MASK_24 = 0x01000000,    /* chan 24: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_25 = 0x02000000,    /* chan 25: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_26 = 0x04000000,    /* chan 26: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_37 = 0x08000000,    /* chan 27: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_28 = 0x10000000,    /* chan 28: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_29 = 0x20000000,    /* chan 29: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_30 = 0x40000000,    /* chan 30: 32-channel card  */
    CCURAOCC_CONVERTER_MASK_31 = 0x80000000,    /* chan 31: 32-channel card  */

    /* End Converter */
    CCURAOCC_ALL_CONVERTER_MASK = 0xFFFFFFFF,
} _ccuraocc_converter_mask_t;

typedef struct
{
    int converter_interface_busy;
    int converter_update_mode;
    int converter_data_format;
    int converter_output_range;
} _ccuraocc_converter_csr_t;

typedef _ccuraocc_converter_csr_t
    ccuraocc_converter_csr_t[CCURAOCC_MAX_CONVERTERS];
```

// converter_interface_busy
- CCURAOCC_CONVERTER_IDLE
- CCURAOCC_CONVERTER_BUSY

```
// converter_update_mode
- CCURAOCC_CONVERTER_MODE_IMMEDIATE
- CCURAOCC_CONVERTER_MODE_SYNCHRONIZED
- CCURAOCC_DO_NOT_CHANGE

// converter_data_format
- CCURAOCC_CONVERTER_OFFSET_BINARY
- CCURAOCC_CONVERTER_TWOS_COMPLEMENT
- CCURAOCC_DO_NOT_CHANGE

// converter_output_range
- CCURAOCC_CONVERTER_UNIPOLAR_5V
- CCURAOCC_CONVERTER_UNIPOLAR_10V
- CCURAOCC_CONVERTER_BIPOLAR_5V
- CCURAOCC_CONVERTER_BIPOLAR_10V
- CCURAOCC_CONVERTER_BIPOLAR_2_5V
- CCURAOCC_DO_NOT_CHANGE
```

2.2.29 ccurAOCC_Get_Converter_Update_Selection()

This API provides user with the converter update selection information.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Converter_Update_Selection (void                      *Handle,
                                    _ccuraocc_converter_update_select_t *select)

    Description: Get Converter Update Selection Information

    Input:      void                               *Handle (handle pointer)
    Output:     _ccuraocc_converter_update_select_t *select
                                                (pointer to converter update info)
    Return:     _ccuraocc_lib_error_number_t
                   - CCURAOCC_LIB_NO_ERROR           (successful)
                   - CCURAOCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                   - CCURAOCC_LIB_NOT_OPEN           (device not open)
                   - CCURAOCC_LIB_INVALID_ARG        (invalid argument)
                   - CCURAOCC_LIB_NO_LOCAL_REGION    (local region not present)
 ****************************************************************************/

typedef enum
{
    CCURAOCC_CONVERTER_UPDATE_SELECT_SOFTWARE = (0),
    CCURAOCC_CONVERTER_UPDATE_SELECT_PLL_CLOCK = (1),
    CCURAOCC_CONVERTER_UPDATE_SELECT_EXTERNAL_CLOCK = (4),
} _ccuraocc_converter_update_select_t;
```

2.2.30 ccurAOCC_Get_Driver_Error()

This call returns the last error generated by the driver.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Driver_Error (void                  *Handle,
                            ccuraocc_user_error_t *ret_err)

    Description: Get the last error generated by the driver.

    Input:      void                  *Handle  (handle pointer)
    Output:     ccuraocc_user_error_t *ret_err  (error struct pointer)
```

```
                      -- uint error                                (last library error
                                                                    number)
                      -- char name[CCURAOCC_LIB_ERROR_NAME_SIZE] (last library error
                                                                    name)
                      -- char desc[CCURAOCC_LIB_ERROR_DESC_SIZE] (last library error
                                                                    description)
                      -- int  line_number                        (last library error
                                                                    line number in lib)
    Return:         _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR       (successful)
                    - CCURAOCC_LIB_BAD_HANDLE     (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN       (device not open)
                    - CCURAOCC_LIB_INVALID_ARG    (invalid argument)
                    - CCURAOCC_LIB_IOCTL_FAILED   (driver ioctl call
   **************************************************************************/

   #define CCURAOCC_ERROR_NAME_SIZE    64
   #define CCURAOCC_ERROR_DESC_SIZE    128

   typedef struct _ccuraocc_user_error_t
   {
       uint error;                                /* error number */
       char name[CCURAOCC_ERROR_NAME_SIZE];       /* error name used in driver */
       char desc[CCURAOCC_ERROR_DESC_SIZE];       /* error description */
       uint    error_line_number;                 /* error line number */
   } ccuraocc_user_error_t;

   enum
   {
       CCURAOCC_SUCCESS = 0,
       CCURAOCC_INVALID_PARAMETER,
       CCURAOCC_FIFO_THRESHOLD_TIMEOUT,
       CCURAOCC_DMA_TIMEOUT,
       CCURAOCC_OPERATION_CANCELLED,
       CCURAOCC_RESOURCE_ALLOCATION_ERROR,
       CCURAOCC_INVALID_REQUEST,
       CCURAOCC_FAULT_ERROR,
       CCURAOCC_BUSY,
       CCURAOCC_ADDRESS_IN_USE,
       CCURAOCC_USER_INTERRUPT_TIMEOUT,
       CCURAOCC_DMA_INCOMPLETE,
       CCURAOCC_DATA_UNDERFLOW,
       CCURAOCC_DATA_OVERFLOW,
       CCURAOCC_IO_FAILURE,
       CCURAOCC_PCI_ABORT_INTERRUPT_ACTIVE,
   };
```

2.2.31 ccurAOCC_Get_Driver_Info()

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

```
   /****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Get_Driver_Error (void                *Handle,
                       ccuraocc_user_error_t *ret_err)

   Description: Get device information from driver.

   Input:      void                 *Handle (handle pointer)
   Output:     ccuraocc_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
                -- int                    board_wiring
                -- int                    number_of_channels
                -- int                    number_of_converters
                -- int                    all_channels_mask
                -- int                    all_converters_mask
                -- int                    max_fifo_samples
                -- int                    max_fifo_data
                -- int                    max_fifo_threshold
                -- int                    max_dma_samples
                -- int                    dma_size
                -- double                 cal_ref_voltage
                -- double                 voltage_range
                -- ccuraocc_driver_int_t interrupt
                -- int                    Ccuraocc_Max_Region
                -- ccuraocc_dev_region_t mem_region[CCURAOCC_MAX_REGION]
                -- ccuraocc_sprom_header_t sprom_header
                -- int                    number_of_calbus_channels
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR      (successful)
                  - CCURAOCC_LIB_BAD_HANDLE    (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN      (device not open)
                  - CCURAOCC_LIB_INVALID_ARG   (invalid argument)
                  - CCURAOCC_LIB_IOCTL_FAILED  (driver ioctl call failed)
*************************************************************************/


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

typedef   struct
{
    uint   physical_address;
    uint   size;
    uint   flags;
    uint   *virtual_address;
} ccuraocc_dev_region_t;

typedef struct {
    u_int   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 */
} ccuraocc_sprom_header_t;
#define CCURAOCC_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 */
        int board_wiring;                  /* single_ended, differential */
        int number_of_channels;            /* number of channels in this board */
        int number_of_converters;          /* number of converters in this board */
        int all_channels_mask;             /* all channels mask */
        int all_converters_mask;           /* all converters mask */
        int max_fifo_samples;              /* maximum fifo samples */
        int max_fifo_data;                 /* maximum fifo data */
        int max_fifo_threshold;            /* maximum fifo threshold */
        int max_dma_samples;               /* maximum DMA samples */
        int dma_size;                      /* DMA size in bytes */
        double cal_ref_voltage;            /* calibration ref voltage */
        double voltage_range;              /* board voltage range */
        ccuraocc_driver_int_t interrupt;/* interrupt information */
        int Ccuraocc_Max_Region;           /*kernel DEVICE_COUNT_RESOURCE */
        ccuraocc_dev_region_t mem_region[CCURAOCC_MAX_REGION];
                                           /* memory region */
        ccuraocc_sprom_header_t sprom_header;
                                           /* serial prom header */
        int number_of_calbus_channels;  /* number of calibration bus channels in this
                                            board */
    } ccuraocc_driver_info_t;
```

2.2.32 ccurAOCC_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.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Driver_Read_Mode (void                    *Handle,
                                   _ccuraocc_driver_rw_mode_t *mode)

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

    Input:       void                    *Handle (handle pointer)
    Output:      _ccuraocc_driver_rw_mode_t *mode   (pointer to read mode)
    Return:      _ccuraocc_lib_error_number_t
                   - CCURAOCC_LIB_NO_ERROR          (successful)
                   - CCURAOCC_LIB_BAD_HANDLE        (no/bad handler supplied)
                   - CCURAOCC_LIB_NOT_OPEN          (device not open)
                   - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
                   - CCURAOCC_LIB_NO_LOCAL_REGION   (local region error)
                   - CCURAOCC_LIB_IOCTL_FAILED      (ioctl error)
 *****************************************************************************/

    typedef enum
    {
        CCURAOCC_PIO_CHANNEL,       /* read/write mode */
```

```
        CCURAOCC_DMA_CHANNEL,        /* read/write mode */
        CCURAOCC_PIO_FIFO,           /* write mode */
        CCURAOCC_DMA_FIFO,           /* write mode */
} _ccuraocc_driver_rw_mode_t;
```

2.2.33 ccurAOCC_Get_Driver_Write_Mode()

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

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Driver_Write_Mode (void                        *Handle,
                                    _ccuraocc_driver_rw_mode_t *mode)

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

    Input:       void                         *Handle   (handle pointer)
    Output:      _ccuraocc_driver_rw_mode_t *mode      (pointer to write mode)
    Return:      _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR         (successful)
                  - CCURAOCC_LIB_BAD_HANDLE       (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN         (device not open)
                  - CCURAOCC_LIB_INVALID_ARG      (invalid argument)
                  - CCURAOCC_LIB_NO_LOCAL_REGION  (local region error)
                  - CCURAOCC_LIB_IOCTL_FAILED     (ioctl error)
 *****************************************************************************/

typedef enum
{
    CCURAOCC_PIO_CHANNEL,        /* read/write mode */
    CCURAOCC_DMA_CHANNEL,        /* read/write mode */
    CCURAOCC_PIO_FIFO,           /* write mode */
    CCURAOCC_DMA_FIFO,           /* write mode */
} _ccuraocc_driver_rw_mode_t;
```

2.2.34 ccurAOCC_Get_Fifo_Driver_Threshold()

This API returns to the user the FIFO threshold that was previously set by the user.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Fifo_Driver_Threshold (void *Handle,
                                        uint *threshold)

    Description: Get FIFO Driver Threshold

    Input:       void                         *Handle    (handle pointer)
    Output:      uint                         *threshold (pointer to driver threshold)
    Return:      _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR         (successful)
                  - CCURAOCC_LIB_BAD_HANDLE       (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN         (device not open)
                  - CCURAOCC_LIB_INVALID_ARG      (invalid argument)
 *****************************************************************************/
```

2.2.35 ccurAOCC_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.

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

```
            _ccuraocc_lib_error_number_t
            ccurAOCC_Get_Fifo_Info (void                    *Handle,
                                    ccuraocc_fifo_info_t *fifo)

            Description: Get FIFO Control and Status information

            Input:         void                    *Handle    (handle pointer)
            Output:        ccuraocc_fifo_info_t    *fifo      (pointer to board fifo)
            Return:        _ccuraocc_lib_error_number_t
                            - CCURAOCC_LIB_NO_ERROR        (successful)
                            - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                            - CCURAOCC_LIB_NOT_OPEN        (device not open)
                            - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                            - CCURAOCC_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;
            uint driver_threshold;
        } ccuraocc_fifo_info_t
```

// reset
- CCURAOCC_FIFO_ACTIVE
- CCURAOCC_FIFO_ACTIVATE   (same as CCURAOCC_FIFO_ACTIVE)
- CCURAOCC_FIFO_RESET
// overflow
- CCURAOCC_FIFO_NO_OVERFLOW
- CCURAOCC_FIFO_OVERFLOW

// underflow
- CCURAOCC_FIFO_NO_UNDERFLOW
- CCURAOCC_FIFO_UNDERFLOW

// full
- CCURAOCC_FIFO_NOT_FULL
- CCURAOCC_FIFO_FULL

// threshold_exceeded
- CCURAOCC_FIFO_THRESHOLD_NOT_EXCEEDED
- CCURAOCC_FIFO_THRESHOLD_EXCEEDED

// empty
- CCURAOCC_FIFO_NOT_EMPTY
- CCURAOCC_FIFO_EMPTY

// data_counter
- this field ranges from 0 to 0x20000 entries representing the number of samples currently present in the FIFO.

// threshold
- this field ranges from 0 to 0x20000 entries representing the number of samples in the FIFO where the threshold interrupt should occur. This is the current threshold that is read from the board.

// driver_threshold
- this field ranges from 0 to 0x20000 entries representing the number of samples in the FIFO that was last set by the user.

If the O_NONBLOCK flag is set when opening the device or the FIFO has enough empty space to write all the user specified samples in the write call, then the write will NOT block and all the samples will be written to the FIFO.

If the above is false, then the driver will do the following:
1) If the user specifies a count of samples to write to the FIFO that is greater than or equal to the number of samples that can be held above the user specified driver threshold, the driver will block until the samples in the FIFO drains to the user specified driver threshold.
2) If the user specifies a count of samples to write to the FIFO that is less than the number of samples that can be held above the user specified driver threshold, the driver will block waiting for samples to drain until there is sufficient room to write the user specified sample count.
3) With the above checks, the driver ensures that when the write completes, there are at least sufficient samples still in the FIFO close to the user defined threshold value. In this way, the user has time while the FIFO is draining the remaining samples before returning to write more samples to the FIFO.

2.2.36 ccurAOCC_Get_Fifo_Threshold()

This call simply returns the current hardware FIFO threshold register value.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Fifo_Threshold (void *Handle,
                                 uint *threshold)

    Description: Get FIFO Threshold

    Input:      void                    *Handle     (handle pointer)
    Output:     uint                    *threshold  (pointer to board threshold)
    Return:     _ccuraocc_lib_error_number_t
                - CCURAOCC_LIB_NO_ERROR         (successful)
                - CCURAOCC_LIB_BAD_HANDLE       (no/bad handler supplied)
                - CCURAOCC_LIB_NOT_OPEN         (device not open)
                - CCURAOCC_LIB_INVALID_ARG      (invalid argument)
    ****************************************************************************/
```

2.2.37 ccurAOCC_Get_Interrupt_Control()

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

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Interrupt_Control (void                  *Handle,
                                    ccuraocc_interrupt_t *intr)

    Description: Get Interrupt Control information

    Input:      void                    *Handle     (handle pointer)
    Output:     ccuraocc_interrupt_t   *intr       (pointer to interrupt control)
    Return:     _ccuraocc_lib_error_number_t
                - CCURAOCC_LIB_NO_ERROR         (successful)
                - CCURAOCC_LIB_BAD_HANDLE       (no/bad handler supplied)
                - CCURAOCC_LIB_NOT_OPEN         (device not open)
                - CCURAOCC_LIB_INVALID_ARG      (invalid argument)
                - CCURAOCC_LIB_NO_LOCAL_REGION  (local region not present)
```

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

    typedef struct {
        int     global_int;
        int     fifo_buffer_hi_lo_int;
        int     plx_local_int;
    } ccuraocc_interrupt_t;
```

// global_int
- CCURAOCC_ICSR_GLOBAL_DISABLE
- CCURAOCC_ICSR_GLOBAL_ENABLE

// fifo_buffer_hi_lo_int
- CCURAOCC_ICSR_FIFO_HILO_THRESHOLD_DISABLE
- CCURAOCC_ICSR_FIFO_HILO_THRESHOLD_ENABLE

// plx_local_int
- CCURAOCC_ICSR_LOCAL_PLX_DISABLE
- CCURAOCC_ICSR_LOCAL_PLX_ENABLE

2.2.38 ccurAOCC_Get_Interrupt_Status()

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

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Interrupt_Status (void                    *Handle,
                                   ccuraocc_interrupt_t *intr)

    Description: Get Interrupt Status information

    Input:      void                    *Handle    (handle pointer)
    Output:     ccuraocc_interrupt_t    *intr      (pointer to interrupt status)
    Return:     _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR         (successful)
                    - CCURAOCC_LIB_BAD_HANDLE       (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN         (device not open)
                    - CCURAOCC_LIB_INVALID_ARG      (invalid argument)
                    - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
    ****************************************************************************/

    typedef struct {
        int     global_int;
        int     fifo_buffer_hi_lo_int;
        int     plx_local_int;
    } ccuraocc_interrupt_t;
```

// global_int
- not used

// fifo_buffer_hi_lo_int
- CCURAOCC_ISR_FIFO_HILO_THRESHOLD_NONE
- CCURAOCC_ISR_FIFO_HILO_THRESHOLD_OCCURRED
// plx_local_int
- CCURAOCC_ISR_LOCAL_PLX_NONE
- CCURAOCC_ISR_LOCAL_PLX_OCCURRED

2.2.39 ccurAOCC_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.

```
/****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_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:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR        (successful)
                  - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN        (device not open)
                  - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                  - CCURAOCC_LIB_IOCTL_FAILED    (ioctl error)
 ****************************************************************************/
```

2.2.40 ccurAOCC_Get_Lib_Error_Description()

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

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

   ccurAOCC_Get_Lib_Error_Description()

   Description: Get Error Description of supplied error number.

   Input:  int                              ErrorNumber    (Library error number)
   Output: ccuraocc_lib_error_description_t  *lib_error_desc (error description struct pointer)
            -- int  found
            -- char name[CCURAOCC_LIB_ERROR_NAME_SIZE] (last library error name)
            -- char desc[CCURAOCC_LIB_ERROR_DESC_SIZE] (last library error description)
   Return: none

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

2.2.41 ccurAOCC_Get_Lib_Error()

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

```
/****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Get_Lib_Error (void                  *Handle,
                           ccuraocc_lib_error_t *lib_error)

   Description: Get last error generated by the library.

   Input:  void                  *Handle                 (Handle pointer)
   Output: ccuraocc_lib_error_t *lib_error               (error struct pointer)
            -- uint error                                 (last library error number)
            -- char name[CCURAOCC_LIB_ERROR_NAME_SIZE] (last library error name)
            -- char desc[CCURAOCC_LIB_ERROR_DESC_SIZE] (last library error
                                                         description)
            -- int  line_number                          (last library error line
                                                          number in lib)
            -- char function[CCURAOCC_LIB_ERROR_FUNC_SIZE]
                                                         (library function in error)
```

```
                           -- ccuraocc_lib_error_backtrace_t BT[CCURAOCC_BACK_TRACE_DEPTH]
                                                    (backtrace of errors)
                      -- int  line_number                 (line number in library)
                      -- char function[CCURAOCC_LIB_ERROR_FUNC_SIZE]
                                                    (library function)
        Return: _ccuraocc_lib_error_number_t
                  # CCURAOCC_LIB_NO_ERROR                 (successful)
                  # CCURAOCC_LIB_BAD_HANDLE               (no/bad handler supplied)
                  # CCURAOCC_LIB_NOT_OPEN                 (device not open)
      *************************************************************************/


    typedef struct
    {
        int  line_number;                            /* line number in library */
        char function[CCURAOCC_LIB_ERROR_FUNC_SIZE]; /* library function */
    } ccuraocc_lib_error_backtrace_t;

    typedef struct
    {
        uint error;                                  /* last library error number */
        char name[CCURAOCC_LIB_ERROR_NAME_SIZE];     /* last library error name */
        char desc[CCURAOCC_LIB_ERROR_DESC_SIZE];     /* last libarary error description */
        int  line_number;                            /* last library  error line number in
                                                        lib */
        char function[CCURAOCC_LIB_ERROR_FUNC_SIZE]; /* library function in error */
        ccuraocc_lib_error_backtrace_t BT[CCURAOCC_BACK_TRACE_DEPTH];
                                                     /* backtrace of errors */
    } ccuraocc_lib_error_t;
```

Possible library errors:

```
CCURAOCC_LIB_NO_ERROR                   =  0,  /* successful */
CCURAOCC_LIB_INVALID_ARG                = -1,  /* invalid argument */
CCURAOCC_LIB_ALREADY_OPEN               = -2,  /* already open */
CCURAOCC_LIB_OPEN_FAILED                = -3,  /* open failed */
CCURAOCC_LIB_BAD_HANDLE                 = -4,  /* bad handle */
CCURAOCC_LIB_NOT_OPEN                   = -5,  /* device not opened */
CCURAOCC_LIB_MMAP_SELECT_FAILED         = -6,  /* mmap selection failed */
CCURAOCC_LIB_MMAP_FAILED                = -7,  /* mmap failed */
CCURAOCC_LIB_MUNMAP_FAILED              = -8,  /* munmap failed */
CCURAOCC_LIB_NOT_MAPPED                 = -9,  /* not mapped */
CCURAOCC_LIB_ALREADY_MAPPED             = -10, /* already mapped */
CCURAOCC_LIB_IOCTL_FAILED               = -11, /* driver ioctl failed */
CCURAOCC_LIB_IO_ERROR                   = -12, /* i/o error */
CCURAOCC_LIB_INTERNAL_ERROR             = -13, /* internal library error */
CCURAOCC_LIB_NOT_IMPLEMENTED            = -14, /* call not implemented */
CCURAOCC_LIB_LOCK_FAILED                = -15, /* failed to get lib lock */
CCURAOCC_LIB_NO_LOCAL_REGION            = -16, /* local region not present */
CCURAOCC_LIB_NO_CONFIG_REGION           = -17, /* config region not present */
CCURAOCC_LIB_NO_SOLUTION_FOUND          = -18, /* no solution found */
CCURAOCC_LIB_CONVERTER_RESET            = -19, /* converter not active */
CCURAOCC_LIB_NO_RESOURCE                = -20, /* resource not available */
CCURAOCC_LIB_CALIBRATION_RANGE_ERROR    = -21, /* calibration voltage out of range */
CCURAOCC_LIB_FIFO_OVERFLOW              = -22, /* fifo overflow */
CCURAOCC_LIB_CANNOT_OPEN_FILE           = -23, /* cannot open file */
CCURAOCC_LIB_BAD_DATA_IN_CAL_FILE       = -24, /* bad date in calibration file */
CCURAOCC_LIB_CHANNEL_BUSY               = -25, /* channel busy */
CCURAOCC_LIB_SERIAL_PROM_BUSY           = -26, /* serial prom busy */
CCURAOCC_LIB_SERIAL_PROM_FAILURE        = -27, /* serial prom failure - malfunction or
                                                  not present */
CCURAOCC_LIB_INVALID_CRC                = -28, /* invalid CRC read */
CCURAOCC_LIB_SERIAL_PROM_WRITE_PROTECTED = -29, /* serial prom is write protected */
```

2.2.42 ccurAOCC_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 *ccuraocc_user.h* include file that is supplied with the driver.

```
/***************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Get_Mapped_Config_Ptr (void                        *Handle,
                                   ccuraocc_config_local_data_t **config_ptr)

   Description: Get mapped configuration pointer.
   Input:      void                        *Handle    (handle pointer)
   Output:     ccuraocc_config_local_data_t **config_ptr (config struct ptr)
               -- structure in ccuraocc_user.h
   Return:     _ccuraocc_lib_error_number_t
               - CCURAOCC_LIB_NO_ERROR              (successful)
               - CCURAOCC_LIB_BAD_HANDLE            (no/bad handler
                                                     supplied)
               - CCURAOCC_LIB_NOT_OPEN              (device not open)
               - CCURAOCC_LIB_INVALID_ARG           (invalid argument)
               - CCURAOCC_LIB_NO_CONFIG_REGION      (config region not
                                                     present)
***************************************************************************/
```

2.2.43 ccurAOCC_Get_Mapped_Driver_Library_Ptr()

This API provides a pointer to a shared driver/library structure. This is used internally between the driver and the library.

```
/***************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Get_Mapped_Driver_Library_Ptr (void                *Handle,
                              ccuraocc_driver_library_common_t **driver_lib_ptr)

   Description: Get mapped Driver/Library structure pointer.

   Input:      void                        *Handle (handle pointer)
   Output:     ccuraocc_driver_library_common_t **driver_lib_ptr
                                                   (driver_lib struct ptr)
               -- structure in ccuraocc_user.h
   Return:     _ccuraocc_lib_error_number_t
               - CCURAOCC_LIB_NO_ERROR              (successful)
               - CCURAOCC_LIB_BAD_HANDLE            (no/bad handler supplied)
               - CCURAOCC_LIB_NOT_OPEN              (device not open)
               - CCURAOCC_LIB_INVALID_ARG           (invalid argument)
               - CCURAOCC_LIB_NO_LOCAL_REGION       (local region not present)
***************************************************************************/
```

2.2.44 ccurAOCC_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 *ccuraocc_user.h* include file that is supplied with the driver.

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

```
        _ccuraocc_lib_error_number_t
        ccurAOCC_Get_Mapped_Local_Ptr (void                      *Handle,
                                    ccuraocc_local_ctrl_data_t **local_ptr)


        Description: Get mapped local pointer.

        Input:       void                      *Handle   (handle pointer)
        Output:      ccuraocc_local_ctrl_data_t **local_ptr (local struct ptr)
                     -- structure in ccuraocc_user.h
        Return:      _ccuraocc_lib_error_number_t
                       - CCURAOCC_LIB_NO_ERROR            (successful)
                       - CCURAOCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                       - CCURAOCC_LIB_NOT_OPEN            (device not open)
                       - CCURAOCC_LIB_INVALID_ARG         (invalid argument)
                       - CCURAOCC_LIB_NO_LOCAL_REGION     (local region not present)
        **************************************************************************/
```

2.2.45 ccurAOCC_Get_Open_File_Descriptor()

When the library *ccurAOCC_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.

```
        /**************************************************************************
        _ccuraocc_lib_error_number_t
        ccurAOCC_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:      _ccuraocc_lib_error_number_t
                       - CCURAOCC_LIB_NO_ERROR      (successful)
                       - CCURAOCC_LIB_BAD_HANDLE    (no/bad handler supplied)
                       - CCURAOCC_LIB_NOT_OPEN      (device not open)
                       - CCURAOCC_LIB_INVALID_ARG   (invalid argument)
        **************************************************************************/
```

2.2.46 ccurAOCC_Get_Physical_Memory()

This call returns to the user the physical memory pointer and size that was previously allocated by the *ccurAOCC_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.

```
        /**************************************************************************
        _ccuraocc_lib_error_number_t
        ccurAOCC_Get_Physical_Memory (void                 *Handle,
                                    ccuraocc_phys_mem_t *phys_mem)

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

        Input:       void                 *Handle   (handle pointer)
        Output:      ccuraocc_phys_mem_t *phys_mem  (mem struct pointer)
                     -- void *phys_mem
                     -- uint  phys_mem_size
        Return:      _ccuraocc_lib_error_number_t
                       - CCURAOCC_LIB_NO_ERROR      (successful)
                       - CCURAOCC_LIB_BAD_HANDLE    (no/bad handler supplied)
                       - CCURAOCC_LIB_NOT_OPEN      (device not open)
```

```
                          - CCURAOCC_LIB_INVALID_ARG   (invalid argument)
                          - CCURAOCC_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 */
     } ccuraocc_phys_mem_t;
```

2.2.47 ccurAOCC_Get_PLL_Info()

This call returns the programmed information for the PLL.

```
     /*****************************************************************************
        _ccuraocc_lib_error_number_t
        ccurAOCC_Get_PLL_Info (void                    *Handle,
                            ccuraocc_PLL_struct_t *info)

        Description: Return the value of the PLL information.

        Input:       void                    *Handle (handle pointer)
        Output:      ccuraocc_PLL_struct_t *info;   (pointer to pll info struct)
        Return:      _ccuraocc_lib_error_number_t
                       - CCURAOCC_LIB_NO_ERROR       (successful)
                       - CCURAOCC_LIB_BAD_HANDLE     (no/bad handler supplied)
                       - CCURAOCC_LIB_NOT_OPEN       (device not open)
                       - CCURAOCC_LIB_INVALID_ARG    (invalid argument)
     *****************************************************************************/


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

         uint        ref_freq_divider_src;   /* CCURAOCC_REF_DIVIDER_SRC_OSCILLATOR */
                                             /* CCURAOCC_REF_DIVIDER_SRC_PIN */

         uint        shutdown_1;             /* CCURAOCC_RUNNING */
                                             /* CCURAOCC_SHUTDOWN */

         uint        post_divider1;          /* CCURAOCC_POST_DIVIDER1_1 */
                                             /* CCURAOCC_POST_DIVIDER1_2 */
                                             /* CCURAOCC_POST_DIVIDER1_3 */
                                             /* CCURAOCC_POST_DIVIDER1_4 */
                                             /* CCURAOCC_POST_DIVIDER1_5 */
                                             /* CCURAOCC_POST_DIVIDER1_6 */
                                             /* CCURAOCC_POST_DIVIDER1_7 */
                                             /* CCURAOCC_POST_DIVIDER1_8 */
                                             /* CCURAOCC_POST_DIVIDER1_9 */
                                             /* CCURAOCC_POST_DIVIDER1_10*/
                                             /* CCURAOCC_POST_DIVIDER1_11 */
                                             /* CCURAOCC_POST_DIVIDER1_12 */

         uint        post_divider2;          /* CCURAOCC_POST_DIVIDER2_1 */
                                             /* CCURAOCC_POST_DIVIDER2_2 */
                                             /* CCURAOCC_POST_DIVIDER2_3 */
                                             /* CCURAOCC_POST_DIVIDER2_4 */
                                             /* CCURAOCC_POST_DIVIDER2_5 */
                                             /* CCURAOCC_POST_DIVIDER2_6 */
                                             /* CCURAOCC_POST_DIVIDER2_7 */
                                             /* CCURAOCC_POST_DIVIDER2_8 */
                                             /* CCURAOCC_POST_DIVIDER2_9 */
                                             /* CCURAOCC_POST_DIVIDER2_10*/
                                             /* CCURAOCC_POST_DIVIDER2_11 */
                                             /* CCURAOCC_POST_DIVIDER2_12 */
```

```
    uint        post_divider3;              /* CCURAOCC_POST_DIVIDER3_1 */
                                            /* CCURAOCC_POST_DIVIDER3_2 */
                                            /* CCURAOCC_POST_DIVIDER3_4 */
                                            /* CCURAOCC_POST_DIVIDER3_8 */

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

    uint        feedback_divider_src;       /* CCURAOCC_FEEDBACK_DIVIDER_SRC_VCO */
                                            /* CCURAOCC_FEEDBACK_DIVIDER_SRC_POST */

    uint        clock_output;               /* CCURAOCC_CLOCK_OUTPUT_PECL */
                                            /* CCURAOCC_CLOCK_OUTPUT_CMOS */

    uint        charge_pump_current;        /* CCURAOCC_CHARGE_PUMP_CURRENT_2UA */
                                            /* CCURAOCC_CHARGE_PUMP_CURRENT_4_5UA */
                                            /* CCURAOCC_CHARGE_PUMP_CURRENT_11UA */
                                            /* CCURAOCC_CHARGE_PUMP_CURRENT_22_5UA */

    uint        loop_resistor;              /* CCURAOCC_LOOP_RESISTOR_400K */
                                            /* CCURAOCC_LOOP_RESISTOR_133K */
                                            /* CCURAOCC_LOOP_RESISTOR_30K */
                                            /* CCURAOCC_LOOP_RESISTOR_12K */

    uint        loop_capacitor;             /* CCURAOCC_LOOP_CAPACITOR_185PF */
                                            /* CCURAOCC_LOOP_CAPACITOR_500PF */

    uint        sync_enable;                /* CCURAOCC_SYNC_DISABLE */
                                            /* CCURAOCC_SYNC_ENABLE */
    uint        sync_polarity;              /* CCURAOCC_SYNC_POLARITY_NEGATIVE */
                                            /* CCURAOCC_SYNC_POLARITY_POSITIVE */

    uint        shutdown_2;                 /* CCURAOCC_RUNNING */
                                            /* CCURAOCC_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 */
} ccuraocc_PLL_struct_t;
```

2.2.48 ccurAOCC_Get_PLL_Status()

This call returns the status of the PLL.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_PLL_Status (void                    *Handle,
                             ccuraocc_PLL_status_t *status)

    Description: Return the status of the PLL

    Input:      void                    *Handle     (handle pointer)
    Output:     ccuraocc_PLL_status_t *status;   (pointer to status struct)
    Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR        (successful)
                 - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN        (device not open)
                 - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                 - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
 *****************************************************************************/

typedef struct {
    uint    busy;
    uint    error;
```

```
} ccuraocc_PLL_status_t;
```

// PLL Interface Busy
- CCURAOCC_PLL_IDLE
- CCURAOCC_PLL_BUSY

// PLL Interface Error
- CCURAOCC_PLL_NO_ERROR
- CCURAOCC_PLL_ERROR

2.2.49 ccurAOCC_Get_PLL_Sync()

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

```
/***************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_PLL_Sync (void                  *Handle,
                           ccuraocc_PLL_sync_t *sync)

    Description: Return the value of the PLL Sync information.

    Input:        void                    *Handle      (handle pointer)
    Output:       ccuraocc_PLL_sync_t  *sync;       (pointer to pll sync struct)
    Return:       _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR        (successful)
                    - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN        (device not open)
                    - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                    - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
 ***************************************************************************/

typedef struct {
    uint    sync_start;
    uint    external_go;
    uint    external_sync;
} ccuraocc_PLL_sync_t;
```

// PLL Sync Start
- CCURAOCC_PLL_START
- CCURAOCC_PLL_STOP

// External Go
- CCURAOCC_EXTERNAL_GO_OUT_ENABLE
- CCURAOCC_EXTERNAL_GO_OUT_DISABLE

// External Sync
- CCURAOCC_EXTERNAL_SYNC_OUT_ENABLE
- CCURAOCC_EXTERNAL_SYNC_OUT_DISABLE

2.2.50 ccurAOCC_Get_Sample_Rate()

With this API, the user will be able to obtain the current sample rate, clock frequency and clock divider.

```
/***************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Get_Sample_Rate (void   *Handle,
                              double *sample_rate,
                              double *clock_freq,
                              uint   *divider)

    Description: Get Sample Rate
```

```
   Input:      void *Handle                      (handle pointer)
   Output:     double           *sample_rate     (pointer to sample rate SPS)
               double           *clock_freq      (pointer to clock freq MHz)
               uint             *divider         (pointer to divider)
   Return:     _ccuraocc_lib_error_number_t
               - CCURAOCC_LIB_NO_ERROR           (successful)
               - CCURAOCC_LIB_NO_LOCAL_REGION    (local region not present)
               - CCURAOCC_LIB_INVALID_ARG        (invalid argument)
  *****************************************************************************/
```

2.2.51 ccurAOCC_Get_TestBus_Control()

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

```
/*****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Get_TestBus_Control (void                      *Handle,
                                 _ccuraocc_testbus_control_t *test_control)

   Description: Return the value of the Test Bus control information

   Input:      void                    *Handle  (handle pointer)
   Output:     _ccuraocc_testbus_control_t *test_control
                                               (pointer to pll sync struct)
   Return:     _ccuraocc_lib_error_number_t
               - CCURAOCC_LIB_NO_ERROR           (successful)
               - CCURAOCC_LIB_NO_LOCAL_REGION    (local region error)
               - CCURAOCC_LIB_BAD_HANDLE         (no/bad handler supplied)
               - CCURAOCC_LIB_NOT_OPEN           (device not open)
               - CCURAOCC_LIB_INVALID_ARG        (invalid argument)
  *****************************************************************************/

typedef enum
{
    CCURAOCC_TBUS_CONTROL_OPEN    = (0),
    CCURAOCC_TBUS_CONTROL_CAL_BUS = (1),
    CCURAOCC_TBUS_CONTROL_5V_REF  = (2),
} _ccuraocc_testbus_control_t;
```

2.2.52 ccurAOCC_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 *CCURAOCC_CHANNEL_DATA*, *CCURAOCC_GAIN_CALIBRATION* and, *CCURAOCC_OFFSET_CALIBRATION* return *CCURAOCC_MAX_CHANNELS* unsigned integers. The *CCURAOCC_SPI_RAM* command returns *CCURAOCC_SPI_RAM_SIZE* unsigned integers.

```
/*****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Get_Value (void             *Handle,
                       CCURAOCC_CONTROL cmd,
                       void             *value)

   Description: Return the value of the specified board register.

   Input:      void             *Handle         (handle pointer)
               CCURAOCC_CONTROL  cmd             (register definition)
   Output:     void             *value;         (pointer to value)
   Return:     _ccuraocc_lib_error_number_t
               - CCURAOCC_LIB_NO_ERROR           (successful)
               - CCURAOCC_LIB_BAD_HANDLE         (no/bad handler supplied)
```

```
                              - CCURAOCC_LIB_NOT_OPEN          (device not open)
                              - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
                              - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
          ************************************************************************/

          typedef enum
          {
              CCURAOCC_BOARD_INFORMATION,            /* R Only */
              CCURAOCC_BOARD_CSR,                    /* R/W */

              CCURAOCC_INTERRUPT_CONTROL,            /* R/W */
              CCURAOCC_INTERRUPT_STATUS,             /* R/W */

              CCURAOCC_CONVERTER_CSR_0,              /* R/W */
              CCURAOCC_CONVERTER_CSR_1,              /* R/W */
              CCURAOCC_CONVERTER_CSR_2,              /* R/W */
              CCURAOCC_CONVERTER_CSR_3,              /* R/W */
              CCURAOCC_CONVERTER_CSR_4,              /* R/W */
              CCURAOCC_CONVERTER_CSR_5,              /* R/W */
              CCURAOCC_CONVERTER_CSR_6,              /* R/W */
              CCURAOCC_CONVERTER_CSR_7,              /* R/W */
              CCURAOCC_CONVERTER_CSR_8,              /* R/W */
              CCURAOCC_CONVERTER_CSR_9,              /* R/W */
              CCURAOCC_CONVERTER_CSR_10,             /* R/W */
              CCURAOCC_CONVERTER_CSR_11,             /* R/W */
              CCURAOCC_CONVERTER_CSR_12,             /* R/W */
              CCURAOCC_CONVERTER_CSR_13,             /* R/W */
              CCURAOCC_CONVERTER_CSR_14,             /* R/W */
              CCURAOCC_CONVERTER_CSR_15,             /* R/W */
              CCURAOCC_CONVERTER_CSR_16,             /* R/W */
              CCURAOCC_CONVERTER_CSR_17,             /* R/W */
              CCURAOCC_CONVERTER_CSR_18,             /* R/W */
              CCURAOCC_CONVERTER_CSR_19,             /* R/W */
              CCURAOCC_CONVERTER_CSR_20,             /* R/W */
              CCURAOCC_CONVERTER_CSR_21,             /* R/W */
              CCURAOCC_CONVERTER_CSR_22,             /* R/W */
              CCURAOCC_CONVERTER_CSR_23,             /* R/W */
              CCURAOCC_CONVERTER_CSR_24,             /* R/W */
              CCURAOCC_CONVERTER_CSR_25,             /* R/W */
              CCURAOCC_CONVERTER_CSR_26,             /* R/W */
              CCURAOCC_CONVERTER_CSR_27,             /* R/W */
              CCURAOCC_CONVERTER_CSR_28,             /* R/W */
              CCURAOCC_CONVERTER_CSR_29,             /* R/W */
              CCURAOCC_CONVERTER_CSR_30,             /* R/W */
              CCURAOCC_CONVERTER_CSR_31,             /* R/W */

              CCURAOCC_PLL_SYNC,                     /* R/W */

              CCURAOCC_CONVERTER_UPDATE_SELECTION,   /* R/W */
              CCURAOCC_CHANNEL_SELECT,               /* R/W */

              CCURAOCC_CALIBRATOR_BUS_CONTROL,       /* R/W */
              CCURAOCC_TEST_BUS_CONTROL,             /* R/W */
              CCURAOCC_CALIBRATOR_ADC_CONTROL,       /* R/W */

              CCURAOCC_FIFO_CSR,                     /* R/W */
              CCURAOCC_FIFO_THRESHOLD,               /* R/W */
              CCURAOCC_CALIBRATOR_ADC_DATA,          /* R only */

              CCURAOCC_FIRMWARE_SPI_COUNTER_STATUS,  /* R/W */

              CCURAOCC_CHANNEL_DATA,                 /* R/W */
              CCURAOCC_CHANNEL_DATA_0,               /* R/W */
              CCURAOCC_CHANNEL_DATA_1,               /* R/W */
```

```
        CCURAOCC_CHANNEL_DATA_2,                /* R/W */
        CCURAOCC_CHANNEL_DATA_3,                /* R/W */
        CCURAOCC_CHANNEL_DATA_4,                /* R/W */
        CCURAOCC_CHANNEL_DATA_5,                /* R/W */
        CCURAOCC_CHANNEL_DATA_6,                /* R/W */
        CCURAOCC_CHANNEL_DATA_7,                /* R/W */
        CCURAOCC_CHANNEL_DATA_8,                /* R/W */
        CCURAOCC_CHANNEL_DATA_9,                /* R/W */
        CCURAOCC_CHANNEL_DATA_10,               /* R/W */
        CCURAOCC_CHANNEL_DATA_11,               /* R/W */
        CCURAOCC_CHANNEL_DATA_12,               /* R/W */
        CCURAOCC_CHANNEL_DATA_13,               /* R/W */
        CCURAOCC_CHANNEL_DATA_14,               /* R/W */
        CCURAOCC_CHANNEL_DATA_15,               /* R/W */
        CCURAOCC_CHANNEL_DATA_16,               /* R/W */
        CCURAOCC_CHANNEL_DATA_17,               /* R/W */
        CCURAOCC_CHANNEL_DATA_18,               /* R/W */
        CCURAOCC_CHANNEL_DATA_19,               /* R/W */
        CCURAOCC_CHANNEL_DATA_20,               /* R/W */
        CCURAOCC_CHANNEL_DATA_21,               /* R/W */
        CCURAOCC_CHANNEL_DATA_22,               /* R/W */
        CCURAOCC_CHANNEL_DATA_23,               /* R/W */
        CCURAOCC_CHANNEL_DATA_24,               /* R/W */
        CCURAOCC_CHANNEL_DATA_25,               /* R/W */
        CCURAOCC_CHANNEL_DATA_26,               /* R/W */
        CCURAOCC_CHANNEL_DATA_27,               /* R/W */
        CCURAOCC_CHANNEL_DATA_28,               /* R/W */
        CCURAOCC_CHANNEL_DATA_29,               /* R/W */
        CCURAOCC_CHANNEL_DATA_30,               /* R/W */
        CCURAOCC_CHANNEL_DATA_31,               /* R/W */

        CCURAOCC_FIFO_DATA,                     /* W Only */

        CCURAOCC_PLL_0_STATUS,                  /* R Only */
        CCURAOCC_PLL_0_ACCESS,                  /* R/W */
        CCURAOCC_PLL_0_READ_1,                  /* R/W */
        CCURAOCC_PLL_0_READ_2,                  /* R/W */

        CCURAOCC_GAIN_CALIBRATION,              /* R/W */
        CCURAOCC_OFFSET_CALIBRATION,            /* R/W */

        CCURAOCC_CALIBRATOR_ADC_POSITIVE_GAIN,  /* R/W */
        CCURAOCC_CALIBRATOR_ADC_NEGATIVE_GAIN,  /* R/W */
        CCURAOCC_CALIBRATOR_ADC_OFFSET,         /* R/W */

        CCURAOCC_SPI_RAM,                       /* R/W */

    } CCURAOCC_CONTROL;
```

2.2.53 ccurAOCC_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
    ccurAOCC_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)
    *****************************************************************************/
```

2.2.54 ccurAOCC_Identify_Board()

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

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Identify_Board (void                  *Handle,
                             _ccuraocc_identify_t Identify)

    Description: Identify the board by setting the front LED

    Input:   void                   *Handle              (Handle pointer)
             _ccuraocc_identify_t Identify               (Identify board settings)
                # CCURAOCC_IDENTIFY_OFF                  (turn off flashing)
                # CCURAOCC_IDENTIFY_ON                   (turn on flashing)
                # Number of seconds to flash             (flash for number of seconds)
    Output:  none
    Return:  _ccuraocc_lib_error_number_t
                # CCURAOCC_LIB_NO_ERROR                  (successful)
                # CCURAOCC_LIB_BAD_HANDLE                (no/bad handler supplied)
                # CCURAOCC_LIB_NO_LOCAL_REGION           (local region not present)
                # CCURAOCC_LIB_NOT_OPEN                  (device not open)
                # CCURAOCC_LIB_INVALID_ARG               (invalid argument)
    *****************************************************************************/
```

2.2.55 ccurAOCC_Initialize_Board()

This call resets the board to a default initial state.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Initialize_Board (void *Handle)

    Description: Initialize the board.

    Input:       void      *Handle              (handle pointer)
    Output:      None
    Return:      _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR        (successful)
                  - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN        (device not open)
                  - CCURAOCC_LIB_IOCTL_FAILED    (driver ioctl call failed)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
    *****************************************************************************/
```

2.2.56 ccurAOCC_Initialize_PLL_Input_Struct()

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

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Initialize_PLL_Input_Struct (void                    *Handle,
                                          ccuraocc_PLL_setting_t *input)

    Description: Initialize the clock structure.

    Input:       void                    *Handle    (handle pointer)
                 ccuraocc_PLL_setting_t *input       (pointer to input clock struct)
    Output:      none
    Return:      _ccuraocc_lib_error_number_t
```

```
                             - CCURAOCC_LIB_NO_ERROR        (successful)
                             - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                             - CCURAOCC_LIB_NOT_OPEN        (device not open)
                             - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                             - CCURAOCC_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 */
        } ccuraocc_PLL_setting_t;

        - CCURAOCC_DEFAULT                     (-1)    /* Set defaults */
        - CCURAOCC_DEFAULT_REFERENCE_FREQ      (65.536)/* MHz */
        - CCURAOCC_DEFAULT_TOLERANCE           (1000)  /* ppm (parts per million) */
        - CCURAOCC_DEFAULT_MIN_ALLOWABLE_FREQ  (1.0)   /* MHz */
        - CCURAOCC_DEFAULT_VCO_GAIN            (520)   /* MHz/volts */
        - CCURAOCC_DEFAULT_MIN_VCO_FREQ        (100)   /* MHz */
        - CCURAOCC_DEFAULT_MAX_VCO_FREQ        (400)   /* MHz */
        - CCURAOCC_DEFAULT_MIN_REF_DIVIDER     (1)     /* minimum reference divider */
        - CCURAOCC_DEFAULT_MAX_REF_DIVIDER     (4095)  /* maximum reference divider */
        - CCURAOCC_DEFAULT_MIN_FEEDBK_DIVIDER  (12)    /* minimum feedback divider */
        - CCURAOCC_DEFAULT_MAX_FEEDBK_DIVIDER  (16383) /* maximum feedback divider */


        fRef              = CCURAOCC_DEFAULT_REFERENCE_FREQ;
        maximizeVCOspeed  = CCURAOCC_DEFAULT_VCO_SPEED;
        fPFDmin           = CCURAOCC_DEFAULT_MIN_ALLOWABLE_FREQ;
        max_tol           = CCURAOCC_DEFAULT_TOLERANCE;
        kfVCO             = CCURAOCC_DEFAULT_VCO_GAIN;
        fVcoMin           = CCURAOCC_DEFAULT_MIN_VCO_FREQ;
        fVcoMax           = CCURAOCC_DEFAULT_MAX_VCO_FREQ;
        nRefMin           = CCURAOCC_DEFAULT_MIN_REF_DIVIDER;
        nRefMax           = CCURAOCC_DEFAULT_MAX_REF_DIVIDER;
        nFbkMin           = CCURAOCC_DEFAULT_MIN_FEEDBK_DIVIDER;
        nFbkMax           = CCURAOCC_DEFAULT_MAX_FEEDBK_DIVIDER;
        fDesired          = CCURAOCC_DEFAULT;
```

2.2.57 ccurAOCC_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 *ccurAOCC_Munmap_Physical_Memory()* API call
to remove the previously allocated physical memory.

```
        /***************************************************************************
           _ccuraocc_lib_error_number_t
           ccurAOCC_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:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR           (successful)
                  - CCURAOCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN           (device not open)
                  - CCURAOCC_LIB_INVALID_ARG        (invalid argument)
                  - CCURAOCC_LIB_MMAP_SELECT_FAILED (mmap selection failed)
                  - CCURAOCC_LIB_MMAP_FAILED        (mmap failed)
    *************************************************************************/
```

2.2.58 ccurAOCC_Munmap_Physical_Memory()

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

```
/*************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Munmap_Physical_Memory (void *Handle)

    Description: Unmap a previously mapped physical DMA memory.

    Input:      void        *Handle              (handle pointer)
    Output:     None
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR       (successful)
                  - CCURAOCC_LIB_BAD_HANDLE     (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN       (device not open)
                  - CCURAOCC_LIB_MUNMAP_FAILED  (failed to un-map memory)
                  - CCURAOCC_LIB_NOT_MAPPED     (memory not mapped)
    *************************************************************************/
```

2.2.59 ccurAOCC_NanoDelay()

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

```
/*************************************************************************
    void
    ccurAOCC_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
    *************************************************************************/
```

2.2.60 ccurAOCC_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. A character special file */dev/ccuraocc<Board_Number>* must exist 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 *write(2)* calls which will change the default writing in block mode.

Additionally, this library provides the user with an *O_APPEND* flag. The purpose of this flag is to request the driver to open an already opened board. Though the driver allows multiple open calls to the same board with the use of this flag, it becomes the responsibility of the user to ensure that no two applications or threads are communicating with the board at the same time; otherwise, results will be unpredictable. Several tests supplied with the driver have the *O_APPEND* flag enabled. This is only for convenience during testing and debugging and is not intended for the applications to be invoked or running while the user applications are accessing the board.

```
/****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_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:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR          (successful)
                 - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
                 - CCURAOCC_LIB_ALREADY_OPEN      (device already opened)
                 - CCURAOCC_LIB_OPEN_FAILED       (device open failed)
                 - CCURAOCC_LIB_ALREADY_MAPPED    (memory already mmapped)
                 - CCURAOCC_LIB_MMAP_SELECT_FAILED (mmap selection failed)
                 - CCURAOCC_LIB_MMAP_FAILED       (mmap failed)
   ****************************************************************************/
```

2.2.61 ccurAOCC_Open_Wave()

This call is identical to the *ccurAOCC_Open()* call with the exception, that the character special file */dev/ccuraocc_wave<Board Number>* is opened and must exist for the call to be successful. One character special file is created for each board found when the driver is successfully loaded. When the driver is loaded, two character special files */dev/ccuraocc<Board Number>* and */dev/ccuraocc_wave<Board Number>* are created for each board found. Currently the optional Concurrent Real-Time Wave Generation Program *WC-DA3218-WAVE* opens the board with the */dev/ccuraocc_wave<Board Number>* naming convention. The user can edit the *ccuraocc_config* file and reload the driver in order to direct wave generation application to specific boards.

```
/****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Open_Wave (void **My_Handle,
                     int  Board_Number,
                     int  oflag)

   Description: Open a Wave device.

   Input:      void      **Handle              (handle pointer to pointer)
               int        Board_Number          (0-9 board number)
               int        oflag                 (open flags)
   Output:     None
   Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR          (successful)
                 - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
                 - CCURAOCC_LIB_ALREADY_OPEN      (device already opened)
                 - CCURAOCC_LIB_OPEN_FAILED       (device open failed)
                 - CCURAOCC_LIB_ALREADY_MAPPED    (memory already mmapped)
                 - CCURAOCC_LIB_MMAP_SELECT_FAILED (mmap selection failed)
                 - CCURAOCC_LIB_MMAP_FAILED       (mmap failed)
   ****************************************************************************/
```

2.2.62 ccurAOCC_Perform_ADC_Calibration()

This board has an on-board Analog to Digital Converter (ADC) which is used to calibrate the analog output channels. Prior to calibration the output channels this ADC needs to calibrated first. This calibration is performed using the on-board calibration voltage source. Once ADC calibration is complete, appropriate values are set in the positive gain, negative gain and offset.

```
/****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Perform_ADC_Calibration (void *Handle)

   Description: Perform ADC Calibration

   Input:      void   *Handle                (handle pointer)
   Output:     none
   Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR        (successful)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                  - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN        (device not open)
 ****************************************************************************/
```

2.2.63 ccurAOCC_Perform_Channel_Gain_Calibration()

The user can perform a gain calibration for a selected set of channels with this API. They need to make sure that the ADC has been calibrated first.

```
/****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Perform_Channel_Gain_Calibration (void                 *Handle,
                                          _ccuraocc_channel_mask_t chan_mask)

   Description: Perform Selected Channels Gain Calibration

   Input:      void                    *Handle  (handle pointer)
               _ccuraocc_channel_mask_t chan_mask (selected channel mask)
   Output:     none
   Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR        (successful)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                  - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN        (device not open)
                  - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
 ****************************************************************************/
```

2.2.64 ccurAOCC_Perform_Channel_Offset_Calibration()

The user can perform an offset calibration for a selected set of channels with this API. They need to make sure that the ADC has been calibrated first.

```
/****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Perform_Channel_Offset_Calibration (void                 *Handle,
                                          _ccuraocc_channel_mask_t chan_mask)
   Description: Perform Selected Channels Offset Calibration

   Input:      void                    *Handle  (handle pointer)
               _ccuraocc_channel_mask_t chan_mask (selected channel mask)
   Output:     none
   Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR        (successful)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                  - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
```

```
                          - CCURAOCC_LIB_NOT_OPEN          (device not open)
                          - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
         **********************************************************************/
```

2.2.65 ccurAOCC_Perform_Auto_Calibration()

This call is used to create the offset and gain values for a selected set of channels. Prior to performing channel calibration, the ADC is first calibrated to ensure accurate results. This offset and gain is then applied to each channel by the hardware when setting analog output values.

This call takes approximately two seconds to run and is normally issued after the system is rebooted and whenever the channel configuration is changed. If the board has not been calibrated after a system reboot, then voltages returned will be unpredictable.

```
   /**********************************************************************
      _ccuraocc_lib_error_number_t
      ccurAOCC_Perform_Auto_Calibration (void                      *Handle,
                                         _ccuraocc_channel_mask_t chan_mask)

      Description: Perform Auto Calibration

      Input:      void                     *Handle   (handle pointer)
                  _ccuraocc_channel_mask_t chan_mask (selected channel mask)
      Output:     none
      Return:     _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR        (successful)
                    - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN        (device not open)
                    - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                    - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                    - CCURAOCC_LIB_NO_RESOURCE     (no free PLL available)
                    - CCURAOCC_LIB_IO_ERROR        (read error)
         **********************************************************************/
```

2.2.66 ccurAOCC_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. The user can always issue the *ccurAOCC_Get_PLL_Info()* call to retrieve the current clock settings, and then edit specific options with this call. The user can also use the *CCURAOCC_DO_NOT_CHANGE* parameter for any argument value in the *ccuraocc_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 *ccuraocc_PLL_encode_t* structure.

```
   /**********************************************************************
      _ccuraocc_lib_error_number_t
      ccurAOCC_Program_PLL_Advanced (void                  *Handle,
                                     int                    Program,
                                     ccuraocc_PLL_struct_t *input,
                                     ccuraocc_PLL_encode_t *current_encoded,
                                     ccuraocc_PLL_encode_t *new_encoded)

      Description: Program PLL Access values for the specified PLL.

      Input:      void                  *Handle   (handle pointer)
                  ccuraocc_PLL_struct_t *input    (pointer to pll input struct)
      Output:     int Program                     (decide to program board)
                  ccuraocc_PLL_encode_t *current_encoded
                                                  (pointer to current encoded PLL
                  ccuraocc_PLL_encode_t *new_encoded  (pointer to new encoded PLL
      Return:     _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR          (successful)
```

```
                           - CCURAOCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                           - CCURAOCC_LIB_NOT_OPEN            (device not open)
                           - CCURAOCC_LIB_INVALID_ARG         (invalid argument)
         ************************************************************************/


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

           uint          ref_freq_divider_src;   /* CCURAOCC_REF_DIVIDER_SRC_OSCILLATOR */
                                                  /* CCURAOCC_REF_DIVIDER_SRC_PIN */

           uint          shutdown_1;             /* CCURAOCC_RUNNING */
                                                  /* CCURAOCC_SHUTDOWN */

           uint          post_divider1;          /* CCURAOCC_POST_DIVIDER1_1 */
                                                  /* CCURAOCC_POST_DIVIDER1_2 */
                                                  /* CCURAOCC_POST_DIVIDER1_3 */
                                                  /* CCURAOCC_POST_DIVIDER1_4 */
                                                  /* CCURAOCC_POST_DIVIDER1_5 */
                                                  /* CCURAOCC_POST_DIVIDER1_6 */
                                                  /* CCURAOCC_POST_DIVIDER1_7 */
                                                  /* CCURAOCC_POST_DIVIDER1_8 */
                                                  /* CCURAOCC_POST_DIVIDER1_9 */
                                                  /* CCURAOCC_POST_DIVIDER1_10*/
                                                  /* CCURAOCC_POST_DIVIDER1_11 */
                                                  /* CCURAOCC_POST_DIVIDER1_12 */

           uint          post_divider2;          /* CCURAOCC_POST_DIVIDER2_1 */
                                                  /* CCURAOCC_POST_DIVIDER2_2 */
                                                  /* CCURAOCC_POST_DIVIDER2_3 */
                                                  /* CCURAOCC_POST_DIVIDER2_4 */
                                                  /* CCURAOCC_POST_DIVIDER2_5 */
                                                  /* CCURAOCC_POST_DIVIDER2_6 */
                                                  /* CCURAOCC_POST_DIVIDER2_7 */
                                                  /* CCURAOCC_POST_DIVIDER2_8 */
                                                  /* CCURAOCC_POST_DIVIDER2_9 */
                                                  /* CCURAOCC_POST_DIVIDER2_10*/
                                                  /* CCURAOCC_POST_DIVIDER2_11 */
                                                  /* CCURAOCC_POST_DIVIDER2_12 */

           uint          post_divider3;          /* CCURAOCC_POST_DIVIDER3_1 */
                                                  /* CCURAOCC_POST_DIVIDER3_2 */
                                                  /* CCURAOCC_POST_DIVIDER3_4 */
                                                  /* CCURAOCC_POST_DIVIDER3_8 */

           uint          feedback_divider;       /* [13:00] */
           uint          feedback_divider_src;   /* CCURAOCC_FEEDBACK_DIVIDER_SRC_VCO */
                                                  /* CCURAOCC_FEEDBACK_DIVIDER_SRC_POST */

           uint          clock_output;           /* CCURAOCC_CLOCK_OUTPUT_PECL */
                                                  /* CCURAOCC_CLOCK_OUTPUT_CMOS */

           uint          charge_pump_current;    /* CCURAOCC_CHARGE_PUMP_CURRENT_2UA */
                                                  /* CCURAOCC_CHARGE_PUMP_CURRENT_4_5UA */
                                                  /* CCURAOCC_CHARGE_PUMP_CURRENT_11UA */
                                                  /* CCURAOCC_CHARGE_PUMP_CURRENT_22_5UA */

           uint          loop_resistor;          /* CCURAOCC_LOOP_RESISTOR_400K */
                                                  /* CCURAOCC_LOOP_RESISTOR_133K */
                                                  /* CCURAOCC_LOOP_RESISTOR_30K */
                                                  /* CCURAOCC_LOOP_RESISTOR_12K */

           uint          loop_capacitor;         /* CCURAOCC_LOOP_CAPACITOR_185PF */
                                                  /* CCURAOCC_LOOP_CAPACITOR_500PF */
```

```
    uint        sync_enable;                /* CCURAOCC_SYNC_DISABLE */
                                            /* CCURAOCC_SYNC_ENABLE */

    uint        sync_polarity;              /* CCURAOCC_SYNC_POLARITY_NEGATIVE */
                                            /* CCURAOCC_SYNC_POLARITY_POSITIVE */

    uint        shutdown_2;                 /* CCURAOCC_RUNNING */
                                            /* CCURAOCC_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 */
} ccuraocc_PLL_struct_t;

typedef struct {
    uint    reg[CCURAOCC_PLL_AR_REGISTER_ADDRESS_MAX];
} ccuraocc_PLL_encode_t;
```

2.2.67 ccurAOCC_Program_PLL_Clock()

This call is available for use by advanced users to program a specified clock. This *ccurAOCC_Program_PLL_Clock()* call is a higher level call than the above *ccurAOCC_Program_PLL_Advanced()* call. In this case, the user only needs to supply the desired clock frequency *(that ranges from 200 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 *CCURAOCC_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.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Program_PLL_Clock (void                  *Handle,
                                int                    Program,
                                ccuraocc_PLL_clock_t  *clock)

    Description: Program PLL Clock for give maximum tolerance

    Input:      void                    *Handle      (handle pointer)
                int Program                          (decide to program board)
                ccuraocc_PLL_clock_t   *clock        (pointer to user clock struct)
    Output:     ccuraocc_PLL_clock_t   *clock        (pointer to user clock struct)
    Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR             (successful)
                 - CCURAOCC_LIB_INVALID_ARG          (invalid argument)
                 - CCURAOCC_LIB_NO_SOLUTION_FOUND (no solution found)
                 - CCURAOCC_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 */
} ccuraocc_PLL_clock_t;
```

2.2.68 ccurAOCC_Program_Sample_Rate()

This is the basic call that is used to select a sampling rate for the board. The current range is from 0.2 SPS to 400,000 SPS. The call returns useful clock information and the actual sample rate the board was able to be programmed with.

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

```
     _ccuraocc_lib_error_number_t
     ccurAOCC_Program_Sample_Rate (void                  *Handle,
                                   double                 sample_rate,
                                   double                 *actual_sample_rate,
                                   ccuraocc_PLL_clock_t *pll_clock,
                                   uint *divider)

     Description: Program Sample Rate

     Input:      void          *Handle              (handle pointer)
                 double        sample_rate          (sample rate to program)
     Output:     double        *actual_sample_rate (pointer to actual sample rate)
                 ccuraocc_PLL_clock_t *pll_clock    (pointer to programmed
                                                     pll_clock)
                 uint          *divider             (pointer converter divider)
     Return:     _ccuraocc_lib_error_number_t
                   - CCURAOCC_LIB_NO_ERROR          (successful)
                   - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
                   - CCURAOCC_LIB_NO_LOCAL_REGION   (local region not present)
                   - CCURAOCC_LIB_NO_RESOURCE       (PLL in use)
 *******************************************************************************/

     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 */
     } ccuraocc_PLL_clock_t;
```

2.2.69 ccurAOCC_Read()

This call is provided for users to read the channels registers that were previously written to. 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.

```
/*******************************************************************************
     _ccuraocc_lib_error_number_t
     ccurAOCC_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:     _ccuraocc_lib_error_number_t
                   - CCURAOCC_LIB_NO_ERROR       (successful)
                   - CCURAOCC_LIB_BAD_HANDLE     (no/bad handler supplied)
                   - CCURAOCC_LIB_NOT_OPEN       (device not open)
                   - CCURAOCC_LIB_IO_ERROR       (read failed)
                   - CCURAOCC_LIB_FIFO_OVERFLOW  (fifo overflow)
                   - CCURAOCC_LIB_INVALID_ARG    (invalid argument)
 *******************************************************************************/
```

2.2.70 ccurAOCC_Read_Channels()

This call performs a programmed I/O read of all the selected channels and returns various channel information in the *ccuraocc_read_channels_t* structure.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Read_Channels (void                       *Handle,
                            ccuraocc_read_channels_t *rdc)

    Description: Read Channels and return channel specific information

    Input:      void                       *Handle  (handle pointer)
                ccuraocc_read_channels_t *rdc      (perform_convertion)
    Output:     ccuraocc_read_channels_t *rdc      (pointer to rdc struct)
    Return:     _ccuraocc_lib_error_number_t
                   - CCURAOCC_LIB_NO_ERROR      (successful)
                   - CCURAOCC_LIB_BAD_HANDLE    (no/bad handler supplied)
                   - CCURAOCC_LIB_NOT_OPEN      (device not open)
                   - CCURAOCC_LIB_INVALID_ARG   (invalid argument)
*****************************************************************************/

typedef struct
{
    char select_channel;
    union
    {
        char convert_rawdata_to_volts;  /* for reading from channel registers */
        char convert_volts_to_rawdata;  /* for writing to channel registers */
    };
    char channel_synchronized_update_flag;
    char converter_data_format;
    char converter_output_range;
    int channel_data_raw;
    double channel_data_volts;
} ccuraocc_single_channel_data_t;

typedef struct
{
    ccuraocc_single_channel_data_t rchan[CCURAOCC_MAX_CHANNELS];
} ccuraocc_read_channels_t;
```

The user needs to set the *select_channel* and the *convert_rawdata_to_volts* fields in the *ccuraocc_single_channel_data_t* structure for information on each channel they need to acquire. To select a channel, the *select_channel* field needs to be set to *CCURAOCC_TRUE*. If the *convert_rawdata_to_volts* field is set to *CCURAOCC_TRUE,* the call will also convert the raw data read from the registers to voltages by applying the correct data format and voltage range.

2.2.71 ccurAOCC_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 *ccurAOCC_Write_Channels_Calibration()* routine. Any blank lines or entries starting with '#' or '*' are ignored during parsing.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Read_Channels_Calibration (void *Handle,
                                        char *filename)

    Description: Read Channels Calibration information
```

```
    Input:          void                    *Handle     (handle pointer)
    Output:         char                    *filename   (pointer to filename)
    Return:         _ccuraocc_lib_error_number_t
                      - CCURAOCC_LIB_NO_ERROR         (successful)
                      - CCURAOCC_LIB_BAD_HANDLE       (no/bad handler supplied)
                      - CCURAOCC_LIB_NOT_OPEN         (device not open)
                      - CCURAOCC_LIB_NO_LOCAL_REGION  (local region not present)
                      - CCURAOCC_LIB_CANNOT_OPEN_FILE (file not readable)
    **************************************************************************/
```

Format:

```
#Chan   Offset              Gain
#====   ======              ====
 ch00:  0.1983642578125000  0.3991699218750000
 ch01:  0.0860595703125000  0.2078247070312500
 ch02:  0.1992797851562500  0.4129028320312500
 ch03:  0.0830078125000000  0.1345825195312500
----
 ch28:  0.1766967773437500  0.3732299804687500
 ch29:  0.1361083984375000  0.2694702148437500
 ch30:  0.1257324218750000  0.2728271484375000
 ch31:  0.0469970703125000  0.0830078125000000
```

2.2.72 ccurAOCC_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.

```
/**************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Read_Serial_Prom(void               *Handle,
                       ccuraocc_sprom_rw_t *spr)

   Description: Read Serial Prom for specified number of words

   Input:          void                    *Handle     (handle pointer)
                   ccuraocc_sprom_rw_t    *spr         (pointer to struct)
                     -- u_short word_offset
                     -- u_short num_words
   Output:         ccuraocc_sprom_rw_t    *spr         (pointer to struct)
                     -- u_short *data_ptr
   Return:         _ccuraocc_lib_error_number_t
                      - CCURAOCC_LIB_NO_ERROR          (successful)
                      - CCURAOCC_LIB_NO_LOCAL_REGION   (error)
                      - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
                      - CCURAOCC_LIB_SERIAL_PROM_BUSY  (serial prom busy)
                      - CCURAOCC_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 */
} ccuraocc_sprom_rw_t;
```

2.2.73 ccurAOCC_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.

```
/****************************************************************************
   _ccuraocc_lib_error_number_t
    ccurAOCC_Read_Serial_Prom_Item(void                        *Handle,
                                  _ccuraocc_sprom_access_t item,
                                  void                    *item_ptr)

   Description: Read Serial Prom for specified item

   Input:        void                    *Handle   (handle pointer)
                 _ccuraocc_sprom_access_t item     (select item)
                   -- CCURAOCC_SPROM_HEADER
                   -- CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V
                   -- CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V
                   -- CCURAOCC_SPROM_FACTORY_BIPOLAR_5V
                   -- CCURAOCC_SPROM_FACTORY_BIPOLAR_10V
                   -- CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V
                   -- CCURAOCC_SPROM_USER_CHECKPOINT_1
                   -- CCURAOCC_SPROM_USER_CHECKPOINT_2
   Output:       void                    *item_ptr (pointer to item struct)
                   -- *ccuraocc_sprom_header_t
                   -- *ccuraocc_sprom_factory_t
                   -- *ccuraocc_sprom_user_checkpoint_t
   Return:       _ccuraocc_lib_error_number_t
                   - CCURAOCC_LIB_NO_ERROR            (successful)
                   - CCURAOCC_LIB_NO_LOCAL_REGION     (error)
                   - CCURAOCC_LIB_INVALID_ARG         (invalid argument)
                   - CCURAOCC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
                   - CCURAOCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
 ****************************************************************************/

typedef enum {
    CCURAOCC_SPROM_HEADER=1,
    CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V,
    CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V,
    CCURAOCC_SPROM_FACTORY_BIPOLAR_5V,
    CCURAOCC_SPROM_FACTORY_BIPOLAR_10V,
    CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V,
    CCURAOCC_SPROM_USER_CHECKPOINT_1,
    CCURAOCC_SPROM_USER_CHECKPOINT_2,
} _ccuraocc_sprom_access_t;
```

The *void* pointer **item_ptr* points to one of the following structures depending on the selected *item* that needs to be returned.

```
typedef struct {
    u_int   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 */
} ccuraocc_sprom_header_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 32/64 m/c*/ /* 0x008 - 0x00F - date */
    };
    u_short offset[CCURAOCC_MAX_CHANNELS];   /* 0x010 - 0x04F - offset */
    u_short gain[CCURAOCC_MAX_CHANNELS];     /* 0x050 - 0x08F - gain */
} ccuraocc_sprom_factory_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 32/64 m/c*/ /* 0x008 - 0x00F - date */
        };
        u_short offset[CCURAOCC_MAX_CHANNELS];    /* 0x010 - 0x04F - offset */
        u_short gain[CCURAOCC_MAX_CHANNELS];      /* 0x050 - 0x08F - gain */
        u_int   converter_csr[CCURAOCC_MAX_CONVERTERS];
                                                  /* 0x090 - 0x10F - channel config */
} ccuraocc_sprom_user_checkpoint_t;
```

2.2.74 ccurAOCC_Read_Single_Channel()

This call is similar to the *ccurAOCC_Read_Channels()*, except, information is returned for a single channel. Once again useful information on the selected channel is provided to the user.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Read_Single_Channel (void                        *Handle,
                                  int                         chan,
                                  ccuraocc_single_channel_data_t *rdc)

    Description: Read Single Channel

    Input:      void                        *Handle (handle pointer)
                int                         chan    (channel to read)
                ccuraocc_single_channel_data_t *rdc  (perform_convertion)
    Output:     ccuraocc_single_channel_data_t *rdc  (pointer to rdc struct)
    Return:     _ccuraocc_lib_error_number_t
                - CCURAOCC_LIB_NO_ERROR              (successful)
                - CCURAOCC_LIB_BAD_HANDLE            (no/bad handler supplied)
                - CCURAOCC_LIB_NOT_OPEN              (device not open)
                - CCURAOCC_LIB_INVALID_ARG           (invalid argument)
 *****************************************************************************/

typedef struct
{
    char select_channel;
    union
    {
        char convert_rawdata_to_volts;  /* for reading from channel registers */
        char convert_volts_to_rawdata;  /* for writing to channel registers */
    };
    char channel_synchronized_update_flag;
    char converter_data_format;
    char converter_output_range;
    int channel_data_raw;
    double channel_data_volts;
} ccuraocc_single_channel_data_t;
```

The user needs to set the channel number in *chan* and the *convert_rawdata_to_volts* field in the *ccuraocc_single_channel_data_t* structure for information on the channel they need to acquire. The *select_channel* field is ignored. If the *convert_rawdata_to_volts* field is set to *CCURAOCC_TRUE,* the call will also convert the raw data read from the registers to voltages by applying the correct data format and voltage range.

2.2.75 ccurAOCC_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.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_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:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR     (successful)
                  - CCURAOCC_LIB_BAD_HANDLE   (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN     (device not open)
                  - CCURAOCC_LIB_IOCTL_FAILED (driver ioctl call failed)
 ****************************************************************************/
```

2.2.76 ccurAOCC_Reset_ADC_Calibrator()

This call performs a reset of the offset, positive gain and negative gain registers default state. Basically, at this point, the Calibrator will be un-calibrated.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Reset_ADC_Calibrator (void *Handle)

    Description: Reset ADC Calibrator

    Input:      void *Handle                 (handle pointer)
    Output:     None
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR       (successful)
                  - CCURAOCC_LIB_BAD_HANDLE     (no/bad handler supplied)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
 ****************************************************************************/
```

2.2.77 ccurAOCC_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.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Reset_Board (void *Handle)

    Description: Reset the board.

    Input:      void *Handle                 (handle pointer)
    Output:     None
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR     (successful)
                  - CCURAOCC_LIB_BAD_HANDLE   (no/bad handler supplied)
```

```
                    - CCURAOCC_LIB_NOT_OPEN        (device not open)
                    - CCURAOCC_LIB_IOCTL_FAILED    (driver ioctl call failed)
                    - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
     ********************************************************************************/
```

2.2.78 ccurAOCC_Reset_Channel_Calibration()

This call resets the offset and gain registers for the selected channels.

```
     /********************************************************************************
        _ccuraocc_lib_error_number_t
        ccurAOCC_Reset_Channel_Calibration (void                    *Handle,
                                       _ccuraocc_channel_mask_t chan_mask)

        Description: Reset Selected Channel Calibration

        Input:      void                    *Handle   (handle pointer)
                    _ccuraocc_channel_mask_t   chan_mask (selected channel mask)
        Output:     None
        Return:     _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR          (successful)
                    - CCURAOCC_LIB_BAD_HANDLE        (no/bad handler supplied)
                    - CCURAOCC_LIB_NO_LOCAL_REGION   (local region not present)
                    - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
     ********************************************************************************/
```

2.2.79 ccurAOCC_Reset_Fifo()

This call performs a FIFO reset. All data held in the FIFO is cleared and the FIFO is rendered empty.

```
     /********************************************************************************
        _ccuraocc_lib_error_number_t
        ccurAOCC_Reset_Fifo (void *Handle)

        Description: Reset Fifo

        Input:      void *Handle                    (handle pointer)
        Output:     none
        Return:     _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR          (successful)
                    - CCURAOCC_LIB_BAD_HANDLE        (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN          (device not open)
                    - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
     ********************************************************************************/
```

2.2.80 ccurAOCC_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 channel voltage range 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.

```
     /********************************************************************************
        _ccuraocc_lib_error_number_t
        ccurAOCC_Restore_Factory_Calibration (void *Handle)

        Description: Restore Factory board calibration from serial prom

        Input:      void *Handle                    (handle pointer)
        Output:     none
        Return:     _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR          (successful)
                    - CCURAOCC_LIB_BAD_HANDLE        (no/bad handler supplied)
```

```
                      - CCURAOCC_LIB_NOT_OPEN         (device not open)
                      - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                      - CCURAOCC_LIB_INVALID_CRC     (invalid CRC)
      *************************************************************************/
```

2.2.81 ccurAOCC_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 channel configuration 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.

```
      /*************************************************************************
         _ccuraocc_lib_error_number_t
         ccurAOCC_Restore_User_Checkpoint(void                    *Handle,
                                          _ccuraocc_sprom_access_t item)

         Description: Restore User Checkpoint from serial prom

         Input:       void                      *Handle    (handle pointer)
                      _ccuraocc_sprom_access_t item        (select item)
                        -- CCURAOCC_SPROM_USER_CHECKPOINT_1
                        -- CCURAOCC_SPROM_USER_CHECKPOINT_2
         Output:      none
         Return:      _ccuraocc_lib_error_number_t
                      - CCURAOCC_LIB_NO_ERROR           (successful)
                      - CCURAOCC_LIB_NO_LOCAL_REGION    (error)
                      - CCURAOCC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
                      - CCURAOCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
                      - CCURAOCC_LIB_INVALID_CRC        (invalid CRC)
                      - CCURAOCC_LIB_INVALID_ARG        (invalid argument)
      *************************************************************************/

      typedef enum {
         CCURAOCC_SPROM_HEADER=1,
         CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V,
         CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V,
         CCURAOCC_SPROM_FACTORY_BIPOLAR_5V,
         CCURAOCC_SPROM_FACTORY_BIPOLAR_10V,
         CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V,
         CCURAOCC_SPROM_USER_CHECKPOINT_1,
         CCURAOCC_SPROM_USER_CHECKPOINT_2,
      } _ccuraocc_sprom_access_t;
```

2.2.82 ccurAOCC_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.

```
      /*************************************************************************
         _ccuraocc_lib_error_number_t
         ccurAOCC_Select_Driver_Read_Mode (void                     *Handle,
                                           _ccuraocc_driver_rw_mode_t mode)

         Description: Select Driver Read Mode

         Input:       void                      *Handle (handle pointer)
                      _ccuraocc_driver_rw_mode_t mode    (select read mode)
         Output:      none
         Return:      _ccuraocc_lib_error_number_t
                      - CCURAOCC_LIB_NO_ERROR           (successful)
```

```
                          - CCURAOCC_LIB_BAD_HANDLE        (no/bad handler supplied)
                          - CCURAOCC_LIB_NOT_OPEN          (device not open)
                          - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
                          - CCURAOCC_LIB_NO_LOCAL_REGION   (local region not present)
    ***********************************************************************/

    typedef enum
    {
        CCURAOCC_PIO_CHANNEL,        /* read/write mode */
        CCURAOCC_DMA_CHANNEL,        /* read/write mode */
        CCURAOCC_PIO_FIFO,           /* write mode */
        CCURAOCC_DMA_FIFO,           /* write mode */
    } _ccuraocc_driver_rw_mode_t;
```

2.2.83 ccurAOCC_Select_Driver_Write_Mode()

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

```
    /******************************************************************************
        _ccuraocc_lib_error_number_t
        ccurAOCC_Select_Driver_Write_Mode (void                          *Handle,
                                        _ccuraocc_driver_rw_mode_t mode)

        Description: Select Driver Write Mode

        Input:      void                      *Handle (handle pointer)
                    _ccuraocc_driver_rw_mode_t mode    (select write mode)
        Output:     none
        Return:     _ccuraocc_lib_error_number_t
                          - CCURAOCC_LIB_NO_ERROR         (successful)
                          - CCURAOCC_LIB_BAD_HANDLE       (no/bad handler supplied)
                          - CCURAOCC_LIB_NOT_OPEN         (device not open)
                          - CCURAOCC_LIB_INVALID_ARG      (invalid argument)
                          - CCURAOCC_LIB_NO_LOCAL_REGION  (local region not present)
    ***********************************************************************/

    typedef enum
    {
        CCURAOCC_PIO_CHANNEL,        /* read/write mode */
        CCURAOCC_DMA_CHANNEL,        /* read/write mode */
        CCURAOCC_PIO_FIFO,           /* write mode */
        CCURAOCC_DMA_FIFO,           /* write mode */
    } _ccuraocc_driver_rw_mode_t;
```

2.2.84 ccurAOCC_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:

       - ccurAOCC_Create_Factory_Calibration()
       - ccurAOCC_Create_User_Checkpoint()
       - ccurAOCC_Write_Serial_Prom()
       - ccurAOCC_Write_Serial_Prom_Item()

```
    /******************************************************************
        _ccuraocc_lib_error_number_t
```

```
        ccurAOCC_Serial_Prom_Write_Override (void *Handle,
                                             int  action)

        Description: Set Serial Prom Write Override

        Input:       void            *Handle          (handle pointer)
                     int              action          (override action)
                       -- CCURAOCC_TRUE
                       -- CCURAOCC_FALSE
        Output:      none
        Return:      _ccuraocc_lib_error_number_t
                       - CCURAOCC_LIB_NO_ERROR       (successful)
                       - CCURAOCC_LIB_BAD_HANDLE     (no/bad handler supplied)
                       - CCURAOCC_LIB_NOT_OPEN       (device not open)
                       - CCURAOCC_LIB_INVALID_ARG    (invalid argument)
                       - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
     *************************************************************************/
```

When *action* is set to *CCURAOCC_TRUE*, the serial prom write protecting is disabled, otherwise, it is enabled.

2.2.85 ccurAOCC_Set_Board_CSR()

This call is used to activate or reset the channel converters and to select an output clock that is fed to another card. Until the board converters are active, no data can be written to the channel registers.

```
     /*************************************************************************
        _ccuraocc_lib_error_number_t
        ccurAOCC_Set_Board_CSR (void                   *Handle,
                          ccuraocc_board_csr_t *bcsr)

        Description: Set Board Control and Status information

        Input:       void *Handle                  (handle pointer)
                     ccuraocc_board_csr_t   *bcsr   (pointer to board csr)
        Output:      none
        Return:      _ccuraocc_lib_error_number_t
                       - CCURAOCC_LIB_NO_ERROR       (successful)
                       - CCURAOCC_LIB_BAD_HANDLE     (no/bad handler supplied)
                       - CCURAOCC_LIB_NOT_OPEN       (device not open)
                       - CCURAOCC_LIB_INVALID_ARG    (invalid argument)
                       - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
     *************************************************************************/

     typedef struct
     {
         int external_clock_detected;       /* external clock detected */
         int all_converter_reset;           /* all converter reset */
         int external_clock_output;         /* external clock selection */
         int identify_board;                /* identify board */
     } ccuraocc_board_csr_t;
```

// all_converter_reset
- CCURAOCC_BCSR_ALL_CONVERTER_ACTIVE
- CCURAOCC_BCSR_ALL_CONVERTER_RESET
- CCURAOCC_DO_NOT_CHANGE

// external_clock_output
- CCURAOCC_BCSR_EXTCLK_OUTPUT_SOFTWARE_FLAG:
- CCURAOCC_BCSR_EXTCLK_OUTPUT_PLL_CLOCK:
- CCURAOCC_BCSR_EXTCLK_OUTPUT_EXTERNAL_CLOCK:
- CCURAOCC_DO_NOT_CHANGE:

// identify_board
- CCURAOCC_BCSR_IDENTIFY_BOARD_DISABLE
- CCURAOCC_BCSR_IDENTIFY_BOARD_ENABLE
- CCURAOCC_DO_NOT_CHANGE:

2.2.86 ccurAOCC_Set_Calibrator_ADC_Control()

The board has an on-board Analog to Digital Converter (ADC) that is used during calibration of the channels. This call returns the ADC control and range information. Normally, the user does not need this API. It is used internally by the API to calibrate the channels.

```
/*****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Set_Calibrator_ADC_Control (void                    *Handle,
                                 _ccuraocc_calib_adc_control_t adc_control)

   Description: Set Calibrator ADC Control Information

   Input:      void                          *Handle    (handle pointer)
               _ccuraocc_calib_adc_control_t  adc_control (ADC control)
   Output:     none
   Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR              (successful)
                 - CCURAOCC_LIB_NO_LOCAL_REGION       (local region error)
                 - CCURAOCC_LIB_BAD_HANDLE            (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN              (device not open)
                 - CCURAOCC_LIB_INVALID_ARG           (invalid argument)
 *****************************************************************************/


typedef enum
{
   CCURAOCC_CALADC_CONTROL_BIPOLAR_0_5V   = (0), /*  0V  to  +5V (10V p-p) */
   CCURAOCC_CALADC_CONTROL_BIPOLAR_0_10V  = (1), /*  0V  to +10V (20V p-p) */
   CCURAOCC_CALADC_CONTROL_BIPOLAR_5_5V   = (2), /* -5V  to  +5V (20V p-p) */
   CCURAOCC_CALADC_CONTROL_BIPOLAR_10_10V = (3), /* -10V to +10V (40V p-p) */
} _ccuraocc_calib_adc_control_t;
```

2.2.87 ccurAOCC_Set_Calibrator_ADC_NegativeGainCal()

The call converts the user supplied floating point value *Float* to raw value and writes it to the ADC Negative Gain Calibration register.

```
/*****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Set_Calibrator_ADC_NegativeGainCal (void   *Handle,
                                         double Float)

   Description: Set Calibrator ADC Negative Gain Data

   Input:      void      *Handle            (handle pointer)
               double     Float             (Float ADC Cal)
   Output:     none
   Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR        (successful)
                 - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                 - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN        (device not open)
 *****************************************************************************/
```

2.2.88 ccurAOCC_Set_Calibrator_ADC_OffsetCal()

The call converts the user supplied floating point value *Float* to raw value and writes it to the ADC Offset Calibration register.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Calibrator_ADC_OffsetCal (void   *Handle,
                                           double Float)

    Description: Set Calibrator ADC Offset Data

    Input:      void        *Handle             (handle pointer)
                double      Float               (Float ADC Cal)
    Output:     none
    Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR        (successful)
                 - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                 - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN        (device not open)
 *****************************************************************************/
```

2.2.89 ccurAOCC_Set_Calibrator_ADC_PositiveGainCal()

The call converts the user supplied floating point value *Float* to raw value and writes it to the ADC Positive Gain Calibration register.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Calibrator_ADC_PositiveGainCal (void   *Handle,
                                                 double Float)

    Description: Set Calibrator ADC Positive Gain Data

    Input:      void        *Handle             (handle pointer)
                double      Float               (Float ADC Cal)
    Output:     none
    Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR        (successful)
                 - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                 - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN        (device not open)
 *****************************************************************************/
```

2.2.90 ccurAOCC_Set_Calibrator_Bus_Control()

The ADC *(calibrator)* can only return information for one element at a time. Prior to reading the ADC data, the user needs to select the element whose information is to be returned. This call provides the ability to connect one of the following elements to the ADC in order to return its value.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Calibrator_Bus_Control (void                     *Handle,
                                 _ccuraocc_calib_bus_control_t adc_bus_control)

    Description: Set Calibration Bus Control Information

    Input:      void                          *Handle        (handle pointer)
                _ccuraocc_calib_bus_control_t adc_bus_control (cal Bus control)
    Output:     none
    Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR               (successful)
                 - CCURAOCC_LIB_NO_LOCAL_REGION        (local region error)
```

```
                       - CCURAOCC_LIB_BAD_HANDLE             (no/bad handler supplied)
                       - CCURAOCC_LIB_NOT_OPEN               (device not open)
                       - CCURAOCC_LIB_INVALID_ARG            (invalid argument)
          **************************************************************************/
```

```
        typedef enum
        {
            CCURAOCC_CALBUS_CONTROL_GROUND       = (0),
            CCURAOCC_CALBUS_CONTROL_POSITIVE_REF = (1),
            CCURAOCC_CALBUS_CONTROL_NEGATIVE_REF = (2),
            CCURAOCC_CALBUS_CONTROL_OPEN         = (3),

            CCURAOCC_CALBUS_CONTROL_CHAN_0       = (0x20),  // 8 & 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_1       = (0x21),  // 8 & 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_2       = (0x22),  // 8 & 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_3       = (0x23),  // 8 & 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_4       = (0x24),  // 8 & 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_5       = (0x25),  // 8 & 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_6       = (0x26),  // 8 & 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_7       = (0x27),  // 8 & 32-channel card

            CCURAOCC_CALBUS_CONTROL_CHAN_8       = (0x28),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_9       = (0x29),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_10      = (0x2A),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_11      = (0x2B),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_12      = (0x2C),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_13      = (0x2D),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_14      = (0x2E),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_15      = (0x2F),  // 32-channel card

            CCURAOCC_CALBUS_CONTROL_CHAN_16      = (0x30),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_17      = (0x31),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_18      = (0x32),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_19      = (0x33),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_20      = (0x34),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_21      = (0x35),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_22      = (0x36),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_23      = (0x37),  // 32-channel card

            CCURAOCC_CALBUS_CONTROL_CHAN_24      = (0x38),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_25      = (0x39),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_26      = (0x3A),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_27      = (0x3B),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_28      = (0x3C),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_29      = (0x3D),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_30      = (0x3E),  // 32-channel card
            CCURAOCC_CALBUS_CONTROL_CHAN_31      = (0x3F),  // 32-channel card

        } _ccuraocc_calib_bus_control_t;
```

2.2.91 ccurAOCC_Set_Calibration_ChannelGain()

This single call can be used to set a user supplied floating point *gain. Float* value for a selected set of channel calibration registers. The call returns the raw value written to the register in *gain.Raw*.

```
        /**************************************************************************
          _ccuraocc_lib_error_number_t
          ccurAOCC_Set_Calibration_ChannelGain (void                    *Handle,
                                                _ccuraocc_channel_mask_t chan_mask,
                                                ccuraocc_converter_cal_t *gain)

          Description: Set Calibration Channel Gain
```

```
      Input:        void                    *Handle   (handle pointer)
                    _ccuraocc_channel_mask_t  chan_mask (selected channel mask)
                    ccuraocc_converter_cal_t  *gain    (Float gain value)
      Output:       ccuraocc_converter_cal_t  *gain    (Raw gain value)
      Return:       _ccuraocc_lib_error_number_t
                      - CCURAOCC_LIB_NO_ERROR          (successful)
                      - CCURAOCC_LIB_NO_LOCAL_REGION   (local region not present)
                      - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
    **************************************************************************/


    typedef enum
    {
        CCURAOCC_CHANNEL_MASK_0  = 0x00000001,     /* chan 0 : 8 & 32-channel card */
        CCURAOCC_CHANNEL_MASK_1  = 0x00000002,     /* chan 1 : 8 & 32-channel card */
        CCURAOCC_CHANNEL_MASK_2  = 0x00000004,     /* chan 2 : 8 & 32-channel card */
        CCURAOCC_CHANNEL_MASK_3  = 0x00000008,     /* chan 3 : 8 & 32-channel card */
        CCURAOCC_CHANNEL_MASK_4  = 0x00000010,     /* chan 4 : 8 & 32-channel card */
        CCURAOCC_CHANNEL_MASK_5  = 0x00000020,     /* chan 5 : 8 & 32-channel card */
        CCURAOCC_CHANNEL_MASK_6  = 0x00000040,     /* chan 6 : 8 & 32-channel card */
        CCURAOCC_CHANNEL_MASK_7  = 0x00000080,     /* chan 7 : 8 & 32-channel card */

        CCURAOCC_CHANNEL_MASK_8  = 0x00000100,     /* chan 8 : 32-channel card */
        CCURAOCC_CHANNEL_MASK_9  = 0x00000200,     /* chan 9 : 32-channel card */
        CCURAOCC_CHANNEL_MASK_10 = 0x00000400,     /* chan 0 : 32-channel card */
        CCURAOCC_CHANNEL_MASK_11 = 0x00000800,     /* chan 11: 32-channel card */
        CCURAOCC_CHANNEL_MASK_12 = 0x00001000,     /* chan 12: 32-channel card */
        CCURAOCC_CHANNEL_MASK_13 = 0x00002000,     /* chan 13: 32-channel card */
        CCURAOCC_CHANNEL_MASK_14 = 0x00004000,     /* chan 14: 32-channel card */
        CCURAOCC_CHANNEL_MASK_15 = 0x00008000,     /* chan 15: 32-channel card */

        CCURAOCC_CHANNEL_MASK_16 = 0x00010000,     /* chan 16: 32-channel card */
        CCURAOCC_CHANNEL_MASK_17 = 0x00020000,     /* chan 17: 32-channel card */
        CCURAOCC_CHANNEL_MASK_18 = 0x00040000,     /* chan 18: 32-channel card */
        CCURAOCC_CHANNEL_MASK_19 = 0x00080000,     /* chan 19: 32-channel card */
        CCURAOCC_CHANNEL_MASK_20 = 0x00100000,     /* chan 20: 32-channel card */
        CCURAOCC_CHANNEL_MASK_21 = 0x00200000,     /* chan 21: 32-channel card */
        CCURAOCC_CHANNEL_MASK_22 = 0x00400000,     /* chan 22: 32-channel card */
        CCURAOCC_CHANNEL_MASK_23 = 0x00800000,     /* chan 23: 32-channel card */

        CCURAOCC_CHANNEL_MASK_24 = 0x01000000,     /* chan 24: 32-channel card */
        CCURAOCC_CHANNEL_MASK_25 = 0x02000000,     /* chan 25: 32-channel card */
        CCURAOCC_CHANNEL_MASK_26 = 0x04000000,     /* chan 26: 32-channel card */
        CCURAOCC_CHANNEL_MASK_37 = 0x08000000,     /* chan 27: 32-channel card */
        CCURAOCC_CHANNEL_MASK_28 = 0x10000000,     /* chan 28: 32-channel card */
        CCURAOCC_CHANNEL_MASK_29 = 0x20000000,     /* chan 29: 32-channel card */
        CCURAOCC_CHANNEL_MASK_30 = 0x40000000,     /* chan 30: 32-channel card */
        CCURAOCC_CHANNEL_MASK_31 = 0x80000000,     /* chan 31: 32-channel card */

        /* End Channel */
        CCURAOCC_ALL_CHANNEL_MASK = 0xFFFFFFFF,
    } _ccuraocc_channel_mask_t;

    typedef struct
    {
        uint Raw[CCURAOCC_MAX_CHANNELS];
        double Float[CCURAOCC_MAX_CHANNELS];
    } ccuraocc_converter_cal_t;
```

2.2.92 ccurAOCC_Set_Calibration_ChannelOffset()

This single call can be used to set a user supplied floating point *offset*. *Float* value for a selected set of channel calibration registers. The call returns the raw value written to the register in *offset.Raw.*

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Calibration_ChannelOffset (void                    *Handle,
                                             _ccuraocc_channel_mask_t chan_mask,
                                             ccuraocc_converter_cal_t *offset)

    Description: Set Calibration Channel Offset

    Input:      void                     *Handle   (handle pointer)
                _ccuraocc_channel_mask_t  chan_mask (selected channel mask)
                ccuraocc_converter_cal_t  *offset   (Float offset value)
    Output:     ccuraocc_converter_cal_t  *offset   (Raw offset value)
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR          (successful)
                  - CCURAOCC_LIB_NO_LOCAL_REGION   (local region not present)
                  - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
 ****************************************************************************/
```

Information on structures are described in the above API *ccurAOCC_Set_Calibration_ChannelGain()*.

2.2.93 ccurAOCC_Set_Channel_Selection()

This API is only applicable when performing FIFO write() operations. With this API, the user can select the specific channels that are going to be placed in the FIFO. For proper synchronization with the hardware, the user needs to ensure that the FIFO is empty before placing the first sample in the FIFO. The first sample represents the lowest channel number data. The next data in the FIFO belongs to the next higher channel number in the *channel selection* mask, respectively, until all samples for all channels in the channel selection mask are placed in the FIFO. The process is then repeated for the first channel. If at any point, an under-run is detected, the user will need to ensure that the FIFO is empty before placing new samples in the FIFO in order to be once again synchronized with the hardware.

It is not advisable to change the channel selection when there are samples in the FIFO that are destined to go to the output, as the change will take effect immediately and data destined for a specific channel could end up on another channel.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Channel_Selection (void *Handle,
                                     uint channel_select)

    Description: Set Channel_Selection

    Input:      void             *Handle         (handle pointer)
                uint              channel_select (channel selection mask)
    Output:     none
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR        (successful)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                  - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
 ****************************************************************************/
```

Information on structure is described in the above API *ccurAOCC_Get_Calibration_ChannelGain()*.

2.2.94 ccurAOCC_Set_Converter_Clock_Divider()

This API sets the clock divider register. This divider is applied to the board PLL clock to determine the sample rate. A value of '0' or '1' does not change the sample rate.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Converter_Clock_Divider (void *Handle,
                                           uint divider)
```

```
       Description: Set Converter Clock Divider

       Input:      void                    *Handle   (handle pointer)
                   uint                    divider   (clock divider)
       Output:     none
       Return:     _ccuraocc_lib_error_number_t
                     - CCURAOCC_LIB_NO_ERROR        (successful)
                     - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                     - CCURAOCC_LIB_NOT_OPEN        (device not open)
                     - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                     - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
       ************************************************************************/
```

// divider range
- CCURAOCC_CONVERTER_CLOCK_DIVIDER_MIN
- CCURAOCC_CONVERTER_CLOCK_DIVIDER_MAX

2.2.95 ccurAOCC_Set_Converter_CSR()

This sets the control information for the selected converters. The converter cannot be written too while the *CCURAOCC_CONVERTER_BUSY* flag is set in the *converter_interface_busy* field. When a converter is set for *CCURAOCC_CONVERTER_MODE_IMMEDIATE* mode, data written for that channel is output immediately, whether it is written to the channel registers or the FIFO. If the converters are in *CCURAOCC_CONVERTER_MODE_SYNCHRONIZED* mode, no data is written to any channels until at least one channel has its channel data registers synchronized update flag set as well.

Normal operation is for users to set the converter configuration for all channels prior to starting the output transfer. Data is always present in the channel registers, however, the output to the lines only takes place when a physical write to the registers occur. If data was written to the output registers with one channel configuration, the physical output lines would reflect that voltage. Now, if the user decides to change the converter configuration, e.g. the voltage range to a different value, the outputs will not reflect the change until the next data is written to the channel registers. This is also true for FIFO transfers. If the boards is actively sending out data at a given channel configuration, changing the channel configuration will not have any effect on the sample that is already out, however, the next sample going out to the line will reflect the changed configuration.

```
       /************************************************************************
       _ccuraocc_lib_error_number_t
       ccurAOCC_Set_Converter_CSR (void                    *Handle,
                                   _ccuraocc_converter_mask_t conv_mask,
                                   ccuraocc_converter_csr_t   ccsr)

       Description: Set Converter Control and Status information

       Input:      void                    *Handle   (handle pointer)
                   _ccuraocc_converter_mask_t conv_mask (selected converter)
                   ccuraocc_converter_csr_t   ccsr      (converter csr)
       Output:     none
       Return:     _ccuraocc_lib_error_number_t
                     - CCURAOCC_LIB_NO_ERROR        (successful)
                     - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                     - CCURAOCC_LIB_NOT_OPEN        (device not open)
                     - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                     - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
       ************************************************************************/

       typedef enum
       {
           CCURAOCC_CONVERTER_MASK_0  = 0x00000001,   /* chan 0 : 8 & 32-channel card */
           CCURAOCC_CONVERTER_MASK_1  = 0x00000002,   /* chan 1 : 8 & 32-channel card */
           CCURAOCC_CONVERTER_MASK_2  = 0x00000004,   /* chan 2 : 8 & 32-channel card */
```

```
        CCURAOCC_CONVERTER_MASK_3  = 0x00000008,    /* chan 3 : 8 & 32-channel card */
        CCURAOCC_CONVERTER_MASK_4  = 0x00000010,    /* chan 4 : 8 & 32-channel card */
        CCURAOCC_CONVERTER_MASK_5  = 0x00000020,    /* chan 5 : 8 & 32-channel card */
        CCURAOCC_CONVERTER_MASK_6  = 0x00000040,    /* chan 6 : 8 & 32-channel card */
        CCURAOCC_CONVERTER_MASK_7  = 0x00000080,    /* chan 7 : 8 & 32-channel card */

        CCURAOCC_CONVERTER_MASK_8  = 0x00000100,    /* chan 8 : 32-channel card */
        CCURAOCC_CONVERTER_MASK_9  = 0x00000200,    /* chan 9 : 32-channel card */
        CCURAOCC_CONVERTER_MASK_10 = 0x00000400,    /* chan 10: 32-channel card */
        CCURAOCC_CONVERTER_MASK_11 = 0x00000800,    /* chan 11: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_12 = 0x00001000,    /* chan 12: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_13 = 0x00002000,    /* chan 13: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_14 = 0x00004000,    /* chan 14: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_15 = 0x00008000,    /* chan 15: 32-channel card  */

        CCURAOCC_CONVERTER_MASK_16 = 0x00010000,    /* chan 16: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_17 = 0x00020000,    /* chan 17: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_18 = 0x00040000,    /* chan 18: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_19 = 0x00080000,    /* chan 19: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_20 = 0x00100000,    /* chan 20: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_21 = 0x00200000,    /* chan 21: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_22 = 0x00400000,    /* chan 22: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_23 = 0x00800000,    /* chan 23: 32-channel card  */

        CCURAOCC_CONVERTER_MASK_24 = 0x01000000,    /* chan 24: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_25 = 0x02000000,    /* chan 25: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_26 = 0x04000000,    /* chan 26: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_37 = 0x08000000,    /* chan 27: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_28 = 0x10000000,    /* chan 28: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_29 = 0x20000000,    /* chan 29: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_30 = 0x40000000,    /* chan 30: 32-channel card  */
        CCURAOCC_CONVERTER_MASK_31 = 0x80000000,    /* chan 31: 32-channel card  */

    /* End Converter */
    CCURAOCC_ALL_CONVERTER_MASK = 0xFFFFFFFF,
} _ccuraocc_converter_mask_t;

typedef struct
{
    int converter_interface_busy;
    int converter_update_mode;
    int converter_data_format;
    int converter_output_range;
} _ccuraocc_converter_csr_t;

typedef _ccuraocc_converter_csr_t
    ccuraocc_converter_csr_t[CCURAOCC_MAX_CONVERTERS];
```

// converter_interface_busy
- CCURAOCC_CONVERTER_IDLE
- CCURAOCC_CONVERTER_BUSY

// converter_update_mode
- CCURAOCC_CONVERTER_MODE_IMMEDIATE
- CCURAOCC_CONVERTER_MODE_SYNCHRONIZED
- CCURAOCC_DO_NOT_CHANGE

// converter_data_format
- CCURAOCC_CONVERTER_OFFSET_BINARY
- CCURAOCC_CONVERTER_TWOS_COMPLEMENT
- CCURAOCC_DO_NOT_CHANGE

// converter_output_range
- CCURAOCC_CONVERTER_UNIPOLAR_5V
- CCURAOCC_CONVERTER_UNIPOLAR_10V
- CCURAOCC_CONVERTER_BIPOLAR_5V
- CCURAOCC_CONVERTER_BIPOLAR_10V
- CCURAOCC_CONVERTER_BIPOLAR_2_5V
- CCURAOCC_DO_NOT_CHANGE

2.2.96 ccurAOCC_Set_Converter_Update_Selection()

This sets the converter update selection to software control or clock control. Clock control is required for FIFO operation.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Converter_Update_Selection (void                         *Handle,
                                    _ccuraocc_converter_update_select_t select)

    Description: Set Converter Update Selection

    Input:      void              *Handle (handle pointer)
                _ccuraocc_converter_update_select_t
                                select  (pointer to converter update selection)
    Output:     none
    Return:     _ccuraocc_lib_error_number_t
                   - CCURAOCC_LIB_NO_ERROR        (successful)
                   - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                   - CCURAOCC_LIB_NOT_OPEN        (device not open)
                   - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                   - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
 ****************************************************************************/


typedef enum
{
    CCURAOCC_CONVERTER_UPDATE_SELECT_SOFTWARE = (0),
    CCURAOCC_CONVERTER_UPDATE_SELECT_PLL_CLOCK = (1),
    CCURAOCC_CONVERTER_UPDATE_SELECT_EXTERNAL_CLOCK = (4),
} _ccuraocc_converter_update_select_t;
```

2.2.97 ccurAOCC_Set_Fifo_Driver_Threshold()

The threshold field ranges from 0 to 0x3FFFF entries representing the number of samples in the FIFO that was last set by the user. This value is used by the driver during FIFO write operations so that if the FIFO has samples that exceed the threshold value, the write will block until the threshold is reached before commencing the write.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Fifo_Driver_Threshold (void *Handle,
                                    uint threshold)

    Description: Set the threshold value in the driver

    Input:      void     *Handle                (handle pointer)
                uint     threshold               (threshold to set)
    Output:     None
    Return:     _ccuraocc_lib_error_number_t
                   - CCURAOCC_LIB_NO_ERROR        (successful)
                   - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                   - CCURAOCC_LIB_NOT_OPEN        (device not open)
                   - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                   - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
 ****************************************************************************/
```

2.2.98 ccurAOCC_Set_Fifo_Threshold()

This call directly updates the hardware FIFO threshold register. In some cases, during FIFO *write* operations, the driver adjusts this threshold based on user supplied threshold *ccurAOCC_Set_Fifo_Driver_Threshold()*, hence, changes to this register may be lost. The user can opt to perform their own FIFO drain management, in which case, this call will be useful.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_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:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR      (successful)
                  - CCURAOCC_LIB_BAD_HANDLE    (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN      (device not open)
                  - CCURAOCC_LIB_INVALID_ARG   (invalid argument)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
 ****************************************************************************/
```

2.2.99 ccurAOCC_Set_Interrupt_Control()

This call is used to enable or disable interrupt handling.

```
/****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Interrupt_Control (void                *Handle,
                                    ccuraocc_interrupt_t *intr)

    Description: Set Interrupt Control information

    Input:      void                 *Handle   (handle pointer)
                ccuraocc_interrupt_t  *intr     (pointer to interrupt control)
    Output:     none
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR      (successful)
                  - CCURAOCC_LIB_BAD_HANDLE    (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN      (device not open)
                  - CCURAOCC_LIB_INVALID_ARG   (invalid argument)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
 ****************************************************************************/
```

```
typedef struct {
    int     global_int;
    int     fifo_buffer_hi_lo_int;
    int     plx_local_int;
} ccuraocc_interrupt_t;
```

// global_int
- CCURAOCC_ICSR_GLOBAL_DISABLE
- CCURAOCC_ICSR_GLOBAL_ENABLE
- CCURAOCC_DO_NOT_CHANGE

// fifo_buffer_hi_lo_int
- CCURAOCC_ICSR_FIFO_HILO_THRESHOLD_DISABLE
- CCURAOCC_ICSR_FIFO_HILO_THRESHOLD_ENABLE

- CCURAOCC_DO_NOT_CHANGE

// plx_local_int
- CCURAOCC_ICSR_LOCAL_PLX_DISABLE
- CCURAOCC_ICSR_LOCAL_PLX_ENABLE
- CCURAOCC_DO_NOT_CHANGE

2.2.100 ccurAOCC_Set_Interrupt_Status()

This call is used to clear the interrupt condition.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Interrupt_Status (void                 *Handle,
                                   ccuraocc_interrupt_t *intr)

    Description: Set Interrupt Status information

    Input:      void                  *Handle  (handle pointer)
                ccuraocc_interrupt_t  *intr    (pointer to interrupt status)
    Output:     none
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR       (successful)
                  - CCURAOCC_LIB_INVALID_ARG    (invalid argument)
 *****************************************************************************/
typedef struct {
    int    global_int;
    int    fifo_buffer_hi_lo_int;
    int    plx_local_int;
} ccuraocc_interrupt_t;
```

// global_int
- not used

// fifo_buffer_hi_lo_int
- CCURAOCC_ICSR_FIFO_HILO_THRESHOLD_DISABLE
- CCURAOCC_ICSR_FIFO_HILO_THRESHOLD_ENABLE
- CCURAOCC_DO_NOT_CHANGE

// plx_local_int
- CCURAOCC_ICSR_LOCAL_PLX_DISABLE
- CCURAOCC_ICSR_LOCAL_PLX_ENABLE
- CCURAOCC_DO_NOT_CHANGE

2.2.101 ccurAOCC_Set_Interrupt_Timeout_Seconds()

This call sets the write *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 write call will wait before it times out. The call could time out if either the FIFO fails to drain or a DMA fails to complete. The device should have been opened in the blocking mode *(O_NONBLOCK not set)* for writes to wait for the operation to complete.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Set_Interrupt_Timeout_Seconds (void *Handle,
                                            int   timeout_secs)

    Description: Set Interrupt Timeout Seconds

    Input:      void          *Handle        (handle pointer)
                int           timeout_secs   (interrupt tout secs)
    Output:     none
```

```
         Return:        _ccuraocc_lib_error_number_t
                          - CCURAOCC_LIB_NO_ERROR    (successful)
                          - CCURAOCC_LIB_INVALID_ARG (invalid argument)
         ******************************************************************/
```

2.2.102 ccurAOCC_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.

```
/**************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Set_PLL_Sync (void               *Handle,
                          ccuraocc_PLL_sync_t *sync)

   Description: Set the value of the PLL Synchronization Register

   Input:       void                *Handle    (handle pointer)
                ccuraocc_PLL_sync_t *sync;      (pointer to sync struct)
   Output:      none
   Return:       _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
 **************************************************************************/

   typedef struct {
       uint    sync_start;
       uint    external_go;
       uint    external_sync;
   } ccuraocc_PLL_sync_t;
   // PLL Sync Start
   - CCURAOCC_PLL_START
   - CCURAOCC_PLL_STOP
   - CCURAOCC_DO_NOT_CHANGE

   // External Go
   - CCURAOCC_EXTERNAL_GO_OUT_ENABLE
   - CCURAOCC_EXTERNAL_GO_OUT_DISABLE
   - CCURAOCC_DO_NOT_CHANGE

   // External Sync
   - CCURAOCC_EXTERNAL_SYNC_OUT_ENABLE
   - CCURAOCC_EXTERNAL_SYNC_OUT_DISABLE
   - CCURAOCC_DO_NOT_CHANGE
```

2.2.103 ccurAOCC_Set_TestBus_Control()

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

```
/**************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Set_TestBus_Control (void                      *Handle,
                                 _ccuraocc_testbus_control_t test_control)

   Description: Set Test Bus Control Selection

   Input:       void                      *Handle    (handle pointer)
                _ccuraocc_testbus_control_t test_control
                                               (pointer to test bus control)
   Output:      none
   Return:       _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR               (successful)
```

```
                            -  CCURAOCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                            -  CCURAOCC_LIB_NOT_OPEN            (device not open)
                            -  CCURAOCC_LIB_INVALID_ARG         (invalid argument)
    *****************************************************************************/


        typedef enum
        {
            CCURAOCC_TBUS_CONTROL_OPEN    = (0),
            CCURAOCC_TBUS_CONTROL_CAL_BUS = (1),
            CCURAOCC_TBUS_CONTROL_5V_REF  = (2),
        } _ccuraocc_testbus_control_t;
```

2.2.104 ccurAOCC_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 *CCURAOCC_CHANNEL_DATA*, *CCURAOCC_GAIN_CALIBRATION* and, *CCURAOCC_OFFSET_CALIBRATION* expect *CCURAOCC_MAX_CHANNELS* unsigned integers. The *CCURAOCC_SPI_RAM* command expect *CCURAOCC_SPI_RAM_SIZE* unsigned integers.

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

```
    /*****************************************************************************
        _ccuraocc_lib_error_number_t
        ccurAOCC_Set_Value (void             *Handle,
                            CCURAOCC_CONTROL cmd,
                            void             *value)

        Description: Set the value of the specified board register.

        Input:       void             *Handle    (handle pointer)
                     CCURAOCC_CONTROL cmd         (register definition)
                     void             *value      (pointer to value to be set)
        Output:      None
        Return:      _ccuraocc_lib_error_number_t
                        -  CCURAOCC_LIB_NO_ERROR    (successful)
                        -  CCURAOCC_LIB_BAD_HANDLE  (no/bad handler supplied)
                        -  CCURAOCC_LIB_NOT_OPEN    (device not open)
                        -  CCURAOCC_LIB_INVALID_ARG (invalid argument)
    *****************************************************************************/


        typedef enum
        {
            CCURAOCC_BOARD_INFORMATION,            /* R Only */
            CCURAOCC_BOARD_CSR,                    /* R/W */

            CCURAOCC_INTERRUPT_CONTROL,            /* R/W */
            CCURAOCC_INTERRUPT_STATUS,             /* R/W */

            CCURAOCC_CONVERTER_CSR_0,              /* R/W : 8 & 32-channel card */
            CCURAOCC_CONVERTER_CSR_1,              /* R/W : 8 & 32-channel card */
            CCURAOCC_CONVERTER_CSR_2,              /* R/W : 8 & 32-channel card */
            CCURAOCC_CONVERTER_CSR_3,              /* R/W : 8 & 32-channel card */
            CCURAOCC_CONVERTER_CSR_4,              /* R/W : 8 & 32-channel card */
            CCURAOCC_CONVERTER_CSR_5,              /* R/W : 8 & 32-channel card */
            CCURAOCC_CONVERTER_CSR_6,              /* R/W : 8 & 32-channel card */
            CCURAOCC_CONVERTER_CSR_7,              /* R/W : 8 & 32-channel card */

            CCURAOCC_CONVERTER_CSR_8,              /* R/W : 32-channel card */
            CCURAOCC_CONVERTER_CSR_9,              /* R/W : 32-channel card */
            CCURAOCC_CONVERTER_CSR_10,             /* R/W : 32-channel card */
            CCURAOCC_CONVERTER_CSR_11,             /* R/W : 32-channel card */
```

```
        CCURAOCC_CONVERTER_CSR_12,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_13,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_14,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_15,              /* R/W : 32-channel card */

        CCURAOCC_CONVERTER_CSR_16,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_17,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_18,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_19,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_20,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_21,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_22,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_23,              /* R/W : 32-channel card */

        CCURAOCC_CONVERTER_CSR_24,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_25,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_26,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_27,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_28,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_29,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_30,              /* R/W : 32-channel card */
        CCURAOCC_CONVERTER_CSR_31,              /* R/W : 32-channel card */

        CCURAOCC_PLL_SYNC,                      /* R/W */

        CCURAOCC_CONVERTER_UPDATE_SELECTION,    /* R/W */
        CCURAOCC_CHANNEL_SELECT,                /* R/W */

        CCURAOCC_CALIBRATOR_BUS_CONTROL,        /* R/W */
        CCURAOCC_TEST_BUS_CONTROL,              /* R/W */
        CCURAOCC_CALIBRATOR_ADC_CONTROL,        /* R/W */

        CCURAOCC_FIFO_CSR,                      /* R/W */
        CCURAOCC_FIFO_THRESHOLD,                /* R/W */

        CCURAOCC_CALIBRATOR_ADC_DATA,           /* R only */

        CCURAOCC_FIRMWARE_SPI_COUNTER_STATUS,   /* R/W */

        CCURAOCC_CHANNEL_DATA,                  /* R/W : 8 & 32-channel card */

        CCURAOCC_CHANNEL_DATA_0,                /* R/W : 8 & 32-channel card */
        CCURAOCC_CHANNEL_DATA_1,                /* R/W : 8 & 32-channel card */
        CCURAOCC_CHANNEL_DATA_2,                /* R/W : 8 & 32-channel card */
        CCURAOCC_CHANNEL_DATA_3,                /* R/W : 8 & 32-channel card */
        CCURAOCC_CHANNEL_DATA_4,                /* R/W : 8 & 32-channel card */
        CCURAOCC_CHANNEL_DATA_5,                /* R/W : 8 & 32-channel card */
        CCURAOCC_CHANNEL_DATA_6,                /* R/W : 8 & 32-channel card */
        CCURAOCC_CHANNEL_DATA_7,                /* R/W : 8 & 32-channel card */

        CCURAOCC_CHANNEL_DATA_8,                /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_9,                /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_10,               /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_11,               /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_12,               /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_13,               /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_14,               /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_15,               /* R/W : 32-channel card */

        CCURAOCC_CHANNEL_DATA_16,               /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_17,               /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_18,               /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_19,               /* R/W : 32-channel card */
        CCURAOCC_CHANNEL_DATA_20,               /* R/W : 32-channel card */
```

```
    CCURAOCC_CHANNEL_DATA_21,                /* R/W : 32-channel card */
    CCURAOCC_CHANNEL_DATA_22,                /* R/W : 32-channel card */
    CCURAOCC_CHANNEL_DATA_23,                /* R/W : 32-channel card */

    CCURAOCC_CHANNEL_DATA_24,                /* R/W : 32-channel card */
    CCURAOCC_CHANNEL_DATA_25,                /* R/W : 32-channel card */
    CCURAOCC_CHANNEL_DATA_26,                /* R/W : 32-channel card */
    CCURAOCC_CHANNEL_DATA_27,                /* R/W : 32-channel card */
    CCURAOCC_CHANNEL_DATA_28,                /* R/W : 32-channel card */
    CCURAOCC_CHANNEL_DATA_29,                /* R/W : 32-channel card */
    CCURAOCC_CHANNEL_DATA_30,                /* R/W : 32-channel card */
    CCURAOCC_CHANNEL_DATA_31,                /* R/W : 32-channel card */

    CCURAOCC_FIFO_DATA,                      /* W Only */

    CCURAOCC_PLL_0_STATUS,                   /* R Only */
    CCURAOCC_PLL_0_ACCESS,                   /* R/W */
    CCURAOCC_PLL_0_READ_1,                   /* R/W */
    CCURAOCC_PLL_0_READ_2,                   /* R/W */

    CCURAOCC_GAIN_CALIBRATION,               /* R/W */
    CCURAOCC_OFFSET_CALIBRATION,             /* R/W */

    CCURAOCC_CALIBRATOR_ADC_POSITIVE_GAIN,   /* R/W */
    CCURAOCC_CALIBRATOR_ADC_NEGATIVE_GAIN,   /* R/W */
    CCURAOCC_CALIBRATOR_ADC_OFFSET,          /* R/W */

    CCURAOCC_SPI_RAM,                        /* R/W */

} CCURAOCC_CONTROL;
```

2.2.105 ccurAOCC_Shutdown_PLL_Clock()

This board has a single programmable clock that supplies clocking to all the converters. This call shuts down the PLL Clock.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Shutdown_PLL_Clock (void *Handle)

    Description: Shutdown_PLL_Clock

    Input:      void              *Handle  (handle pointer)
    Output:     none
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR   (successful)
                  - CCURAOCC_LIB_BAD_HANDLE (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN   (device not open)
 *****************************************************************************/
```

2.2.106 ccurAOCC_Start_PLL_Clock()

This call is used to resume a PLL Clock. No FIFO conversion will take place if the clock is stopped.

```
/*****************************************************************************
    int ccurAOCC_Start_PLL_Clock (void *Handle)
    Description: Start PLL Clock

    Input:      void              *Handle  (handle pointer)
    Output:     none
    Return:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR        (successful)
                  - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
```

```
                   - CCURAOCC_LIB_NOT_OPEN          (device not open)
                   - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
    ******************************************************************************/
```

2.2.107 ccurAOCC_Stop_PLL_Clock()

This call is stops an already running PLL Clock..

```
    /******************************************************************************
       _ccuraocc_lib_error_number_t
       ccurAOCC_Stop_PLL_Clock (void *Handle)

       Description: Stop PLL Clock

       Input:       void                   *Handle  (handle pointer)
       Output:      none
       Return:      _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR          (successful)
                    - CCURAOCC_LIB_BAD_HANDLE        (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN          (device not open)
                    - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
    ******************************************************************************/
```

2.2.108 ccurAOCC_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.

```
    /******************************************************************************
       _ccuraocc_lib_error_number_t
       ccurAOCC_View_Factory_Calibration (void                    *Handle,
                                          _ccuraocc_sprom_access_t item,
                                          char                     *filename)

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

       Input:       void                   *Handle  (handle pointer)
                    _ccuraocc_sprom_access_t item    (select item)
                       -- CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V
                       -- CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V
                       -- CCURAOCC_SPROM_FACTORY_BIPOLAR_5V
                       -- CCURAOCC_SPROM_FACTORY_BIPOLAR_10V
                       -- CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V
       Output:      char                   *filename (pointer to filename)
       Return:      _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR           (successful)
                    - CCURAOCC_LIB_BAD_HANDLE         (no/bad handler supplied)
                    - CCURAOCC_LIB_NOT_OPEN           (device not open)
                    - CCURAOCC_LIB_CANNOT_OPEN_FILE   (file not readable)
                    - CCURAOCC_LIB_NO_LOCAL_REGION    (error)
                    - CCURAOCC_LIB_SERIAL_PROM_BUSY    (serial prom busy)
                    - CCURAOCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
                    - CCURAOCC_LIB_INVALID_ARG        (invalid argument)
    ******************************************************************************/

    typedef enum {
       CCURAOCC_SPROM_HEADER=1,
       CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V,
       CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V,
       CCURAOCC_SPROM_FACTORY_BIPOLAR_5V,
       CCURAOCC_SPROM_FACTORY_BIPOLAR_10V,
       CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V,
```

```
                  CCURAOCC_SPROM_USER_CHECKPOINT_1,
                  CCURAOCC_SPROM_USER_CHECKPOINT_2,
        } _ccuraocc_sprom_access_t;
```

2.2.109 ccurAOCC_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.

```
/***************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_View_User_Checkpoint (void                     *Handle,
                                   _ccuraocc_sprom_access_t item,
                                   char                     *filename)

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

    Input:       void                     *Handle  (handle pointer)
                 _ccuraocc_sprom_access_t item     (select item)
                   -- CCURAOCC_SPROM_USER_CHECKPOINT_1
                   -- CCURAOCC_SPROM_USER_CHECKPOINT_2
    Output:      char                     *filename  (pointer to filename)
    Return:      _ccuraocc_lib_error_number_t
                   - CCURAOCC_LIB_NO_ERROR             (successful)
                   - CCURAOCC_LIB_BAD_HANDLE           (no/bad handler supplied)
                   - CCURAOCC_LIB_NOT_OPEN             (device not open)
                   - CCURAOCC_LIB_CANNOT_OPEN_FILE     (file not readable)
                   - CCURAOCC_LIB_NO_LOCAL_REGION      (error)
                   - CCURAOCC_LIB_SERIAL_PROM_BUSY     (serial prom busy)
                   - CCURAOCC_LIB_SERIAL_PROM_FAILURE  (serial prom failure)
                   - CCURAOCC_LIB_INVALID_ARG          (invalid argument)
 ***************************************************************************/

typedef enum {
    CCURAOCC_SPROM_HEADER=1,
    CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V,
    CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V,
    CCURAOCC_SPROM_FACTORY_BIPOLAR_5V,
    CCURAOCC_SPROM_FACTORY_BIPOLAR_10V,
    CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V,
    CCURAOCC_SPROM_USER_CHECKPOINT_1,
    CCURAOCC_SPROM_USER_CHECKPOINT_2,
} _ccuraocc_sprom_access_t;
```

2.2.110 ccurAOCC_VoltsToData()

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

```
/***************************************************************************
    uint
    ccurAOCC_VoltsToData (double volts,
                          int    format,
                          int    select_voltage_range)

    Description: Convert Volts to data

    Input:       double  volts                  (volts to convert)
                 int     format                 (conversion format)
                 int     select_voltage_range   (select voltage range)
    Output:      none
```

```
   Return:      uint     data                    (returned data)
   ************************************************************************/
```

The *format* can be: CCURAOCC_CONVERTER_OFFSET_BINARY
                     CCURAOCC_CONVERTER_TWOS_COMPLEMENT
If an invalid *format* is supplied, the call defaults to CCURAOCC_CONVERTER_OFFSET_BINARY.

The *select_voltage_range* can be: CCURAOCC_CONVERTER_UNIPOLAR_5V
                                   CCURAOCC_CONVERTER_UNIPOLAR_10V
                                   CCURAOCC_CONVERTER_BIPOLAR_5V
                                   CCURAOCC_CONVERTER_BIPOLAR_10V
                                   CCURAOCC_CONVERTER_BIPOLAR_2_5V

If the data to volts conversion is for the on-board Analog to Digital Converter (ADC), nicknamed
"Calibrator", then the following parameters to be supplied to the *select_voltage_range*.

        CCURAOCC_CALADC_RANGE_BIPOLAR_5V
        CCURAOCC_CALADC_RANGE_BIPOLAR_10V
        CCURAOCC_CALADC_RANGE_BIPOLAR_20V

If an invalid voltage range is selected, the call defaults to CCURAOCC_CONVERTER_UNIPOLAR_5V.

2.2.111 ccurAOCC_VoltsToDataChanCal()

This call converts user supplied volts to raw data for calibration registers.

```
/*****************************************************************************
   uint
   ccurAOCC_VoltsToDataChanCal (double volts)

   Description: Convert Volts to Data (for Channel Calibration)

   Input:       double   volts                   (volts to convert)
   Output:      none
   Return:      uint     data                    (returned data)
   ************************************************************************/
```

2.2.112 ccurAOCC_Wait_For_Channel_Idle()

The write to a channel register takes a finite time to complete. A channel busy indicator is set in the corresponding channel converter. If the busy flag is set and the user attempts to issue another write to the *same* channel, then data could get lost. For this reason, users must make sure that the channel converter is not busy before performing a write. This call basically waits for a channels converter busy bit to go idle before returning.

```
/*****************************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Wait_For_Channel_Idle (void *Handle,
                                   int  chan)

   Description: Wait for Channel to go idle

   Input:       void            *Handle         (handle pointer)
                int             chan            (channel to test)
   Output:      none                            (return busy status)
   Return:      _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR        (successful)
                  - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN        (device not open)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                  - CCURAOCC_LIB_CHANNEL_BUSY    (channel is busy)
                  - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
```

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

2.2.113 ccurAOCC_Wait_For_Interrupt()

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

```
/***********************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_Wait_For_Interrupt (void                    *Handle,
                                 ccuraocc_driver_int_t *drv_int)

   Description: Wait For Interrupt

   Input:       void                     *Handle   (handle pointer)
   Output:      ccuraocc_driver_int_t  *drv_int  (pointer to drv_int struct)
   Return:      _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR         (successful)
                  - CCURAOCC_LIB_BAD_HANDLE       (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN         (device not open)
                  - CCURAOCC_LIB_NO_LOCAL_REGION (local region not present)
                  - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
         ***********************************************************************/

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

// mask
- CCURAOCC_INTSTAT_LOCAL_PLX_MASK
- CCURAOCC_INTSTAT_FIFO_HILO_THRESHOLD_MASK
```

2.2.114 ccurAOCC_Write()

This call basically invokes the *write(2)* system call. The actual write operation performed will depend on the write mode selected via the *ccurAOCC_Select_Driver_Write_Mode()* call prior to invoking this call. For channel write operations, the driver expects any number of samples from 1 to 32 for 32-Channel card or 1 to 8 for the 8-Channel card. These samples are directly written to the channel registers via Programmed I/O or DMA depending on the write mode. If the user has requested one of the FIFO write modes, then they need to ensure that the channel selection is first set and that the samples written should correspond to the active channels. Additionally, prior to starting the clocks, the user will need to "prime" the FIFO, otherwise, they could probably get an under-run and would require resetting of the FIFO to get back in sync with the hardware.

Refer to the *write(2)* system call under *Direct Driver Access* section for more information on the various modes.

```
/***********************************************************************
   _ccuraocc_lib_error_number_t
   ccurAOCC_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:         _ccuraocc_lib_error_number_t
                      - CCURAOCC_LIB_NO_ERROR        (successful)
                      - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                      - CCURAOCC_LIB_NOT_OPEN        (device not open)
                      - CCURAOCC_LIB_IO_ERROR        (write failed)
                      - CCURAOCC_LIB_NOT_IMPLEMENTED (call not implemented)
                      - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
            **************************************************************************/
```

2.2.115 ccurAOCC_Write_Channels()

This call performs a programmed I/O writes to selected channels as specified by information in the *ccuraocc_write_channels_t* structure.

```
    /**************************************************************************
        _ccuraocc_lib_error_number_t
        ccurAOCC_Write_Channels (void                    *Handle,
                            ccuraocc_write_channels_t *wdc)

        Description: Write Channels

        Input:        void                    *Handle (handle pointer)
                      ccuraocc_write_channels_t *wdc   (perform_convertion)
        Output:       ccuraocc_write_channels_t *wdc   (pointer to rdc struct)
        Return:       _ccuraocc_lib_error_number_t
                        - CCURAOCC_LIB_NO_ERROR        (successful)
                        - CCURAOCC_LIB_BAD_HANDLE      (no/bad handler supplied)
                        - CCURAOCC_LIB_NOT_OPEN        (device not open)
                        - CCURAOCC_LIB_INVALID_ARG     (invalid argument)
         **************************************************************************/
    typedef struct
    {
        char select_channel;
        union
        {
            char convert_rawdata_to_volts;  /* for reading from channel registers */
            char convert_volts_to_rawdata;  /* for writing to channel registers */
        };
        char channel_synchronized_update_flag;
        char converter_data_format;
        char converter_output_range;
        int channel_data_raw;
        double channel_data_volts;
    } ccuraocc_single_channel_data_t;

    typedef struct
    {
        ccuraocc_single_channel_data_t wchan[CCURAOCC_MAX_CHANNELS];
    } ccuraocc_write_channels_t;
```

The user needs to set the *select_channel* and the *convert_volts_to_rawdata* fields in the *ccuraocc_single_channel_data_t* structure for information on each channel they need to write. To select a channel, the *select_channel* field needs to be set to *CCURAOCC_TRUE*. The call will write the *channel_data_raw* content in the structure to the channel register, unless, the *convert_volts_to_rawdata* field is set to *CCURAOCC_TRUE*. In that case, the call will convert the floating point voltage in the *channel_data_volts* to raw and write that to the channel register. Additionally, this raw information will also be stored in the *channel_data_raw* field of the structure.

2.2.116 ccurAOCC_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 *ccurAOCC_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.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_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:     _ccuraocc_lib_error_number_t
                  - CCURAOCC_LIB_NO_ERROR            (successful)
                  - CCURAOCC_LIB_BAD_HANDLE          (no/bad handler supplied)
                  - CCURAOCC_LIB_NOT_OPEN            (device not open)
                  - CCURAOCC_LIB_INVALID_ARG         (invalid argument)
                  - CCURAOCC_LIB_NO_LOCAL_REGION     (local region not
present)
                  - CCURAOCC_LIB_IO_ERROR            (read error)
                  - CCURAOCC_LIB_CANNOT_OPEN_FILE    (file not writeable)
                  - CCURAOCC_LIB_CALIBRATION_RANGE_ERROR (range error)
 *****************************************************************************/
```

Format:

```
#Chan   Offset                Gain
#====   ======                ====
 ch00:  0.1983642578125000    0.3991699218750000
 ch01:  0.0860595703125000    0.2078247070312500
 ch02:  0.1992797851562500    0.4129028320312500
 ch03:  0.0830078125000000    0.1345825195312500
----
 ch28:  0.1766967773437500    0.3732299804687500
 ch29:  0.1361083984375000    0.2694702148437500
 ch30:  0.1257324218750000    0.2728271484375000
 ch31:  0.0469970703125000    0.0830078125000000
```

2.2.117 ccurAOCC_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 *ccurAOCC_Serial_Prom_Write_Override()* to allowing writing to the serial prom.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Write_Serial_Prom(void              *Handle,
                               ccuraocc_sprom_rw_t *spw)

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

    Input:      void              *Handle         (handle pointer)
                ccuraocc_sprom_rw_t *spw           (pointer to struct)
                  -- u_short word_offset
                  -- u_short num_words
```

```
                        -- u_short *data_ptr
    Output:       none
    Return:       _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR          (successful)
                    - CCURAOCC_LIB_NO_LOCAL_REGION   (error)
                    - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
                    - CCURAOCC_LIB_SERIAL_PROM_BUSY  (serial prom busy)
                    - CCURAOCC_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 */
   } ccuraocc_sprom_rw_t;
```

2.2.118 ccurAOCC_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 *ccurAOCC_Serial_Prom_Write_Override()* to allowing writing to the serial prom.

```
   /***************************************************************************
      _ccuraocc_lib_error_number_t
      ccurAOCC_Write_Serial_Prom_Item(void                    *Handle,
                                      _ccuraocc_sprom_access_t item,
                                      void                    *item_ptr)

   Description: Write Serial Prom with specified item

   Input:        void                   *Handle  (handle pointer)
                 _ccuraocc_sprom_access_t item    (select item)
                    -- CCURAOCC_SPROM_HEADER
                    -- CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V
                    -- CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V
                    -- CCURAOCC_SPROM_FACTORY_BIPOLAR_5V
                    -- CCURAOCC_SPROM_FACTORY_BIPOLAR_10V
                    -- CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V
                    -- CCURAOCC_SPROM_USER_CHECKPOINT_1
                    -- CCURAOCC_SPROM_USER_CHECKPOINT_2
   Output:       void                   *item_ptr (pointer to item struct)
                    -- *ccuraocc_sprom_header_t
                    -- *ccuraocc_sprom_factory_t
                    -- *ccuraocc_sprom_user_checkpoint_t
   Return:       _ccuraocc_lib_error_number_t
                    - CCURAOCC_LIB_NO_ERROR          (successful)
                    - CCURAOCC_LIB_NO_LOCAL_REGION   (error)
                    - CCURAOCC_LIB_INVALID_ARG       (invalid argument)
                    - CCURAOCC_LIB_SERIAL_PROM_BUSY  (serial prom busy)
                    - CCURAOCC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
   ***************************************************************************/

   typedef enum {
       CCURAOCC_SPROM_HEADER=1,
       CCURAOCC_SPROM_FACTORY_UNIPOLAR_5V,
       CCURAOCC_SPROM_FACTORY_UNIPOLAR_10V,
       CCURAOCC_SPROM_FACTORY_BIPOLAR_5V,
       CCURAOCC_SPROM_FACTORY_BIPOLAR_10V,
```

```
            CCURAOCC_SPROM_FACTORY_BIPOLAR_2_5V,
            CCURAOCC_SPROM_USER_CHECKPOINT_1,
            CCURAOCC_SPROM_USER_CHECKPOINT_2,
    } _ccuraocc_sprom_access_t;
```

The *void* pointer **item_ptr* points to one of the following structures depending on the selected *item* that needs to be written.

```
typedef struct {
    u_int   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 */
} ccuraocc_sprom_header_t;

typedef struct {
    u_short crc;                            /* 0x000 - 0x001 - CRC */
    u_short spare_002_007[0x6/2];           /* 0x002 - 0x007 - spare */
    time_t  date;                           /* 0x008 - 0x00F - date */
    u_short offset[CCURAOCC_MAX_CHANNELS];  /* 0x010 - 0x04F - offset */
    u_short gain[CCURAOCC_MAX_CHANNELS];    /* 0x050 - 0x08F - gain */
} ccuraocc_sprom_factory_t;

typedef struct {
    u_short crc;                            /* 0x000 - 0x001 - CRC */
    u_short spare_002_007[0x6/2];           /* 0x002 - 0x007 - spare */
    time_t  date;                           /* 0x008 - 0x00F - date */
    u_short offset[CCURAOCC_MAX_CHANNELS];  /* 0x010 - 0x04F - offset */
    u_short gain[CCURAOCC_MAX_CHANNELS];    /* 0x050 - 0x08F - gain */
    u_int   converter_csr[CCURAOCC_MAX_CONVERTERS];
                                            /* 0x090 - 0x10F - channel config */
} ccuraocc_sprom_user_checkpoint_t;
```

2.2.119 ccurAOCC_Write_Single_Channel()

This call is similar to the *ccurAOCC_Write_Channels()*, except, information is written for a single channel.

```
/*****************************************************************************
    _ccuraocc_lib_error_number_t
    ccurAOCC_Write_Single_Channel (void                          *Handle,
                                   int                           chan,
                                   ccuraocc_single_channel_data_t *wdc)
    Description: Write Single Channel

    Input:      void                          *Handle (handle pointer)
                int                           chan    (channel to write)
                ccuraocc_single_channel_data_t *wdc    (perform_convertion)
    Output:     ccuraocc_single_channel_data_t *wdc    (pointer to wdc struct)
    Return:     _ccuraocc_lib_error_number_t
                 - CCURAOCC_LIB_NO_ERROR                (successful)
                 - CCURAOCC_LIB_BAD_HANDLE              (no/bad handler supplied)
                 - CCURAOCC_LIB_NOT_OPEN                (device not open)
                 - CCURAOCC_LIB_INVALID_ARG             (invalid argument)
 *****************************************************************************/

typedef struct
{
    char select_channel;
    union
    {
        char convert_rawdata_to_volts;  /* for reading from channel registers */
        char convert_volts_to_rawdata;  /* for writing to channel registers */
    };
```

```
        char channel_synchronized_update_flag;
        char converter_data_format;
        char converter_output_range;
        int channel_data_raw;
        double channel_data_volts;
} ccuraocc_single_channel_data_t;
```

The user needs to set the channel number in *chan*. If the *convert_volts_to_rawdata* flag is set to *CCURAOCC_TRUE,* the call takes the user supplied voltage in the *channel_data_volts* and converts it to raw data based on the customer supplied data format and voltage range. Additionally, the converted raw value will also be placed in the *channel_data_raw* field.

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 ccuraocc_dump

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

Usage: ccuraocc_dump [-b board]

<u>Example display:</u>

```
Device Name    : /dev/ccuraocc0
Board Serial No: 0 (0x00000000)

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

======= LOCAL BOARD REGISTERS =========
LBR: @0x0000 --> 0x92870141
LBR: @0x0004 --> 0x00000301
LBR: @0x0008 --> 0x00000000
. . .
LBR: @0x07f4 --> 0x00000000
LBR: @0x07f8 --> 0x00000000
LBR: @0x07fc --> 0x00000000

======= LOCAL CONFIG REGISTERS =========
LCR: @0x0000 --> 0xfffff800
LCR: @0x0004 --> 0x00000001
LCR: @0x0008 --> 0x00200000
. . .
LCR: @0x00fc --> 0x00000000
LCR: @0x0100 --> 0x00000000
LCR: @0x0104 --> 0x00000000

======= PCI CONFIG REG ADDR MAPPING =========
PCR: @0x0000 --> 0x92871542
PCR: @0x0004 --> 0x02b00017
PCR: @0x0008 --> 0x08800001
. . .
PCR: @0x0048 --> 0x00004c00
PCR: @0x004c --> 0x00000003
PCR: @0x0050 --> 0x00000000

======= PCI BRIDGE REGISTERS =========
PBR: @0x0000 --> 0x811110b5
PBR: @0x0004 --> 0x00100417
PBR: @0x0008 --> 0x06040021
. . .
PBR: @0x0110 --> 0x00000000
```

```
    PBR: @0x0114 --> 0x00000000
    PBR: @0x0118 --> 0x00000000


    ======= MAIN CONTROL REGISTERS =========
    MCR: @0x0000 --> 0x00000033
    MCR: @0x0004 --> 0x8000ff00
    MCR: @0x0008 --> 0x00000000
    . . .
    MCR: @0x005c --> 0x0000029a
    MCR: @0x0060 --> 0x00000019
    MCR: @0x0064 --> 0x00000000
```

3.1.2 ccuraocc_rdreg

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

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

<u>Example display:</u>

```
Device Name    : /dev/ccuraocc0
Board Serial No: 12345678 (0x00bc614e)

Read at offset 0x0000: 0x92870123
```

3.1.3 ccuraocc_reg

This test dumps the board registers.

Usage: ccuraocc_reg [-b board]

<u>Example display:</u>

```
Device Name    : /dev/ccuraocc0
Board Serial No: 12345678 (0x00bc614e)

LOCAL Register 0x7ffff7ffc000 Offset=0x0

#### LOCAL REGS #### (length=2048)
+LCL+       0   92870121  00000301  00000000  00000000 *...!............*
+LCL+    0x10   00000001  00000001  00000001  00000001 *................*
+LCL+    0x20   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=252)
+CFG+       0   fffff800  00000001  00200000  00300400 *.......... ...0..*
+CFG+    0x10   00000000  00000000  42430343  00000000 *........BC.C....*
+CFG+    0x20   00000000  00000000  00000000  00000000 *................*
. . .
+CFG+    0xd0   00000000  00000000  00000000  00000000 *................*
+CFG+    0xe0   00000000  00000000  00000050  00000000 *...........P....*
+CFG+    0xf0   00000000  00000000  00000043           *...........C    *
```

```
======= 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                   =0x100f767c     @0x0000006c
    pcihidr                 =0x905610b5     @0x00000070
    pcihrev                 =0x000000ba     @0x00000074
    dmamode0                =0x00000043     @0x00000080
    dmapadr0                =0x79f00000     @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                 =0x00001011     @0x000000a8
    dmacsr1                 =0x00200000     @0x000000ac
    las1rr                  =0x00000000     @0x000000f0
    las1ba                  =0x00000000     @0x000000f4
    lbrd1                   =0x00000043     @0x000000f8
    dmdac                   =0x00000000     @0x000000fc
    pciarb                  =0x00000000     @0x00000100
    pabtadr                 =0x1cc8ffc0     @0x00000104

======= LOCAL REGISTERS =========
    board_info              =0x92870201     @0x00000000
    board_csr               =0x00000301     @0x00000004
    interrupt_control       =0x00000000     @0x00000008
    interrupt_status        =0x00000000     @0x0000000c
    converter_csr[0]        =0x00000000     @0x00000020
    converter_csr[1]        =0x00000000     @0x00000024
    converter_csr[2]        =0x00000000     @0x00000028
    converter_csr[3]        =0x00000000     @0x0000002c
    converter_csr[4]        =0x00000000     @0x00000030
    converter_csr[5]        =0x00000000     @0x00000034
    converter_csr[6]        =0x00000000     @0x00000038
    converter_csr[7]        =0x00000000     @0x0000003c
    converter_csr[8]        =0x00000000     @0x00000040
    converter_csr[9]        =0x00000000     @0x00000044
    converter_csr[10]       =0x00000000     @0x00000048
```

```
        converter_csr[11]            =0x00000000        @0x0000004c
        converter_csr[12]            =0x00000000        @0x00000050
        converter_csr[13]            =0x00000000        @0x00000054
        converter_csr[14]            =0x00000000        @0x00000058
        converter_csr[15]            =0x00000000        @0x0000005c
        converter_csr[16]            =0x00000000        @0x00000060
        converter_csr[17]            =0x00000000        @0x00000064
        converter_csr[18]            =0x00000000        @0x00000068
        converter_csr[19]            =0x00000000        @0x0000006c
        converter_csr[20]            =0x00000000        @0x00000070
        converter_csr[21]            =0x00000000        @0x00000074
        converter_csr[22]            =0x00000000        @0x00000078
        converter_csr[23]            =0x00000000        @0x0000007c
        converter_csr[24]            =0x00000000        @0x00000080
        converter_csr[25]            =0x00000000        @0x00000084
        converter_csr[26]            =0x00000000        @0x00000088
        converter_csr[27]            =0x00000000        @0x0000008c
        converter_csr[28]            =0x00000000        @0x00000090
        converter_csr[29]            =0x00000000        @0x00000094
        converter_csr[30]            =0x00000000        @0x00000098
        converter_csr[31]            =0x00000000        @0x0000009c
        PLL_sync                     =0x00000000        @0x000000a0
        converter_update_select      =0x00000000        @0x000000a4
        channel_select               =0xffffffff        @0x000000a8
        calib_bus_control            =0x00000000        @0x000000b0
        test_bus_control             =0x00000000        @0x000000b4
        calib_adc_control            =0x00000003        @0x000000b8
        fifo_csr                     =0x85000000        @0x000000c0
        fifo_threshold               =0x0001fc00        @0x000000c4
        WriteSampleCount             =0x00004000        @0x000000c8
        ScopeTrigger                 =0x00000002        @0x000000cc
        calib_adc_data               =0x00000002        @0x000000d0
        spi_counter_status           =0x00000000        @0x000000f0

        channel_data[0..31]
@0x0100 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x0120 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x0140 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x0160 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

        fifo_data                    =0x00000001        @0x00000190
        pll[P0].PLL_status           =0x00000000        @0x000001a0
        pll[P0].PLL_access           =0x00000600        @0x000001a4
        pll[P0].PLL_read_1           =0x00000000        @0x000001a8
        pll[P0].PLL_read_2           =0x00000000        @0x000001ac

        gain_calibration[0..31]
@0x0200 0000051c 000002a9 00000549 000001b9 000002fe 000004ec 00000526 0000051c
@0x0220 000002f9 0000027b 0000054c 0000058e 0000014c 00000280 00000625 00000687
@0x0240 00000394 0000069d 00000604 00000256 000000ee 00000226 0000039c 00000822
@0x0260 00000450 0000020f 0000023b 00000672 000004c7 00000373 0000037e 00000110

        offset_calibration[0..31]
@0x0280 0000028a 0000011a 0000028d 00000110 00000184 000002a2 000002b7 000002b9
@0x02a0 0000013b 0000012e 00000290 00000291 000000a6 00000119 00000308 00000313
@0x02c0 000001c3 0000033f 00000320 000000fb 0000009d 0000012f 000001c0 0000042b
@0x02e0 0000020c 00000117 00000125 0000036c 00000243 000001be 0000019c 0000009a

        calib_adc_positive_gain      =0x8006c6f0        @0x00000400
        calib_adc_negative_gain      =0x8008759d        @0x00000404
        calib_adc_offset             =0x00000002        @0x00000408
        sprom_stat_addr_write_data   =0x03ff0000        @0x00000500
        sprom_read_data              =0x03ff0000        @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
```

3.1.4 ccuraocc_regedit

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

Usage: ccuraocc_tst [-b board]

Example display:

```
Device Name     : /dev/ccuraocc0
Board Serial No : 12345678 (0x00bc614e)
Initialize_Board: Firmware Rev. 0x01 successful

Virtual Address: 0x7ffff7ffc000
    1 = Create Physical Memory       2 = Destroy Physical memory
    3 = Display Channel Data          4 = Display Driver Information
    5 = Display Firmware RAM          6 = Display Physical Memory Info
    7 = Display Registers (CONFIG)    8 = Display Registers (LOCAL)
    9 = Dump Physical Memory         10 = Reset Board
   11 = Write Register (LOCAL)       12 = Write Register (CONFIG)
   13 = Write Physical Memory

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

3.1.5 ccuraocc_tst

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

Usage: ccuraocc_tst [-b board]

Example display:

```
Device Name     : /dev/ccuraocc0
Board Serial No : 12345678 (0x00bc614e)
Initialize_Board: Firmware Rev. 0x01 successful

  01 = add irq                      02 = disable pci interrupts
  03 = enable pci interrupts        04 = get device error
  05 = get driver info              06 = get physical mem
  07 = init board                   08 = mmap select
  09 = mmap(CONFIG registers)       10 = mmap(LOCAL registers)
  11 = mmap(physical memory)        12 = munmap(physical memory)
  13 = no command                   14 = read operation
  15 = remove irq                   16 = reset board
  17 = write operation

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

3.1.6 ccuraocc_wreg

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

```
Usage: ./ccuraocc_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 offset is 0x4
 -v value: hex value to write at offset -- default value is 0x0
```

Example display:

```
Device Name    : /dev/ccuraocc0
Board Serial No: 12345678 (0x00bc614e)

Writing 0x00000000 to offset 0x0000
Read at offset 0x0000: 0x92870123
```

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/ccuraocc_calibrate

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

```
Usage: ./ccuraocc_calibrate [-A] [-b board] [-C ChanMask] [-f format]
                            [-i inCalFile] [-o outCalFile] [-p] [-R] [-T TestBus]
                            [-V VoltageRange] [-X ExtClock] [-Z CalBusCtrl]
 -A                  (perform Auto Calibration)
 -b <board>          (board #, default = 0)
 -C <ChanMask>       (channel selection mask, default = all channels)
 -f <format 'b', '2'> (default  = 'b' Offset Binary)
 -i <In Cal File>    (input calibration file [input->board_reg])
 -o <Out Cal File>   (output calibration file [board_reg->output])
 -p                  (program board converters)
 -R                  (reset channel calibration)
 -T <TestBus>        (default  = No Change
                         'b'    - Calibration Bus Control
                         'o'    - Open
                         'r'    - 5 Volt Reference
 -V <VoltageRange>   (default  = 'b10' Bipolar 10 volts)
                         'u5'   - Unipolar 5 volts  (  +0 --> +5  )
                         'u10'  - Unipolar 10 volts (  +0 --> +10 )
                         'b5'   - Bipolar 5 volts   (  -5 --> +5  )
                         'b10'  - Bipolar 10 volts  ( -10 --> +10 )
                         'b2.5' - Bipolar 2.5 volts (-2.5 --> +2.5)
 -X [s,p,e]          (Board External Clock Output Selection)
                         's'    - software clock output
                         'p'    - PLL clock output
                         's'    - External clock output
 -Z <CalBusCtrl>     (default  = No Change
                         'g'    - Ground
                         'n'    - Negative
                         'o'    - Open
                         'p'    - Positive
                         '0..31'- Channel Number

 e.g. ./ccuraocc_calibrate -A (Auto Calibrate all channels)
      ./ccuraocc_calibrate -R -C0xf (Reset calibration for channels 0-3)
```

Example display:

./ccuraocc_calibrate -b0 -A -Vb10   (32 channel, single-ended)

```
Device Name    : /dev/ccuraocc0
Board Serial No: 691349 (0x000a8c95)
#### Programming Board ####
#### Auto Calibration Started ####
#### Auto Calibration Complete ####
Total Time: 2 sec

===> Dump to 'stdout'
#Date          : Tue Oct  1 11:34:08 2019
#Board Serial No: 691349 (0x000a8c95)

#Chan   Offset              Gain                Range
#====   ==================  ==================  ============
 ch00:   0.0537109375000000  -1.2094116210937500  BiPolar 10v
 ch01:   0.0543212890625000  -1.2023925781250000  BiPolar 10v
 ch02:   0.1489257812500000  -1.0021972656250000  BiPolar 10v
 ch03:   0.1361083984375000  -1.0693359375000000  BiPolar 10v
 ch04:   0.0180053710937500  -1.2915039062500000  BiPolar 10v
 ch05:  -0.0326538085937500  -1.4224243164062500  BiPolar 10v
 ch06:   0.0195312500000000  -1.3308715820312500  BiPolar 10v
 ch07:   0.0677490234375000  -1.1968994140625000  BiPolar 10v
 ch08:   0.0421142578125000   0.0695800781250000  BiPolar 10v
 ch09:  -0.0192260742187500  -0.0186157226562500  BiPolar 10v
 ch10:   0.1025390625000000   0.1678466796875000  BiPolar 10v
 ch11:   0.0027465820312500  -0.0073242187500000  BiPolar 10v
 ch12:  -0.0161743164062500  -0.0039672851562500  BiPolar 10v
 ch13:   0.0146484375000000   0.0524902343750000  BiPolar 10v
 ch14:  -0.0134277343750000  -0.0384521484375000  BiPolar 10v
 ch15:  -0.0231933593750000  -0.0509643554687500  BiPolar 10v
 ch16:  -0.0744628906250000  -0.1531982421875000  BiPolar 10v
 ch17:   0.0192260742187500   0.0430297851562500  BiPolar 10v
 ch18:   0.0677490234375000   0.0711059570312500  BiPolar 10v
 ch19:   0.0268554687500000   0.0302124023437500  BiPolar 10v
 ch20:   0.0515747070312500   0.1046752929687500  BiPolar 10v
 ch21:   0.0433349609375000   0.0897216796875000  BiPolar 10v
 ch22:   0.0396728515625000   0.0708007812500000  BiPolar 10v
 ch23:   0.0271606445312500   0.0451660156250000  BiPolar 10v
 ch24:   0.0732421875000000   0.1477050781250000  BiPolar 10v
 ch25:  -0.0018310546875000  -0.0073242187500000  BiPolar 10v
 ch26:  -0.0097656250000000  -0.0549316406250000  BiPolar 10v
 ch27:   0.0030517578125000  -0.0006103515625000  BiPolar 10v
 ch28:   0.0064086914062500   0.0375366210937500  BiPolar 10v
 ch29:   0.0332641601562500   0.0363159179687500  BiPolar 10v
 ch30:   0.0286865234375000   0.0277709960937500  BiPolar 10v
 ch31:  -0.0091552734375000   0.0039672851562500  BiPolar 10v
```

3.2.2 lib/ccuraocc_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: ./ccuraocc_compute_pll_clock -[bfstv]
         -b <board>              (board #, default = 0)
         -f <desired freq>       (default = 13.824000 MHz)
         -f <freq_start,freq_end,freq_inc>
         -s                      (Minimize VCO Speed)
         -t <max error tolerance> (default = 1000 ppm)
         -v                      (enable verbose)
```

<u>Example display:</u>

```
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)          = 100
        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/ccuraocc_disp

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

```
Usage: ./ccuraocc_disp [-A] [-a#] [-b board] [-C] [-d delay] [-D debugfile]
                       [-E ExpInp] [-f format] [-l loopcnt] [-m mode]
                       [-n numchans] [-o outfile] [-p] [-v OutputVolts]
                       [-V OutputRange] [-X ExtClock]
 -A                      (perform Auto Calibration)
 -a <#>                  (display rolling average of # values.)
 -b <board>             (default = 0)
 -C                      (Display Calibration Gain and Offset)
 -d <delay - msec>      (delay between screen refresh)
 -D <Debug File>        (write to debug file)
 -D @<Debug File>       (No display, however write to debug file)
 -D @                    (No Display and no writing to debug file)
 -E                      (Expected Input Voltage taken from '-v' option. Default
                          tolerance '0.005000')
 -E @<Tol>              (Expected tolerance @Tolerance. Voltage taken from '-v'
                          option)
 -E <ExpInpVolts>@<Tol> (Expected Input Volts@Tolerance. Default tolerance
                          '0.005000')
 -f <format 'b', '2'>   (default = 'b' Offset Binary)
 -l <#>                  (specify loop count)
 -ma                     (ADC Channel Readback mode [CHANNEL])
 -md                     (User DMA read mode [CHANNEL])
 -mD                     (Driver DMA read mode [CHANNEL])
 -mp                     (User PIO read mode [CHANNEL])
 -mP                     (Driver PIO read mode [CHANNEL])
 -n <#>                  (number of channels to display)
 -o <#>@<Output File>   (average # count, write to output file)
 -p                      (program board to max clock first)
 -v <output volts>      (default = '10.000000')
 -V <OutputRange>       (default = 'b10' Bipolar 10 volts)
                               'u5'  - Unipolar 5 volts  (  +0 --> +5  )
```

```
                                    'u10' - Unipolar 10 volts  (  +0 --> +10 )
                                    'b5'  - Bipolar 5 volts    (  -5 --> +5  )
                                    'b10' - Bipolar 10 volts   ( -10 --> +10 )
                                    'b2.5' - Bipolar 2.5 volts (-2.5 --> +2.5)
     -X [s,p,e]                (Board External Clock Output Selection)
                                    's'   - software clock output
                                    'p'   - PLL clock output


     e.g. ./ccuraocc_disp -A -Vb10 -v5.7 -E5.7 -ma (Autocal, +/-10V range, 5.7V,
                          validate, ADC channel readback)
          ./ccuraocc_disp -A -Vb10 -v4.3 -E -ma (Autocal, +/-10V range, 4.3V,
                          validate, ADC channel readback)
```

Example display:

./ccuraocc_disp -ma -Vb10 -v10 -ma -E -l200 -A   (32 channel, single-ended)

```
Device Name: /dev/ccuraocc0

local_ptr: 0x7ffff7ff6000

config_ptr: 0x7ffff7ff5000
#### Auto Calibration Started ####
#### Auto Calibration Complete ####

Rolling Average Count [-a]: 1
Board Number        [-b]: 0  ==> '/dev/ccuraocc0' (32-Channel, 10-Volt, Single-Ended Card)
Board Serial Number     : 691349 (0x000a8c95)
Delay               [-d]: 0 milli-seconds
Expected Input Volts [-E]: 10.000000 volts (Tolerance 0.005000 volts)
Data Format         [-f]: Offset Binary
Loop Count          [-l]: 2000
Read Mode           [-m]: ADC Channel Readback (Channel Data)
Write Mode              : Driver DMA (Channel Data)
Program Board       [-p]: Yes
Output Range        [-V]: (b10) Bipolar 10 volts
All Converters State    : ==== Active ====
External Clock          : **** Not Detected ****
External Clock Output [-X]: Software Clock
Read Error?             : ===== no ==== (0)
Tolerance Exceeded Count : 0
Calibrator ADC Data     : Raw=10000    Volts= 10.00000000 [Bipolar -10V to +10V (40V p-p)]
         ADC Positive   : Raw=800de145 Volts=  1.00042358
         ADC Negative   : Raw=800a181d Volts=  1.00030805
         ADC Offset     : Raw=3ffff    Volts= -0.00015259
         Test Bus Ctrl  : Open (0x00)
         Bus Control    : Channel 31 (0x3f)

Scan count:        200, Total Delta: 10347.0 usec (min=10304.3,max=10374.0,av=10354.0)

        <<<<=== [ADC Readback] Raw Data (Rolling Average Count [1/1]) ===>>>>
        [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]
        =====    =====    =====    =====    =====    =====    =====    =====
[00-07] 0fffe    00002    00000    0ffff    00002    0fffe    00001    00000
[08-15] 00001    0fffe    00000    00002    00003    0ffff    00000    00000
[16-23] 0ffff    00000    00000    00000    00001    00002    0fffe    00000
[24-31] 0ffff    00002    00000    00000    0ffff    0fffe    00001    00002

        <<<<=== [ADC Readback] Volts (Rolling Average Count [1/1]) ===>>>>
        [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]
        ======== ======== ======== ======== ======== ======== ======== ========
[00-07] -0.00031 +0.00031 +0.00000 -0.00015 +0.00031 -0.00031 +0.00015 +0.00000
[08-15] +0.00015 -0.00031 +0.00000 +0.00031 +0.00046 -0.00015 +0.00000 +0.00000
[16-23] -0.00015 +0.00000 +0.00000 +0.00000 +0.00015 +0.00031 -0.00031 +0.00000
[24-31] -0.00015 +0.00031 +0.00000 +0.00000 -0.00015 -0.00031 +0.00015 +0.00031

                Date: Tue Oct  1 11:40:33 2019
   Expected Input Volts: 10.000000 volts (Tolerance 0.005000 volts)
 Tolerence Exceed Count: 0
```

```
              Output Volts: '10.000000'
               Data Format: 'Offset Binary'
              Output Range: 'Bipolar 10 volts'
               Output File: '', Rolling Average Count: 1/1
                Scan Count: 2000

    WorstMinChanVoltsHWM:  -0.000763 (Ch05)
    WorstMaxChanVoltsHWM:   0.000610 (Ch26)


================================================================================
Chan  Minimum   Maximum   Average   DataFormat  OutputRange  TolerExeededCnt
====  =========  =========  =========  ==========  ===========  ================
   0  -0.000458  +0.000458  +0.000305    ofbin        b10             -
   1  -0.000610  +0.000458  -0.000153    ofbin        b10             -
   2  -0.000610  +0.000458  +0.000000    ofbin        b10             -
   3  -0.000610  +0.000458  -0.000305    ofbin        b10             -
   4  -0.000610  +0.000458  -0.000153    ofbin        b10             -
   5  -0.000763  +0.000458  +0.000000    ofbin        b10             -
   6  -0.000610  +0.000458  -0.000153    ofbin        b10             -
   7  -0.000458  +0.000458  -0.000153    ofbin        b10             -
   8  -0.000458  +0.000458  +0.000153    ofbin        b10             -
   9  -0.000610  +0.000458  -0.000153    ofbin        b10             -
  10  -0.000458  +0.000458  +0.000000    ofbin        b10             -
  11  -0.000458  +0.000458  +0.000000    ofbin        b10             -
  12  -0.000610  +0.000458  +0.000305    ofbin        b10             -
  13  -0.000458  +0.000458  -0.000305    ofbin        b10             -
  14  -0.000610  +0.000458  +0.000153    ofbin        b10             -
  15  -0.000458  +0.000458  -0.000153    ofbin        b10             -
  16  -0.000610  +0.000458  -0.000153    ofbin        b10             -
  17  -0.000458  +0.000610  -0.000305    ofbin        b10             -
  18  -0.000610  +0.000458  +0.000305    ofbin        b10             -
  19  -0.000458  +0.000458  -0.000305    ofbin        b10             -
  20  -0.000458  +0.000458  +0.000153    ofbin        b10             -
  21  -0.000610  +0.000458  -0.000153    ofbin        b10             -
  22  -0.000458  +0.000458  -0.000153    ofbin        b10             -
  23  -0.000610  +0.000458  +0.000000    ofbin        b10             -
  24  -0.000610  +0.000458  +0.000000    ofbin        b10             -
  25  -0.000458  +0.000458  +0.000000    ofbin        b10             -
  26  -0.000610  +0.000610  +0.000000    ofbin        b10             -
  27  -0.000458  +0.000458  +0.000000    ofbin        b10             -
  28  -0.000610  +0.000458  -0.000153    ofbin        b10             -
  29  -0.000458  +0.000458  -0.000153    ofbin        b10             -
  30  -0.000610  +0.000458  -0.000305    ofbin        b10             -
  31  -0.000610  +0.000458  +0.000305    ofbin        b10             -
```

3.2.4 lib/ccuraocc_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: ./ccuraocc_identify -[absx]
        -a                  (Identify all cards through a light sequence)
        -b <board>          (board #, default = 0)
        -s <seconds)        (Identify Board: ENABLED for number of seconds,
                             default = 10)
        -s 0                (Identify Board: DISABLED)
        -s <negative value> (Identify Board: ENABLED forever)
        -x                  (silent)

If the '-a' option is selected, all other options are ignored. This option will
sequence through all the cards found in turn as follows:
  1) The first device number will flash its LED for 10 seconds
  2) The remaining devices numbers will be selected sequentially and flash their
     LEDs for 3 seconds
```

<u>Example display:</u>

./ccuraocc_identify -b0   (32-channel, differential)

```
Device Name    : /dev/ccuraocc0
Board ID       : 9287
Board Type     : 01 (32-Channel, Differential)
Board NumChans : 32
Board Wire     : 1 (Differential)
Board Revision : 02
Board Serial No: 652005 (0x0009f2e5)

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

./ccuraocc_identify -b1   (8-channel, single-ended)

```
Device Name    : /dev/ccuraocc1
Board ID       : 9287
Board Type     : 04 (8-Channel, Single-Ended)
Board NumChans : 8
Board Wire     : 0 (Single_Ended)
Board Revision : 41
Board Serial No: 683079 (0x000a6c47)

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

3.2.5 lib/ccuraocc_setchan

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

```
Usage: ./ccuraocc_setchan [-A] [-b board] [-C ChanMask] [-e ExtOutClk]
                          [-f format] [-F SampleRate] [-l LoopCnt] [-m WriteMode]
                          [-n NumSamples] [-p] [-R] [-S] [-t Timeout]
                          [-T TestBus] [-u] [-v OutputVolts] [-V OutputRange]
                          [-w WaveType] [-Z CalBusCtrl]
 -A                    (perform Auto Calibration)
 -b <board>            (board #, default = 0)
 -C <ChanMask>         (channel selection mask, default = all channels)
 -e <ExtOutClk>        (external output clock, default  = no change)
                           's'    - Software Flag
                           'p'    - PLL Clock
                           'e'    - External Clock
 -f <format 'b', '2'>  (default  = 'b' Offset Binary)
 -F <Sample Rate>      (default  = '400000.000000')
 -l <LoopCnt>          (default  = 0)
 -m <WriteMode>        (default  = 'c' Channels Routine)
                           'c'    - Write Channels Routine
                           'd'    - DMA (Channel)
                           'D'    - DMA (FIFO)
                           'p'    - PIO (Channel)
                           'P'    - PIO (FIFO)
 -n <NumSamples>       (Number of Samples, default = 512)
 -p                    (program board converters)
 -R                    (Reset board and exit)
 -S                    (Synchronize Channels, default = Immediate)
 -t <Timeout>          (default  = 30)
 -T <TestBus>          (default  = No Change
                           'b'    - Calibration Bus Control
                           'o'    - Open
                           'r'    - 5 Volt Reference
 -u                    (abort test on underflow)
 -v <output volts>     (default  = '10.000000')
```

```
   -V <OutputRange>       (default  = 'b10' Bipolar 10 volts)
                          'u5'  - Unipolar 5 volts  (  +0 --> +5  )
                          'u10' - Unipolar 10 volts (  +0 --> +10 )
                          'b5'  - Bipolar 5 volts   (  -5 --> +5  )
                          'b10' - Bipolar 10 volts  ( -10 --> +10 )
                          'b2.5' - Bipolar 2.5 volts (-2.5 --> +2.5)
   -w <WaveType>          (default = 'c' Constant Voltage)
                          'c'   - Constant Voltage
                          'u'   - Saw Wave (up)
                          'd'   - Saw Wave (down)
                          's'   - Sine Wave
                          'x'   - Square Wave
                          'X'   - Square Wave - Alternate Sample
                          'y'   - Step Wave (down)
                          'z'   - Step Wave (up)
                          't'   - Triangle Wave
                          'w'   - All Wave
                                  (Sine/Square/StepUp/Triangle/StepDown)
   -X [s,p,e]             (Board External Clock Output Selection)
                          's'   - software clock output
                          'p'   - PLL clock output
                          's'   - External clock output
   -Z <CalBusCtrl>        (default = No Change
                          'g'   - Ground
                          'n'   - Negative
                          'o'   - Open
                          'p'   - Positive
                          '0..31'- Channel Number
```

Example display:

```
Device Name   : /dev/ccuraocc0
Board Serial No: 12345678 (0x00bc614e)

Board Converters are Reset: Programming card
#### Programming Board ####


==========================================================
Programmed PLL Info...
  Desired Clock Frequency   =  0.4000000000 MHz
  Programmed Clock Frequency =  0.4000000000 MHz
  Frequency Delta           = -0.0000000001 Hz
  Synth Error               =  0.0000000000 ppm

  Requested Sample Rate     = 400000.0000000000 SPS
  Actual Sample Rate        = 399999.9999999999 SPS
  Sample Rate Delta         = -0.0000000001 SPS  (-0.000000% error)
  Clock Divider             = 1 (0x00001)
==========================================================
Write Mode: Programmed I/O - Library Channel Routine
Generating a continuous Sine Wave on selected channels: <CTRL-C> to abort
Voltage Selection: 10.000000, Channel Mask Selection: 0xffffffff
8.930 usec/write: 5.09 msec period, 196.46 Hz
```

3.2.6 lib/ccuraocc_smp_affinity

This test provides a useful mechanism to display or set the IRQ to specific set of CPUs. This is useful when we want to make sure that the driver interrupts are not being interfered with other CPU activity.

```
Usage: ./ccuraocc_smp_affinity [-b Board] [-c CpuMask]
 -b Board           (Board number -- default is 0)
 -c CpuMask         (CPU mask in HEX -- default is none)
```

```
 e.g. ./ccuraocc_smp_affinity          (display IRQ CPU mask for selected board)
      ./ccuraocc_smp_affinity -c2      (set IRQ CPU for cpu 1)
      ./ccuraocc_smp_affinity -c4      (set IRQ CPU for cpu 2)
      ./ccuraocc_smp_affinity -c0x8    (set IRQ CPU for cpu 3)
      ./ccuraocc_smp_affinity -cE2     (set IRQ CPU for cpu 1,5,6,7)
```

Example display:

./ccuraocc_smp_affinity
(IRQ57) fc user f8 actual

./ccuraocc_smp_affinity -b1 -c8
(IRQ57) 08 user 08 actual

3.2.7 lib/ccuraocc_sshot

This is a simple program that performs immediate writes to channels in various modes.

```
Usage: ./ccuraocc_sshot [-A] [-b board] [-l loopcnt] [-m mode] [-v volts]
 -A                     (autocal - def=no autocal)
 -b <board>             (default = 0)
 -l <#>                 (specify loop count - def=1000000)
 -md                    (User DMA write mode [CHANNEL])
 -mD                    (Driver DMA write mode [CHANNEL])
 -mp                    (User PIO write mode [CHANNEL])
 -mP                    (Driver PIO write mode [CHANNEL])
 -v <volts>             (default = '10.000000')
```

Example display:

```
Device Name    : /dev/ccuraocc0
Board Serial No: 12345678 (0x00bc614e)
local_ptr      : 0x7ffff7ffc000
config_ptr     : 0x7ffff7ffb800

Write Mode: Driver DMA Channel
      0: delta: 10.992000 (min/max/av 10.770000/14.722000/10.963127

 0: 0ffff  9.999847    1: 0fffd  9.999542    2: 10002 10.000305    3: 10003 10.000458
 4: 0ffff  9.999847    5: 0fffd  9.999542    6: 10001 10.000153    7: 10006 10.000916
 8: 0ffff  9.999847    9: 0fffd  9.999542   10: 0ffff  9.999847   11: 10004 10.000610
12: 0ffff  9.999847   13: 10002 10.000305   14: 10002 10.000305   15: 10004 10.000610
16: 0ffff  9.999847   17: 10006 10.000916   18: 10003 10.000458   19: 10003 10.000458
20: 10002 10.000305   21: 10003 10.000458   22: 0ffff  9.999847   23: 10005 10.000763
24: 10002 10.000305   25: 10005 10.000763   26: 0ffff  9.999847   27: 10003 10.000458
28: 10001 10.000153   29: 10001 10.000153   30: 10002 10.000305   31: 10003 10.000458
```

3.2.8 lib/ccuraocc_tst_lib

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

```
Usage: ccuraocc_tst_lib [-b board]
```

Example display:

```
Device Name: /dev/ccuraocc0
  01 = Abort DMA                       02 = Clear Driver Error
  03 = Clear Library Error             04 = Display BOARD Registers
  05 = Display CONFIG Registers        06 = Get Board CSR
  07 = Get Board Information           08 = Get Channel Selection
  09 = Get Driver Error                10 = Get Driver Information
  11 = Get Driver Read Mode            12 = Get Driver Write Mode
  13 = Get Fifo Driver Threshold       14 = Get Fifo Information
  15 = Get Library Error               16 = Get Mapped Config Pointer
  17 = Get Mapped Driver/Library Pointer  18 = Get Mapped Local Pointer
```

```
19 = Get Physical Memory            20 = Get Sample Rate
21 = Get Test Bus Control           22 = Get Value
23 = Initialize Board               24 = MMap Physical Memory
25 = Munmap Physical Memory         26 = Program Sample Rate
27 = Read Operation                 28 = Read Channels
29 = Read Single Channel            30 = Reset Board
31 = Reset Fifo                     32 = Select Driver Read Mode
33 = Select Driver Write Mode       34 = Set Channel Selection Mask
35 = Set Board CSR                  36 = Set Fifo Driver Threshold
37 = Set Fifo Threshold             38 = Set Test Bus Control
39 = Set Value                      40 = Stop PLL Clock
41 = Write Operation                42 = Write Single Channel
43 = Write Channels                 44 = ### CALIBRATION MENU ###
45 = ### CONVERTER MENU ###         46 = ### INTERRUPT MENU ###
47 = ### PLL MENU ###               48 = ### SERIAL PROM MENU ###

Main Selection ('h'=display menu, 'q'=quit)->
_____
Main Selection ('h'=display menu, 'q'=quit)-> 44
   Command: calibration_menu()
 01 = Dump: Calibration Regs --> File   02 = Dump: File --> Calibration Regs
 03 = Get Calibrator ADC Control        04 = Get Calibrator ADC Data
 05 = Get Calibrator ADC (ALL)          06 = Get Calibrator BUS Control
 07 = Get Calibration Channel Gain      08 = Get Calibration Channel Offset
 09 = Perform ADC Calibration           10 = Perform Auto Calibration
 11 = Perform Channel Gain Calibration  12 = Perform Channel Offset Calibration
 13 = Reset ADC Calibrator              14 = Reset Selected Channel Calibration
 15 = Set Calibrator ADC Control        16 = Set Calibrator ADC Negative Gain
 17 = Set Calibrator ADC Offset         18 = Set Calibrator ADC Positive Gain
 19 = Set Calibrator BUS Control        20 = Set Calibration Channel Gain
 21 = Set Calibration Channel Offset

Calibration Selection ('h'=display menu, 'q'=quit)->
_____
Main Selection ('h'=display menu, 'q'=quit)-> 45
   Command: converter_menu()
 01 = Get Converter Clock Divider     02 = Get Converter CSR
 03 = Get Converter Update Selection  04 = Set Converter Clock Divider
 05 = Set Converter CSR (Config Channels)06 = Set Converter Update Selection

Converter Selection ('h'=display menu, 'q'=quit)->
_____
Main Selection ('h'=display menu, 'q'=quit)-> 46
   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)-> 47
   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

PLL Selection ('h'=display menu, 'q'=quit)->
_____
Main Selection ('h'=display menu, 'q'=quit)-> 48
   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)->
_____
```

3.2.9 lib/ccuraocc_volt

This test does not require a card or any input. It is used to validate the two ccurAOCC_DataToVolts() and ccurAOCC_VoltsToData() calls for the various voltage ranges supported by the card.

```
*** Test Data to Volts ***

1FFFF: exp/rec    4.99998092651367187500/   4.99998092651367187500   data->volts:
                                                                     unipolar/twocomp, 0..5V
00001: exp/rec    2.50001907348632812500/   2.50001907348632812500
00000: exp/rec    2.50000000000000000000/   2.50000000000000000000
3FFFF: exp/rec    2.49998092651367187500/   2.49998092651367187500
20000: exp/rec    0.00000000000000000000/   0.00000000000000000000

1FFFF: exp/rec    9.99996185302734375000/   9.99996185302734375000   data->volts:
                                                                     unipolar/twocomp, 0..10V
00001: exp/rec    5.00003814697265625000/   5.00003814697265625000
00000: exp/rec    5.00000000000000000000/   5.00000000000000000000
3FFFF: exp/rec    4.99996185302734375000/   4.99996185302734375000
20000: exp/rec    0.00000000000000000000/   0.00000000000000000000

1FFFF: exp/rec    2.49998092651367187500/   2.49998092651367187500   data->volts:
                                                                     bipolar/twocomp, +/-2.5V
00001: exp/rec    0.00001907348632812500/   0.00001907348632812500
00000: exp/rec    0.00000000000000000000/   0.00000000000000000000
3FFFF: exp/rec   -0.00001907348632812500/  -0.00001907348632812500
20000: exp/rec   -2.50000000000000000000/  -2.50000000000000000000

1FFFF: exp/rec    4.99996185302734375000/   4.99996185302734375000   data->volts:
                                                                     bipolar/twocomp, +/-5V
00001: exp/rec    0.00003814697265625000/   0.00003814697265625000
00000: exp/rec    0.00000000000000000000/   0.00000000000000000000
3FFFF: exp/rec   -0.00003814697265625000/  -0.00003814697265625000
20000: exp/rec   -5.00000000000000000000/  -5.00000000000000000000

1FFFF: exp/rec    9.99992370605468750000/   9.99992370605468750000   data->volts:
                                                                     bipolar/twocomp, +/-10V
00001: exp/rec    0.00007629394531250000/   0.00007629394531250000
00000: exp/rec    0.00000000000000000000/   0.00000000000000000000
3FFFF: exp/rec   -0.00007629394531250000/  -0.00007629394531250000
20000: exp/rec  -10.00000000000000000000/ -10.00000000000000000000

3FFFF: exp/rec    4.99998092651367187500/   4.99998092651367187500   data->volts:
                                                                     unipolar/offbin, 0..5V
20001: exp/rec    2.50001907348632812500/   2.50001907348632812500
20000: exp/rec    2.50000000000000000000/   2.50000000000000000000
1FFFF: exp/rec    2.49998092651367187500/   2.49998092651367187500
00000: exp/rec    0.00000000000000000000/   0.00000000000000000000

3FFFF: exp/rec    9.99996185302734375000/   9.99996185302734375000   data->volts:
                                                                     unipolar/offbin, 0..10V
20001: exp/rec    5.00003814697265625000/   5.00003814697265625000
20000: exp/rec    5.00000000000000000000/   5.00000000000000000000
1FFFF: exp/rec    4.99996185302734375000/   4.99996185302734375000
00000: exp/rec    0.00000000000000000000/   0.00000000000000000000

3FFFF: exp/rec    2.49998092651367187500/   2.49998092651367187500   data->volts:
                                                                     bipolar/offbin, +/-2.5V
20001: exp/rec    0.00001907348632812500/   0.00001907348632812500
20000: exp/rec    0.00000000000000000000/   0.00000000000000000000
1FFFF: exp/rec   -0.00001907348632812500/  -0.00001907348632812500
00000: exp/rec   -2.50000000000000000000/  -2.50000000000000000000

3FFFF: exp/rec    4.99996185302734375000/   4.99996185302734375000   data->volts:
                                                                     bipolar/offbin, +/-5V
20001: exp/rec    0.00003814697265625000/   0.00003814697265625000
20000: exp/rec    0.00000000000000000000/   0.00000000000000000000
1FFFF: exp/rec   -0.00003814697265625000/  -0.00003814697265625000
```

```
00000: exp/rec  -5.00000000000000000000/ -5.00000000000000000000


3FFFF: exp/rec   9.99992370605468750000/  9.99992370605468750000  data->volts:
                                                                  bipolar/offbin, +/-10V
20001: exp/rec   0.00007629394531250000/  0.00007629394531250000
20000: exp/rec   0.00000000000000000000/  0.00000000000000000000
1FFFF: exp/rec  -0.00007629394531250000/ -0.00007629394531250000
00000: exp/rec -10.00000000000000000000/-10.00000000000000000000


*** Test Volts to Data ***

  5.000000: exp/rec 0x1FFFF/1FFFF   volts->data: unipolar/twocomp, 0..5V
  4.999981: exp/rec 0x1FFFF/1FFFF
  2.500019: exp/rec 0x00001/00001
  2.500000: exp/rec 0x00000/00000
  2.499981: exp/rec 0x3FFFF/3FFFF
  0.000000: exp/rec 0x20000/20000

 10.000000: exp/rec 0x1FFFF/1FFFF   volts->data: unipolar/twocomp, 0..10V
  9.999962: exp/rec 0x1FFFF/1FFFF
  5.000038: exp/rec 0x00001/00001
  5.000000: exp/rec 0x00000/00000
  4.999962: exp/rec 0x3FFFF/3FFFF
  0.000000: exp/rec 0x20000/20000

  2.500000: exp/rec 0x1FFFF/1FFFF   volts->data: bipolar/twocomp, +/-2.5V
  2.499981: exp/rec 0x1FFFF/1FFFF
  0.000019: exp/rec 0x00001/00001
  0.000000: exp/rec 0x00000/00000
 -0.000019: exp/rec 0x3FFFF/3FFFF
 -2.500000: exp/rec 0x20000/20000

  5.000000: exp/rec 0x1FFFF/1FFFF   volts->data: bipolar/twocomp, +/-5V
  4.999962: exp/rec 0x1FFFF/1FFFF
  0.000038: exp/rec 0x00001/00001
  0.000000: exp/rec 0x00000/00000
 -0.000038: exp/rec 0x3FFFF/3FFFF
 -5.000000: exp/rec 0x20000/20000

 10.999924: exp/rec 0x1FFFF/1FFFF   volts->data: bipolar/twocomp, +/-10V
  9.999924: exp/rec 0x1FFFF/1FFFF
  0.000076: exp/rec 0x00001/00001
  0.000000: exp/rec 0x00000/00000
 -0.000076: exp/rec 0x3FFFF/3FFFF
-10.000000: exp/rec 0x20000/20000

  5.000000: exp/rec 0x3FFFF/3FFFF   volts->data: unipolar/offbin, 0..5V
  4.999981: exp/rec 0x3FFFF/3FFFF
  2.500019: exp/rec 0x20001/20001
  2.500000: exp/rec 0x20000/20000
  2.499981: exp/rec 0x1FFFF/1FFFF
  0.000000: exp/rec 0x00000/00000

 10.000000: exp/rec 0x3FFFF/3FFFF   volts->data: unipolar/offbin, 0..10V
  9.999962: exp/rec 0x3FFFF/3FFFF
  5.000038: exp/rec 0x20001/20001
  5.000000: exp/rec 0x20000/20000
  4.999962: exp/rec 0x1FFFF/1FFFF
  0.000000: exp/rec 0x00000/00000

  2.500000: exp/rec 0x3FFFF/3FFFF   volts->data: bipolar/offbin, +/-2.5V
  2.499981: exp/rec 0x3FFFF/3FFFF
  0.000019: exp/rec 0x20001/20001
  0.000000: exp/rec 0x20000/20000
 -0.000019: exp/rec 0x1FFFF/1FFFF
 -2.500000: exp/rec 0x00000/00000

  5.000000: exp/rec 0x3FFFF/3FFFF   volts->data: bipolar/offbin, +/-5V
  4.999962: exp/rec 0x3FFFF/3FFFF
  0.000038: exp/rec 0x20001/20001
  0.000000: exp/rec 0x20000/20000
 -0.000038: exp/rec 0x1FFFF/1FFFF
```

```
    -5.000000: exp/rec 0x00000/00000

 10.000000: exp/rec 0x3FFFF/3FFFF  volts->data: bipolar/offbin, +/-10V
  9.999924: exp/rec 0x3FFFF/3FFFF
  0.000076: exp/rec 0x20001/20001
  0.000000: exp/rec 0x20000/20000
 -0.000076: exp/rec 0x1FFFF/1FFFF
-10.000000: exp/rec 0x00000/00000
 ******* Passed! *******
```

3.2.10 lib/Sprom/ccuraocc_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: ./ccuraocc_sprom [-b board] [-C] [-D] [-F] [-i inCalFile] [-o outCalFile]
                        [-R] [-S serialNo] [-U num] [-V VoltageRange]
 -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)
 -V <VoltageRange>   (Default = 'b10' Bipolar 10 volts)
                      'u5'   - Unipolar 5 volts  (  +0 --> +5  )
                      'u10'  - Unipolar 10 volts (  +0 --> +10 )
                      'b5'   - Bipolar 5 volts   (  -5 --> +5  )
                      'b10'  - Bipolar 10 volts  ( -10 --> +10 )
                      'b2.5' - Bipolar 2.5 volts (-2.5 --> +2.5)

Cannot use '-F' and '-U#' in same command line
   e.g.  ./ccuraocc_sprom -F -V u10 -o CalOut -> Dump Factory u10 to CalOut
         ./ccuraocc_sprom -F -V b2.5 -i CalIn -> Program Factory b2.5 sprom using
                                                 CalIn file
         ./ccuraocc_sprom -U1 -i CalIn      -> Create user checkpoint 1 using
                                                 CalIn file
         ./ccuraocc_sprom -U 2 -i.          -> Create user checkpoint 2 using
                                                 memory register
         ./ccuraocc_sprom -U2 -o CalOut     -> Dump user checkpoint 2 to CalOut
         ./ccuraocc_sprom -F -R             -> Restore memory registers using
                                                 factory settings
         ./ccuraocc_sprom -U 1 -R           -> Restore memory registers using
                                                 user checkpoint 1
```

Appendix A: Calibration

⚠️ *Warning:* Whenever auto-calibration is performed, the channel outputs will be affected. It is important that prior to calibration, any sensitive equipment be disconnected; otherwise it could result in damage to the equipment.

Several library calls are provided to assist the user in calibrating the board. Additionally, the board contains factory calibration information for each of the output voltage ranges. Users can view this information using the supplied API or the serial prom test utility *ccuraocc_sprom*. Though the API and test utility provides capability to edit and change the factory calibration, users should refrain from making any changes to it, as it will no longer reflect the factory calibration shipped with the card. Users can use the factory calibration to restore the calibration information stored for each configured channel prior to commencing a test run. The restore API will update the calibration information for all the channels based on their current voltage range. Note that the factory calibration values were obtained under specific conditions, such as temperature, that may not be the same as the user application. In most cases it will always be better to perform auto-calibration after the board is stabilized in the user environment.

Additionally, the users can perform up to two independent user controlled checkpoints where the active channel configuration and calibration information is stored in the serial prom for all the channels. At any time, the user can restore either of the two checkpoints with an API call or the serial prom test utility prior to a test run. These checkpoints will allow the user to store specific values pertaining to their calibration conditions.

Appendix B: Important Considerations

This section tries to highlight cause and effect on the behavior of the hardware and software which can assist the user in developing their applications:

- The driver allows multiple applications to open the same card concurrently, however, this is not a recommended procedure and should only be considered during debugging and testing otherwise unpredictable results can be observed.
- When the board CSR has all the converters in the reset state, changing the channel configurations or writing to the channel registers will have no effect. The user must first activate the converters prior to issuing any changes to the channel configuration or channel data registers.
- Changing the channel configuration information will have no effect on the output until data is either written to the channel registers or the samples in the FIFO are actually being output.
- Changing the channel selection mask will have immediate affect and therefore any data already in the FIFO will cause different association of samples to channels. In short, if the FIFO is outputting samples, the data appearing on the output lines could possibly belong to the wrong channel. The channel selection mask has no effect when writing to channel registers.
- If an underflow or overflow condition is detected (FIFO empty), the user must reset the FIFO to clear the status and ensure that the FIFO is empty before adding samples to the FIFO so that the hardware and software are synchronized.
- While samples are being output via the FIFO, it is possible that the users may attempt to change the sample rate. Though this may be possible, there may be an abrupt change in the samples with possibly a short period of steady samples when the clock is stopped and restarted.
- If the user changes the clock divider while the FIFO is sending data out, the output frequency will be reflected immediately on all active channels.
- In order to synchronize channels, the channel configuration registers need to have their synchronization flags set and additionally, for any data to be output, at least one of the active channels need to have the synchronize update flag set. The moment the hardware sees a channel data (either in FIFO outputting or channel register writes) with the synchronize update flag set, all channels with the synchronization flags in their channel configuration will be output simultaneously.
- It takes a finite time to write samples to the channel registers and be output to the hardware. Writing too fast to the same channel register could cause loss of samples. Users need to monitor the channel busy flag in the channel configuration register, prior to writing to the channel registers.
- This card has a channel configuration on a per channel basis, unlike other vendor cards which have a single channel configuration for all channels. This means that writing the *same* raw channel could have possibly different output results as determined by the individual channel configuration.
- The API allows the user to write to any part of the serial prom. Normally, the user should not touch the header information and the factory settings, otherwise, vital board information could be lost. They only writes to the serial prom by the user should be related to the user checkpoints.

This page intentionally left blank