

Software Interface

CCRTAICC (WC-ADS6418)

PCIe 64-Channel Analog Input Card (AICC)

<i>Driver</i>	certaicc (WC-ADS6418)	
<i>OS</i>	RedHawk (CentOS or Ubuntu based)	
<i>Vendor</i>	Concurrent Real-Time	
<i>Hardware</i>	PCIe 64-Channel Analog Input Card (CP-ADS6418)	
<i>Date</i>	June 12 th , 2019	Rev 2019.1



This page intentionally left blank

Table of Contents

1. INTRODUCTION	8
1.1 Related Documents	8
2. SOFTWARE SUPPORT	8
2.1 Direct Driver Access.....	8
2.1.1 open(2) system call	8
2.1.2 ioctl(2) system call.....	9
2.1.3 mmap(2) system call.....	11
2.1.4 read(2) system call	12
2.1.5 write(2) system call.....	12
2.2 Application Program Interface (API) Access	12
2.2.1 ccrtAICC_Abort_DMA().....	16
2.2.2 ccrtAICC_ADC_Activate()	16
2.2.3 ccrtAICC_ADC_Get_CSR().....	16
2.2.4 ccrtAICC_ADC_Get_Driver_Read_Mode()	17
2.2.5 ccrtAICC_ADC_Get_Fifo_Channel_Select().....	18
2.2.6 ccrtAICC_ADC_Get_Fifo_Info().....	18
2.2.7 ccrtAICC_ADC_Get_Fifo_Threshold()	19
2.2.8 ccrtAICC_ADC_Get_Input_Control().....	20
2.2.9 ccrtAICC_ADC_Get_Negative_Cal()	20
2.2.10 ccrtAICC_ADC_Get_Offset_Cal().....	20
2.2.11 ccrtAICC_ADC_Get_Positive_Cal().....	21
2.2.12 ccrtAICC_ADC_Perform_Auto_Calibration()	21
2.2.13 ccrtAICC_ADC_Perform_External_Negative_Calibration()	22
2.2.14 ccrtAICC_ADC_Perform_External_Offset_Calibration().....	23
2.2.15 ccrtAICC_ADC_Perform_External_Positive_Calibration().....	24
2.2.16 ccrtAICC_ADC_Perform_Negative_Calibration().....	25
2.2.17 ccrtAICC_ADC_Perform_Offset_Calibration()	26
2.2.18 ccrtAICC_ADC_Perform_Positive_Calibration()	27
2.2.19 ccrtAICC_ADC_Read_Channels().....	28
2.2.20 ccrtAICC_ADC_Read_Channels_Calibration()	29
2.2.21 ccrtAICC_ADC_Reset_Calibration()	30
2.2.22 ccrtAICC_ADC_Reset_Fifo()	30
2.2.23 ccrtAICC_ADC_Set_CSR()	31
2.2.24 ccrtAICC_ADC_Set_Driver_Read_Mode()	32
2.2.25 ccrtAICC_ADC_Set_Fifo_Channel_Select()	32
2.2.26 ccrtAICC_ADC_Set_Fifo_Threshold()	33
2.2.27 ccrtAICC_ADC_Set_Input_Control()	33
2.2.28 ccrtAICC_ADC_Set_Negative_Cal().....	34
2.2.29 ccrtAICC_ADC_Set_Offset_Cal()	34
2.2.30 ccrtAICC_ADC_Set_Positive_Cal()	35
2.2.31 ccrtAICC_ADC_Write_Channels_Calibration()	35
2.2.32 ccrtAICC_Add_Irq().....	36
2.2.33 ccrtAICC_BoardExpirationTimeRemaining().....	36
2.2.34 ccrtAICC_Clear_Cable_Fault()	37
2.2.35 ccrtAICC_Clear_Driver_Error().....	38
2.2.36 ccrtAICC_Clear_Lib_Error().....	38
2.2.37 ccrtAICC_Clock_Generator_Disable_Outputs()	38
2.2.38 ccrtAICC_Clock_Generator_Enable_Outputs()	39
2.2.39 ccrtAICC_Clock_Generator_Soft_Reset()	39
2.2.40 ccrtAICC_Clock_Get_Generator_CSR().....	39

2.2.41	ccrtAICC_Clock_Get_Generator_Info()	40
2.2.42	ccrtAICC_Clock_Get_Generator_Input_Clock_Enable()	43
2.2.43	ccrtAICC_Clock_Get_Generator_Input_Clock_Select()	43
2.2.44	ccrtAICC_Clock_Get_Generator_Input_Clock_Status()	44
2.2.45	ccrtAICC_Clock_Get_Generator_M_Divider()	44
2.2.46	ccrtAICC_Clock_Get_Generator_N_Divider()	45
2.2.47	ccrtAICC_Clock_Get_Generator_Output_Config()	45
2.2.48	ccrtAICC_Clock_Get_Generator_Output_Format()	46
2.2.49	ccrtAICC_Clock_Get_Generator_Output_Mode()	47
2.2.50	ccrtAICC_Clock_Get_Generator_Output_Mux()	47
2.2.51	ccrtAICC_Clock_Get_Generator_P_Divider()	48
2.2.52	ccrtAICC_Clock_Get_Generator_P_Divider_Enable()	49
2.2.53	ccrtAICC_Clock_Get_Generator_R_Divider()	49
2.2.54	ccrtAICC_Clock_Get_Generator_Revision()	50
2.2.55	ccrtAICC_Clock_Get_Generator_Value()	50
2.2.56	ccrtAICC_Clock_Get_Generator_Voltage_Select()	51
2.2.57	ccrtAICC_Clock_Get_Generator_Zero_Delay()	51
2.2.58	ccrtAICC_Clock_ReturnOutputFrequency()	51
2.2.59	ccrtAICC_Clock_Set_Generator_CSR()	52
2.2.60	ccrtAICC_Clock_Set_Generator_Input_Clock_Enable()	52
2.2.61	ccrtAICC_Clock_Set_Generator_Input_Clock_Select()	53
2.2.62	ccrtAICC_Clock_Set_Generator_M_Divider()	53
2.2.63	ccrtAICC_Clock_Set_Generator_N_Divider()	54
2.2.64	ccrtAICC_Clock_Set_Generator_Output_Config()	55
2.2.65	ccrtAICC_Clock_Set_Generator_Output_Format()	55
2.2.66	ccrtAICC_Clock_Set_Generator_Output_Mode()	56
2.2.67	ccrtAICC_Clock_Set_Generator_Output_Mux()	57
2.2.68	ccrtAICC_Clock_Set_Generator_P_Divider()	58
2.2.69	ccrtAICC_Clock_Set_Generator_P_Divider_Enable()	58
2.2.70	ccrtAICC_Clock_Set_Generator_R_Divider()	59
2.2.71	ccrtAICC_Clock_Set_Generator_Value()	59
2.2.72	ccrtAICC_Clock_Set_Generator_Voltage_Select()	60
2.2.73	ccrtAICC_Clock_Set_Generator_Zero_Delay()	60
2.2.74	ccrtAICC_Close()	61
2.2.75	ccrtAICC_Compute_All_Output_Clocks()	61
2.2.76	ccrtAICC_Convert_Physmem2avmm_Address()	62
2.2.77	ccrtAICC_Create_UserProcess()	62
2.2.78	ccrtAICC_DataToVolts()	63
2.2.79	ccrtAICC_Destroy_AllUserProcess()	64
2.2.80	ccrtAICC_Destroy_UserProcess()	64
2.2.81	ccrtAICC_Disable_Pci_Interrupts()	65
2.2.82	ccrtAICC_DMA_Configure()	65
2.2.83	ccrtAICC_DMA_Fire()	66
2.2.84	ccrtAICC_Enable_Pci_Interrupts()	66
2.2.85	ccrtAICC_Fast_Memcpy()	67
2.2.86	ccrtAICC_Fast_Memcpy_Unlocked()	67
2.2.87	ccrtAICC_Fast_Memcpy_Unlocked_FIFO()	68
2.2.88	ccrtAICC_Fraction_To_Hex()	68
2.2.89	ccrtAICC_Get_All_Boards_Driver_Info()	68
2.2.90	ccrtAICC_Get_Board_CSR()	71
2.2.91	ccrtAICC_Get_Board_Info()	71
2.2.92	ccrtAICC_Get_Cable_Fault_CSR()	72
2.2.93	ccrtAICC_Get_Calibration_CSR()	73
2.2.94	ccrtAICC_Get_Driver_Error()	73
2.2.95	ccrtAICC_Get_Driver_Info()	74
2.2.96	ccrtAICC_Get_External_Clock_CSR()	77

2.2.97	ccrtAICC_Get_Interrupt_Status()	78
2.2.98	ccrtAICC_Get_Interrupt_Timeout_Seconds()	78
2.2.99	ccrtAICC_Get_Lib_Error()	78
2.2.100	ccrtAICC_Get_Library_Info()	80
2.2.101	ccrtAICC_Get_Mapped_Config_Ptr()	81
2.2.102	ccrtAICC_Get_Mapped_Driver_Library_Ptr()	81
2.2.103	ccrtAICC_Get_Mapped_Local_Ptr()	82
2.2.104	ccrtAICC_Get_Open_File_Descriptor()	82
2.2.105	ccrtAICC_Get_Physical_Memory()	83
2.2.106	ccrtAICC_Get_RunCount_UserProcess()	83
2.2.107	ccrtAICC_Get_TestBus_Control()	84
2.2.108	ccrtAICC_Get_Value()	84
2.2.109	ccrtAICC_Hex_To_Fraction()	84
2.2.110	ccrtAICC_Identify_Board()	85
2.2.111	ccrtAICC_Initialize_Board()	85
2.2.112	ccrtAICC_MMap_Physical_Memory()	85
2.2.113	ccrtAICC_MsgDma_Configure_ADC_Fifo()	86
2.2.114	ccrtAICC_MsgDma_Configure_Descriptor()	87
2.2.115	ccrtAICC_MsgDma_Configure_Single()	88
2.2.116	ccrtAICC_MsgDma_Fire()	89
2.2.117	ccrtAICC_MsgDma_Fire_ADC_Fifo()	90
2.2.118	ccrtAICC_MsgDma_Fire_Single()	90
2.2.119	ccrtAICC_MsgDma_Free_Descriptor()	90
2.2.120	ccrtAICC_MsgDma_Get_Descriptor()	91
2.2.121	ccrtAICC_MsgDma_Get_Dispatcher_CSR()	91
2.2.122	ccrtAICC_MsgDma_Get_Prefetcher_CSR()	92
2.2.123	ccrtAICC_MsgDma_Release()	93
2.2.124	ccrtAICC_MsgDma_Seize()	93
2.2.125	ccrtAICC_MsgDma_Setup()	94
2.2.126	ccrtAICC_Munmap_Physical_Memory()	94
2.2.127	ccrtAICC_NanoDelay()	95
2.2.128	ccrtAICC_Open()	95
2.2.129	ccrtAICC_Pause_UserProcess()	96
2.2.130	ccrtAICC_Program_All_Output_Clocks()	96
2.2.131	ccrtAICC_Read()	98
2.2.132	ccrtAICC_Reload_Firmware()	98
2.2.133	ccrtAICC_Remove_Irq()	99
2.2.134	ccrtAICC_Reset_Board()	99
2.2.135	ccrtAICC_Reset_Clock()	100
2.2.136	ccrtAICC_Resume_UserProcess()	100
2.2.137	ccrtAICC_Return_Board_Info_Description()	100
2.2.138	ccrtAICC_SDRAM_Activate() **	100
2.2.139	ccrtAICC_SDRAM_Get_CSR() **	101
2.2.140	ccrtAICC_SDRAM_Read() **	102
2.2.141	ccrtAICC_SDRAM_Set_CSR() **	102
2.2.142	ccrtAICC_SDRAM_Write() **	103
2.2.143	ccrtAICC_Set_Board_CSR()	103
2.2.144	ccrtAICC_Set_Calibration_CSR()	104
2.2.145	ccrtAICC_Set_External_Clock_CSR()	104
2.2.146	ccrtAICC_Set_Interrupt_Status()	105
2.2.147	ccrtAICC_Set_Interrupt_Timeout_Seconds()	105
2.2.148	ccrtAICC_Set_TestBus_Control()	105
2.2.149	ccrtAICC_Set_Value()	106
2.2.150	ccrtAICC_SPROM_Read() **	106
2.2.151	ccrtAICC_SPROM_Read_Item() **	107
2.2.152	ccrtAICC_SPROM_Write() **	107

2.2.153	ccrtAICC_SPROM_Write_Item() **	108
2.2.154	ccrtAICC_SPROM_Write_Override() **	108
2.2.155	ccrtAICC_Transfer_Data()	109
2.2.156	ccrtAICC_Update_Clock_Generator_Divider()	110
2.2.157	ccrtAICC_UserProcess_Command()	111
2.2.158	ccrtAICC_VoltsToData()	111
2.2.159	ccrtAICC_Wait_For_Interrupt()	112
2.2.160	ccrtAICC_Write()	112
3.	TEST PROGRAMS	114
3.1	Direct Driver Access Example Tests	114
3.1.1	ccrtaicc_disp	114
3.1.2	ccrtaicc_dump	115
3.1.3	ccrtaicc_rdreg	118
3.1.4	ccrtaicc_reg	119
3.1.5	ccrtaicc_regedit	129
3.1.6	ccrtaicc_tst	130
3.1.7	ccrtaicc_wreg	130
3.1.8	Flash/ccrtaicc_flash	131
3.1.9	Flash/ccrtaicc_label	132
3.1.10	Flash/ccrtaicc_dump_license	133
3.2	Application Program Interface (API) Access Example Tests	134
3.2.1	lib/ccrtaicc_adc	134
3.2.2	lib/ccrtaicc_adc_calibrate	138
3.2.3	lib/ccrtaicc_adc_fifo	140
3.2.4	lib/ccrtaicc_adc_sps	143
3.2.5	lib/ccrtaicc_check_bus	147
3.2.6	lib/ccrtaicc_clock	148
3.2.7	lib/ccrtaicc_disp	149
3.2.8	lib/ccrtaicc_dma	150
3.2.9	lib/ccrtaicc_example	151
3.2.10	lib/ccrtaicc_expires	152
3.2.11	lib/ccrtaicc_identify	153
3.2.12	lib/ccrtaicc_info	153
3.2.13	lib/ccrtaicc_msgdma	155
3.2.14	lib/ccrtaicc_msgdma_info	156
3.2.15	lib/ccrtaicc_smp_affinity	157
3.2.16	lib/ccrtaicc_transfer	157
3.2.17	lib/ccrtaicc_tst_lib	158

This page intentionally left blank

1. Introduction

This document provides the software interface to the *ccrtaicc* driver which communicates with the Concurrent Real-Time PCI Express 64-Channel Analog Input Card (AICC). For additional information on programming, please refer to the *Concurrent Real-Time PCI Express 64-Channel Analog Input Card (AICC) Design Specification* document.

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

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

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

Though API calls exist for the following, the board does not support them:

- SDRAM
- SPROM

1.1 Related Documents

- PCI Express 64-Channel Analog Input Card Driver Installation on RedHawk Release Notes by Concurrent Real-Time.
- PCI Express 64-Channel Analog Input Card (AICC) Design Specification (No. 0610108) by Concurrent Real-Time.

2. Software Support

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

2.1 Direct Driver Access

2.1.1 *open(2)* system call

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

```
int    fp;
fp = open("/dev/ccrtaicc0", O_RDWR);
```

The file pointer '*fp*' is then used as an argument to other system calls. The user can also supply the *O_NONBLOCK* flag if the user does not wish to block waiting for reads to complete. In that case, if the read is not satisfied, the call will fail. The device name specified is of the format *"/dev/ccrtaicc<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 between applications is performed external to the driver API.

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

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_CCRTAICC_ABORT_DMA
IOCTL_CCRTAICC_ADD_IRQ
IOCTL_CCRTAICC_DISABLE_PCI_INTERRUPTS
IOCTL_CCRTAICC_ENABLE_PCI_INTERRUPTS
IOCTL_CCRTAICC_GET_DRIVER_ERROR
IOCTL_CCRTAICC_GET_DRIVER_INFO
IOCTL_CCRTAICC_GET_PHYSICAL_MEMORY
IOCTL_CCRTAICC_GET_ADC_READ_MODE
IOCTL_CCRTAICC_INIT_BOARD
IOCTL_CCRTAICC_INTERRUPT_TIMEOUT_SECONDS
IOCTL_CCRTAICC_MMAP_SELECT
IOCTL_CCRTAICC_NO_COMMAND
IOCTL_CCRTAICC_PCI_CONFIG_REGISTERS
IOCTL_CCRTAICC_REMOVE_IRQ
IOCTL_CCRTAICC_RESET_BOARD
IOCTL_CCRTAICC_SELECT_ADC_READ_MODE
IOCTL_CCRTAICC_WAIT_FOR_INTERRUPT
IOCTL_CCRTAICC_RELOAD_FIRMWARE
IOCTL_CCRTAICC_GET_ALL_BOARDS_DRIVER_INFO
```

IOCTL_CCRTAICC_ABORT_DMA: This *ioctl* does not have any arguments. Its purpose is to abort any DMA already in progress..

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

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

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

IOCTL_CCRTAICC_GET_DRIVER_ERROR: The argument supplied to this *ioctl* is a pointer to the *ccrtaiicc_user_error_t* structure. Information on the structure is located in the *ccrtaiicc_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 *CCRTAICC_SUCCESS*.

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

IOCTL_CCRTAICC_GET_PHYSICAL_MEMORY: The argument supplied to this *ioctl* is a pointer to the *ccrtaiicc_user_phys_mem_t* structure. Information on the structure is located in the *ccrtaiicc_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_CCRTAICC_GET_ADC_READ_MODE: The argument supplied to this *ioctl* is a pointer to an *unsigned long int*. The value returned will be one of the ADC read modes as defined by the *enum_ccrtaiicc_driver_ADC_read_mode_t* located in the *ccrtaiicc_user.h* include file.

IOCTL_CCRTAICC_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_CCRTAICC_RESET_BOARD* call.

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

IOCTL_CCRTAICC_MMAP_SELECT: The argument to this *ioctl* is a pointer to the *ccrtaiicc_mmap_select_t* structure. Information on the structure is located in the *ccrtaiicc_user.h* include file. This call needs to be made prior to the *mmap(2)* system call so as to direct the *mmap(2)* call to perform the requested mapping specified by this *ioctl*. The four possible mappings that are performed by the driver are to *mmap* the local register space (*CCRTAICC_SELECT_LOCAL_MMAP*), the configuration register space (*CCRTAICC_SELECT_CONFIG_MMAP*) the physical memory (*CCRTAICC_SELECT_PHYS_MEM_MMAP*) that is created by the *mmap(2)* system call and the driver/library mapping (*CCRTAICC_SELECT_DRIVER_LIBRARY_MMAP*).

IOCTL_CCRTAICC_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_CCRTAICC_PCI_CONFIG_REGISTERS: The argument supplied to this *ioctl* is a pointer to the *ccrtaiicc_pci_config_reg_addr_mapping_t* structure whose definition is located in the *ccrtaiicc_user.h* include file.

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

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

IOCTL_CCRTAICC_WAIT_FOR_INTERRUPT: The argument to this *ioctl* is a pointer to the *ccrtaiicc_driver_int_t* structure. Information on the structure is located in the *ccrtaiicc_user.h* include file. The user can wait for a DMA or Analog signal complete interrupt. If a time out value greater than zero is specified, the call will time out after the specified seconds, otherwise it will not time out.

IOCTL_CCRTAICC_RELOAD_FIRMWARE: This *ioctl* does not have any arguments. This call performs a reload of the latest firmware that was loaded into the board. Typically, this is used after a new firmware has been installed. It eliminates the need to reboot the kernel after a firmware update.

IOCTL_CCRTAICC_GET_ALL_BOARDS_DRIVER_INFO: The argument to this *ioctl* is a pointer to *ccrtaiicc_all_boards_driver_info*. It provides the ability to supply all driver information for all the *ccrtaiicc* cards in the system to 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, create and map a physical memory that can be used for user DMA or driver/library structure mapping. Prior to making this system call, the user needs to issue the *ioctl(2)* system call with the *IOCTL_CCRTAICC_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;
ccrtaiicc_local_ctrl_data_t *local_ptr;
ccrtaiicc_mmap_select_t mmap_select;
unsigned long mmap_local_size;

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

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

local_ptr = (ccrtaiicc_local_ctrl_data_t *)munmap_local_ptr;
local_ptr = (ccrtaiicc_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

This system call currently supports ADC programmed I/O reads of channel registers and FIFO. The option selected is determined by the `ccrtAICC_ADC_Set_Driver_Read_Mode()` call.

CCRTAICC_ADC_PIO_CHANNEL: Perform .channel registers programmed I/O reads.

CCRTAICC_ADC_PIO_FIFO: Perform FIFO reads using programmed I/O.

2.1.5 write(2) system call

This card does not support this call.

2.2 Application Program Interface (API) Access

The following APIs are the recommended method of communicating with the board for most users:

```
ccrtAICC_Abort_DMA ()
ccrtAICC_ADC_Activate ()
ccrtAICC_ADC_Get_CSR ()
ccrtAICC_ADC_Get_Driver_Read_Mode ()
ccrtAICC_ADC_Get_Fifo_Channel_Select ()
ccrtAICC_ADC_Get_Fifo_Info ()
ccrtAICC_ADC_Get_Fifo_Threshold ()
ccrtAICC_ADC_Get_Input_Control ()
ccrtAICC_ADC_Get_Negative_Cal ()
ccrtAICC_ADC_Get_Offset_Cal ()
ccrtAICC_ADC_Get_Positive_Cal ()
ccrtAICC_ADC_Perform_Auto_Calibration ()
ccrtAICC_ADC_Perform_External_Negative_Calibration ()
ccrtAICC_ADC_Perform_External_Offset_Calibration ()
ccrtAICC_ADC_Perform_External_Positive_Calibration ()
ccrtAICC_ADC_Perform_Negative_Calibration ()
ccrtAICC_ADC_Perform_Offset_Calibration ()
ccrtAICC_ADC_Perform_Positive_Calibration ()
ccrtAICC_ADC_Read_Channels ()
ccrtAICC_ADC_Read_Channels_Calibration ()
ccrtAICC_ADC_Reset_Calibration ()
ccrtAICC_ADC_Reset_Fifo ()
ccrtAICC_ADC_Set_CSR ()
ccrtAICC_ADC_Set_Driver_Read_Mode ()
ccrtAICC_ADC_Set_Fifo_Channel_Select ()
ccrtAICC_ADC_Set_Fifo_Threshold ()
ccrtAICC_ADC_Set_Input_Control ()
ccrtAICC_ADC_Set_Negative_Cal ()
ccrtAICC_ADC_Set_Offset_Cal ()
ccrtAICC_ADC_Set_Positive_Cal ()
ccrtAICC_ADC_Write_Channels_Calibration ()
ccrtAICC_Add_Irq ()
ccrtAICC_BoardExpirationTimeRemaining ()
ccrtAICC_Clear_Cable_Fault ()
ccrtAICC_Clear_Driver_Error ()
ccrtAICC_Clear_Lib_Error ()
ccrtAICC_Clock_Generator_Disable_Outputs ()
ccrtAICC_Clock_Generator_Enable_Outputs ()
ccrtAICC_Clock_Generator_Soft_Reset ()
ccrtAICC_Clock_Get_Generator_CSR ()
ccrtAICC_Clock_Get_Generator_Info ()
ccrtAICC_Clock_Get_Generator_Input_Clock_Enable ()
```

```

ccrtAICC_Clock_Get_Generator_Input_Clock_Select()
ccrtAICC_Clock_Get_Generator_Input_Clock_Status()
ccrtAICC_Clock_Get_Generator_M_Divider()
ccrtAICC_Clock_Get_Generator_N_Divider()
ccrtAICC_Clock_Get_Generator_Output_Config()
ccrtAICC_Clock_Get_Generator_Output_Format()
ccrtAICC_Clock_Get_Generator_Output_Mode()
ccrtAICC_Clock_Get_Generator_Output_Mux()
ccrtAICC_Clock_Get_Generator_P_Divider()
ccrtAICC_Clock_Get_Generator_P_Divider_Enable()
ccrtAICC_Clock_Get_Generator_R_Divider()
ccrtAICC_Clock_Get_Generator_Revision()
ccrtAICC_Clock_Get_Generator_Value()
ccrtAICC_Clock_Get_Generator_Voltage_Select()
ccrtAICC_Clock_Get_Generator_Zero_Delay()
ccrtAICC_Clock_ReturnOutputFrequency()
ccrtAICC_Clock_Set_Generator_CSR()
ccrtAICC_Clock_Set_Generator_Input_Clock_Enable()
ccrtAICC_Clock_Set_Generator_Input_Clock_Select()
ccrtAICC_Clock_Set_Generator_M_Divider()
ccrtAICC_Clock_Set_Generator_N_Divider()
ccrtAICC_Clock_Set_Generator_Output_Config()
ccrtAICC_Clock_Set_Generator_Output_Format()
ccrtAICC_Clock_Set_Generator_Output_Mode()
ccrtAICC_Clock_Set_Generator_Output_Mux()
ccrtAICC_Clock_Set_Generator_P_Divider()
ccrtAICC_Clock_Set_Generator_P_Divider_Enable()
ccrtAICC_Clock_Set_Generator_R_Divider()
ccrtAICC_Clock_Set_Generator_Value()
ccrtAICC_Clock_Set_Generator_Voltage_Select()
ccrtAICC_Clock_Set_Generator_Zero_Delay()
ccrtAICC_Close()
ccrtAICC_Compute_All_Output_Clocks()
ccrtAICC_Convert_Physmem2avmm_Address()
ccrtAICC_Create_UserProcess()
ccrtAICC_DataToVolts()
ccrtAICC_Destroy_AllUserProcess()
ccrtAICC_Destroy_UserProcess()
ccrtAICC_Disable_Pci_Interrupts()
ccrtAICC_DMA_Configure()
ccrtAICC_DMA_Fire()
ccrtAICC_Enable_Pci_Interrupts()
ccrtAICC_Fast_Memcpy()
ccrtAICC_Fast_Memcpy_Unlocked()
ccrtAICC_Fast_Memcpy_Unlocked_FIFO()
ccrtAICC_Fraction_To_Hex()
ccrtAICC_Get_All_Boards_Driver_Info()
ccrtAICC_Get_Board_CSR()
ccrtAICC_Get_Board_Info()
ccrtAICC_Get_Cable_Fault_CSR()
ccrtAICC_Get_Calibration_CSR()
ccrtAICC_Get_Driver_Error()
ccrtAICC_Get_Driver_Info()
ccrtAICC_Get_External_Clock_CSR()
ccrtAICC_Get_Interrupt_Status()
ccrtAICC_Get_Interrupt_Timeout_Seconds()
ccrtAICC_Get_Lib_Error()

```

```

ccrtAICC_Get_Library_Info()
ccrtAICC_Get_Mapped_Config_Ptr()
ccrtAICC_Get_Mapped_Driver_Library_Ptr()
ccrtAICC_Get_Mapped_Local_Ptr()
ccrtAICC_Get_Open_File_Descriptor()
ccrtAICC_Get_Physical_Memory()
ccrtAICC_Get_RunCount_UserProcess()
ccrtAICC_Get_TestBus_Control()
ccrtAICC_Get_Value()
ccrtAICC_Hex_To_Fraction()
ccrtAICC_Identify_Board()
ccrtAICC_Initialize_Board()
ccrtAICC_MMap_Physical_Memory()
ccrtAICC_MsgDma_Configure_ADC_Fifo()
ccrtAICC_MsgDma_Configure_Descriptor()
ccrtAICC_MsgDma_Configure_Single()
ccrtAICC_MsgDma_Fire()
ccrtAICC_MsgDma_Fire_ADC_Fifo()
ccrtAICC_MsgDma_Fire_Single()
ccrtAICC_MsgDma_Free_Descriptor()
ccrtAICC_MsgDma_Get_Descriptor()
ccrtAICC_MsgDma_Get_Dispatcher_CSR()
ccrtAICC_MsgDma_Get_Prefetcher_CSR()
ccrtAICC_MsgDma_Release()
ccrtAICC_MsgDma_Seize()
ccrtAICC_MsgDma_Setup()
ccrtAICC_Munmap_Physical_Memory()
ccrtAICC_NanoDelay()
ccrtAICC_Open()
ccrtAICC_Pause_UserProcess()
ccrtAICC_Program_All_Output_Clocks()
ccrtAICC_Read()
ccrtAICC_Reload_Firmware()
ccrtAICC_Remove_Irq()
ccrtAICC_Reset_Board()
ccrtAICC_Reset_Clock()
ccrtAICC_Resume_UserProcess()
ccrtAICC_Return_Board_Info_Description()
ccrtAICC_SDRAM_Activate() **
ccrtAICC_SDRAM_Get_CSR() **
ccrtAICC_SDRAM_Read() **
ccrtAICC_SDRAM_Set_CSR() **
ccrtAICC_SDRAM_Write() **
ccrtAICC_Set_Board_CSR()
ccrtAICC_Set_Calibration_CSR()
ccrtAICC_Set_External_Clock_CSR()
ccrtAICC_Set_Interrupt_Status()
ccrtAICC_Set_Interrupt_Timeout_Seconds()
ccrtAICC_Set_TestBus_Control()
ccrtAICC_Set_Value()
ccrtAICC_SPROM_Read() **
ccrtAICC_SPROM_Read_Item() **
ccrtAICC_SPROM_Write() **
ccrtAICC_SPROM_Write_Item() **
ccrtAICC_SPROM_Write_Override() **
ccrtAICC_Transfer_Data()
ccrtAICC_Update_Clock_Generator_Divider()

```

```
ccrtAICC_UserProcess_Command()  
ccrtAICC_VoltsToData()  
ccrtAICC_Wait_For_Interrupt()  
ccrtAICC_Write()
```

2.2.1 ccrtAICC_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.

```

/*****
  _ccrtaicc_lib_error_number_t ccrtAICC_Abort_DMA(void *Handle)

  Description: Abort any DMA in progress

  Input:   void *Handle           (Handle pointer)
  Output:  none
  Return:  _ccrtaicc_lib_error_number_t
           # CCRTAICC_LIB_NO_ERROR           (successful)
           # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
           # CCRTAICC_LIB_NOT_OPEN         (device not open)
           # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
           # CCRTAICC_LIB_IOCTL_FAILED     (driver ioctl call failed)
*****/
```

2.2.2 ccrtAICC_ADC_Activate()

This call must be the first call to activate the ADC. Without activation, all other calls to the ADC will fail. The user can also use this call to return the current state of the ADC without any change by specifying a pointer to `current_state` and setting `activate` to `CCRTAICC_ADC_ALL_ENABLE_DO_NOT_CHANGE`. If the ADC is already active and the user issues a `CCRTAICC_ADC_ALL_ENABLE`, no additional activation will be performed. To cause the ADC to go through a full reset, the user needs to issue the `CCRTAICC_ADC_ALL_RESET` which will cause the ADC to disable and then re-enable, setting all its ADC values to a default state. ADC calibration data will *not* be reset.

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_ADC_Activate (void *Handle,
                        _ccrtaicc_adc_all_enable_t activate,
                        _ccrtaicc_adc_all_enable_t *current_state)

  Description: Activate/DeActivate ADC module

  Input:   void *Handle           (Handle pointer)
           _ccrtaicc_adc_all_enable_t activate (activate/deactivate)
           # CCRTAICC_ADC_ALL_DISABLE
           # CCRTAICC_ADC_ALL_ENABLE
           # CCRTAICC_ADC_ALL_RESET
           # CCRTAICC_ADC_ALL_ENABLE_DO_NOT_CHANGE
  Output:  _ccrtaicc_adc_all_enable_t *current_state (active/deactive)
           # CCRTAICC_ADC_ALL_DISABLE
           # CCRTAICC_ADC_ALL_ENABLE
  Return:  _ccrtaicc_lib_error_number_t
           # CCRTAICC_LIB_NO_ERROR           (successful)
           # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
           # CCRTAICC_LIB_NOT_OPEN         (device not open)
           # CCRTAICC_LIB_INVALID_ARG      (invalid argument)
           # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
*****/
```

2.2.3 ccrtAICC_ADC_Get_CSR()

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

```

/*****
  _ccrtaicc_lib_error_number_t
*****/
```



```

ccrtAICC_ADC_Get_CSR (void                *Handle,
                     _ccrtaicc_adc_mask_t adc_mask,
                     ccrtaicc_adc_csr_t   *adc_csr)

Description: Get ADC Control and Status information

Input:  void                *Handle (Handle pointer)
        _ccrtaicc_adc_mask_t   adc_mask (selected ADC mask)
        # CCRTAICC_ADC_MASK_0_15
        # CCRTAICC_ADC_MASK_16_31
        # CCRTAICC_ADC_MASK_32_47
        # CCRTAICC_ADC_MASK_48_63
        # CCRTAICC_ALL_ADC_MASK

Output: _ccrtaicc_adc_csr_t   *adc_csr (pointer to ADC csr)
        _ccrtaicc_adccsr_update_clock_t   adc_update_clock
        # CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_0:
        # CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_1:
        # CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_2:
        # CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_3:
        # CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_0:
        # CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_1:
        # CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_2:
        # CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_3:
        # CCRTAICC_ADC_UPDATE_NORMAL_EXTERNAL_CLOCK:
        # CCRTAICC_ADC_UPDATE_INVERTED_EXTERNAL_CLOCK:
        # CCRTAICC_ADC_UPDATE_CLOCK_NONE:
        _ccrtaicc_adccsr_speed_select_t   adc_speed_select;
        # CCRTAICC_ADC_NORMAL_SPEED:
        # CCRTAICC_ADC_HIGH_SPEED:
        # CCRTAICC_ADC_SPEED_SELECT_DO_NOT_CHANGE:
        _ccrtaicc_adccsr_data_format_t   adc_data_format
        # CCRTAICC_ADC_OFFSET_BINARY:
        # CCRTAICC_ADC_TWOS_COMPLEMENT:
        # CCRTAICC_ADC_DATA_FORMAT_DO_NOT_CHANGE:
        _ccrtaicc_adccsr_input_range_t   adc_input_range
        # CCRTAICC_ADC_BIPOLAR_5V:
        # CCRTAICC_ADC_BIPOLAR_10V:
        # CCRTAICC_ADC_UNIPOLAR_5V:
        # CCRTAICC_ADC_UNIPOLAR_10V:
        # CCRTAICC_ADC_INPUT_RANGE_DO_NOT_CHANGE:

Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_INVALID_ARG             (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION         (local region not present)
        # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE      (ADC is not active)
*****/

```

2.2.4 ccrtAICC_ADC_Get_Driver_Read_Mode()

This call returns the current driver ADC 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.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Get_Driver_Read_Mode (void                *Handle,
                                  _ccrtaicc_driver_ADC_read_mode_t *mode)

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

Input:  void                *Handle (Handle pointer)

```

```

Output:  _ccrtaicc_driver_ADC_read_mode_t    *mode    (select ADC read mode)
        # CCRTAICC_ADC_PIO_CHANNEL
        # CCRTAICC_ADC_PIO_FIFO
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR              (successful)
        # CCRTAICC_LIB_BAD_HANDLE           (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN             (library not open)
        # CCRTAICC_LIB_NO_LOCAL_REGION      (local region not present)
        # CCRTAICC_LIB_IOCTL_FAILED         (driver ioctl call failed)
        # CCRTAICC_LIB_INVALID_ARG         (invalid argument)
*****/

```

2.2.5 ccrtaICC_ADC_Get_Fifo_Channel_Select()

This call returns the current Fifo Channel selection mask. Only samples for these selected channels are placed in the fifo during sample collection.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_ADC_Get_Fifo_Channel_Select(void    *Handle,
                                       _ccrtaicc_adc_channel_mask_t *adc_fifo_channel_select_mask)

Description: ADC Get Fifo Channel Selection

Input:   void    *Handle    (handle pointer)
Output:  _ccrtaicc_adc_channel_mask_t *adc_fifo_channel_select_mask
                                       (channel select mask)

        # CCRTAICC_ADC_CHANNEL_MASK_0
        # CCRTAICC_ADC_CHANNEL_MASK_1
        # CCRTAICC_ADC_CHANNEL_MASK_2
        # CCRTAICC_ADC_CHANNEL_MASK_3
        # CCRTAICC_ADC_CHANNEL_MASK_4
        # CCRTAICC_ADC_CHANNEL_MASK_5
        # CCRTAICC_ADC_CHANNEL_MASK_6
        # " " "
        # CCRTAICC_ADC_CHANNEL_MASK_58
        # CCRTAICC_ADC_CHANNEL_MASK_59
        # CCRTAICC_ADC_CHANNEL_MASK_60
        # CCRTAICC_ADC_CHANNEL_MASK_61
        # CCRTAICC_ADC_CHANNEL_MASK_62
        # CCRTAICC_ADC_CHANNEL_MASK_63
        # CCRTAICC_ALL_ADC_CHANNELS_MASK

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR              (successful)
        # CCRTAICC_LIB_BAD_HANDLE           (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN             (device not open)
        # CCRTAICC_LIB_INVALID_ARG         (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION      (local region not present)
        # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE   (ADC is not active)
*****/

```

2.2.6 ccrtaICC_ADC_Get_Fifo_Info()

This call returns ADC FIFO information to the user.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_ADC_Get_Fifo_Info (void    *Handle,
                             ccrtaicc_adc_fifo_info_t *adc_fifo)

Description: Get ADC FIFO control and Status information

Input:   void    *Handle    (Handle pointer)

```

```

Output:  ccrtaiicc_adc_fifo_info_t      *adc_fifo (pointer to ADC fifo struct)
        _ccrtaiicc_adc_fifo_reset_t    reset;
        # CCRTAICC_ADC_FIFO_ACTIVE
        # CCRTAICC_ADC_FIFO_RESET
        _ccrtaiicc_adc_fifo_overflow_t  overflow;
        # CCRTAICC_ADC_FIFO_NO_OVERFLOW
        # CCRTAICC_ADC_FIFO_OVERFLOW
        _ccrtaiicc_adc_fifo_underflow_t underflow;
        # CCRTAICC_ADC_FIFO_NO_UNDERFLOW
        # CCRTAICC_ADC_FIFO_UNDERFLOW
        _ccrtaiicc_adc_fifo_full_t      full;
        # CCRTAICC_ADC_FIFO_NOT_FULL
        # CCRTAICC_ADC_FIFO_FULL
        _ccrtaiicc_adc_fifo_threshold_t threshold_exceeded;
        # CCRTAICC_ADC_FIFO_THRESHOLD_NOT_EXCEEDED
        # CCRTAICC_ADC_FIFO_THRESHOLD_EXCEEDED
        _ccrtaiicc_adc_fifo_empty_t     empty;
        # CCRTAICC_ADC_FIFO_NOT_EMPTY
        # CCRTAICC_ADC_FIFO_EMPTY
        uint                             data_counter;
        uint                             threshold;
        uint                             max_threshold;
        uint                             driver_threshold;
        _ccrtaiicc_adc_channel_mask_t    channel_select_mask;
        # CCRTAICC_ADC_CHANNEL_MASK_0
        # CCRTAICC_ADC_CHANNEL_MASK_1
        # CCRTAICC_ADC_CHANNEL_MASK_2
        # CCRTAICC_ADC_CHANNEL_MASK_3
        # CCRTAICC_ADC_CHANNEL_MASK_4
        # CCRTAICC_ADC_CHANNEL_MASK_5
        # CCRTAICC_ADC_CHANNEL_MASK_6
        # " " "
        # CCRTAICC_ADC_CHANNEL_MASK_58
        # CCRTAICC_ADC_CHANNEL_MASK_59
        # CCRTAICC_ADC_CHANNEL_MASK_60
        # CCRTAICC_ADC_CHANNEL_MASK_61
        # CCRTAICC_ADC_CHANNEL_MASK_62
        # CCRTAICC_ADC_CHANNEL_MASK_63
        # CCRTAICC_ALL_ADC_CHANNELS_MASK

Return:  _ccrtaiicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
*****/

```

2.2.7 ccrtAICC_ADC_Get_Fifo_Threshold()

This call returns the ADC Fifo threshold information.

```

/*****
_ccrtaiicc_lib_error_number_t
ccrtAICC_ADC_Get_Fifo_Threshold(void *Handle,
                               uint *adc_threshold)

```

Description: ADC Get Fifo Threshold

```

Input:  void          *Handle          (handle pointer)
Output: uint          *adc_threshold    (ADC fifo threshold)
Return: _ccrtaiicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR        (successful)

```

```

# CCRTAICC_LIB_BAD_HANDLE          (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN           (device not open)
# CCRTAICC_LIB_INVALID_ARG        (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION    (local region not present)
# CCRTAICC_LIB_ADC_IS_NOT_ACTIVE  (ADC is not active)
*****/

```

2.2.8 ccrtAICC_ADC_Get_Input_Control()

This call returns the ADC input control information.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Get_Input_Control (void          *Handle,
                               _ccrtaicc_adc_input_control_t *adc_input_control)

Description: Return Input Control information for all the ADCs

Input:   void          *Handle   (handle pointer)
Output:  _ccrtaicc_adc_input_control_t
        *adc_input_control (pointer to control select)

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_ADC_INPUT_CONTROL_EXTERNAL_SIGNAL
        # CCRTAICC_ADC_INPUT_CONTROL_CALIBRATION_BUS
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region error)
        # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN         (device not open)
*****/

```

2.2.9 ccrtAICC_ADC_Get_Negative_Cal()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Get_Negative_Cal (void          *Handle,
                               ccrtaicc_adc_cal_t *cal)

Description: Get the ADC Negative Calibration data.

Input:   void          *Handle   (handle pointer)
Output:  ccrtaicc_adc_cal_t      *cal   (pointer to board cal)
        uint          Raw[CCRTAICC_MAX_ADC_CHANNELS];
        double        Float[CCRTAICC_MAX_ADC_CHANNELS];

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN         (device not open)
        # CCRTAICC_LIB_INVALID_ARG      (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
*****/

```

2.2.10 ccrtAICC_ADC_Get_Offset_Cal()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Get_Offset_Cal (void          *Handle,
                              ccrtaicc_adc_cal_t *cal)

```

Description: Get the ADC Offset Calibration data.

```
Input:  void                    *Handle (handle pointer)
Output: ccrtAICC_adc_cal_t      *cal   (pointer to board cal)
        uint   Raw[CCRTAICC_MAX_ADC_CHANNELS];
        double Float[CCRTAICC_MAX_ADC_CHANNELS];
Return: _ccrtAICC_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
*****/
```

2.2.11 ccrtAICC_ADC_Get_Positive_Cal()

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

```
/*
*_ccrtAICC_lib_error_number_t
ccrtAICC_ADC_Get_Positive_Cal(void *Handle,
                               ccrtAICC_adc_cal_t *cal)

Description: Get the ADC Positive Calibration data.

Input:  void                    *Handle (handle pointer)
Output: ccrtAICC_adc_cal_t      *cal   (pointer to board cal)
        uint   Raw[CCRTAICC_MAX_ADC_CHANNELS];
        double Float[CCRTAICC_MAX_ADC_CHANNELS];
Return: _ccrtAICC_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
*****/
```

2.2.12 ccrtAICC_ADC_Perform_Auto_Calibration()

This single call performs a full ADC calibration of all the selected channels using the internal reference voltages. Users need to perform this calibration after they have configured the channels with the appropriate clocks and frequencies, voltage references and channel speed selection. The reason for this is that these selections will have a direct bearing on the calibrated values. Failure to do so will result in inaccurate ADC readings.

```
/*
*_ccrtAICC_lib_error_number_t
ccrtAICC_ADC_Perform_Auto_Calibration (void *Handle,
                                       _ccrtAICC_adc_channel_t chan_start,
                                       _ccrtAICC_adc_channel_t chan_end)

Description: Perform ADC Auto Calibration

Input:  void                    *Handle (handle pointer)
        _ccrtAICC_adc_channel_t chan_start (start channel number)
        # CCRTAICC_ADC_CHANNEL_0
        # CCRTAICC_ADC_CHANNEL_1
        # CCRTAICC_ADC_CHANNEL_2
        # CCRTAICC_ADC_CHANNEL_3
        # CCRTAICC_ADC_CHANNEL_4
*****/
```

```

# CCRTAICC_ADC_CHANNEL_5
# CCRTAICC_ADC_CHANNEL_6
# " " "
# CCRTAICC_ADC_CHANNEL_58
# CCRTAICC_ADC_CHANNEL_59
# CCRTAICC_ADC_CHANNEL_60
# CCRTAICC_ADC_CHANNEL_61
# CCRTAICC_ADC_CHANNEL_62
# CCRTAICC_ADC_CHANNEL_63
_ccrtaicc_adc_channel_t chan_end          (end channel number)
# CCRTAICC_ADC_CHANNEL_0
# CCRTAICC_ADC_CHANNEL_1
# CCRTAICC_ADC_CHANNEL_2
# CCRTAICC_ADC_CHANNEL_3
# CCRTAICC_ADC_CHANNEL_4
# CCRTAICC_ADC_CHANNEL_5
# CCRTAICC_ADC_CHANNEL_6
# " " "
# CCRTAICC_ADC_CHANNEL_58
# CCRTAICC_ADC_CHANNEL_59
# CCRTAICC_ADC_CHANNEL_60
# CCRTAICC_ADC_CHANNEL_61
# CCRTAICC_ADC_CHANNEL_62
# CCRTAICC_ADC_CHANNEL_63
Output: none
Return: _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR                (successful)
# CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN                (library not open)
# CCRTAICC_LIB_NO_LOCAL_REGION         (local region not present)
# CCRTAICC_LIB_NO_RESOURCE             (no free PLL available)
# CCRTAICC_LIB_IO_ERROR                (read error)
# CCRTAICC_LIB_ADC_IS_NOT_ACTIVE       (ADC is not active)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
# CCRTAICC_LIB_CLOCK_NOT_ASSIGNED_TO_ADC (clock not assigned to ADC)
*****/

```

2.2.13 ccrtAICC_ADC_Perform_External_Negative_Calibration()

Use this call to perform an external negative calibration. Prior to calling this function, the ADC inputs must be provided with a negative signal close to -10 Volts, otherwise this call will fail. Additionally, the user can specify a range of channels.

Users need to perform this calibration after they have configured the channels with the appropriate clocks and frequencies, voltage references and channel speed selection. The reason for this is that these selections will have a direct bearing on the calibrated values. Failure to do so will result in inaccurate ADC readings.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Perform_External_Negative_Calibration(void *Handle,
                                                    _ccrtaicc_adc_channel_t chan_start,
                                                    _ccrtaicc_adc_channel_t chan_end,
                                                    double ReferenceVoltage)

```

Description: Perform ADC External Negative Calibration

```

Input: void *Handle          (handle pointer)
       _ccrtaicc_adc_channel_t chan_start (start channel)
       # CCRTAICC_ADC_CHANNEL_0
       # CCRTAICC_ADC_CHANNEL_1
       # CCRTAICC_ADC_CHANNEL_2
       # CCRTAICC_ADC_CHANNEL_3

```

```

# CCRTAICC_ADC_CHANNEL_4
# CCRTAICC_ADC_CHANNEL_5
# CCRTAICC_ADC_CHANNEL_6
# " " "
# CCRTAICC_ADC_CHANNEL_58
# CCRTAICC_ADC_CHANNEL_59
# CCRTAICC_ADC_CHANNEL_60
# CCRTAICC_ADC_CHANNEL_61
# CCRTAICC_ADC_CHANNEL_62
# CCRTAICC_ADC_CHANNEL_63
_ccrtaicc_adc_channel_t chan_end          (end channel)
# CCRTAICC_ADC_CHANNEL_0
# CCRTAICC_ADC_CHANNEL_1
# CCRTAICC_ADC_CHANNEL_2
# CCRTAICC_ADC_CHANNEL_3
# CCRTAICC_ADC_CHANNEL_4
# CCRTAICC_ADC_CHANNEL_5
# CCRTAICC_ADC_CHANNEL_6
# " " "
# CCRTAICC_ADC_CHANNEL_58
# CCRTAICC_ADC_CHANNEL_59
# CCRTAICC_ADC_CHANNEL_60
# CCRTAICC_ADC_CHANNEL_61
# CCRTAICC_ADC_CHANNEL_62
# CCRTAICC_ADC_CHANNEL_63
double          ReferenceVoltage (Reference Voltage)
Output: none
Return: _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR          (successful)
# CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN        (library not open)
# CCRTAICC_LIB_INVALID_ARG     (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTAICC_LIB_NO_RESOURCE     (no free PLL available)
# CCRTAICC_LIB_IO_ERROR        (read error)
# CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
# CCRTAICC_LIB_CLOCK_NOT_ASSIGNED_TO_ADC (clock not assigned to ADC)
*****/

```

2.2.14 ccrtAICC_ADC_Perform_External_Offset_Calibration()

Use this call to perform an external offset calibration. Prior to calling this function, the ADC inputs must be provided with a offset signal close to 0 Volts, otherwise this call will fail. Additionally, the user can specify a range of channels. Once this call is executed, the user will need to perform external negative and external positive calibrations as this call resets these gains to 1.0 prior to calibration.

Users need to perform this calibration after they have configured the channels with the appropriate clocks and frequencies, voltage references and channel speed selection. The reason for this is that these selections will have a direct bearing on the calibrated values. Failure to do so will result in inaccurate ADC readings.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaicc_adc_perform_external_offset_calibration(void          *Handle,
                                                _ccrtaicc_adc_channel_t chan_start,
                                                _ccrtaicc_adc_channel_t chan_end)

```

Description: Perform ADC External Offset Calibration

```

Input:   void          *Handle          (handle pointer)
         _ccrtaicc_adc_channel_t chan_start (start channel)

```

```

# CCRTAICC_ADC_CHANNEL_0
# CCRTAICC_ADC_CHANNEL_1
# CCRTAICC_ADC_CHANNEL_2
# CCRTAICC_ADC_CHANNEL_3
# CCRTAICC_ADC_CHANNEL_4
# CCRTAICC_ADC_CHANNEL_5
# CCRTAICC_ADC_CHANNEL_6
# " " "
# CCRTAICC_ADC_CHANNEL_58
# CCRTAICC_ADC_CHANNEL_59
# CCRTAICC_ADC_CHANNEL_60
# CCRTAICC_ADC_CHANNEL_61
# CCRTAICC_ADC_CHANNEL_62
# CCRTAICC_ADC_CHANNEL_63
_ccrtaicc_adc_channel_t chan_end          (end channel)
# CCRTAICC_ADC_CHANNEL_0
# CCRTAICC_ADC_CHANNEL_1
# CCRTAICC_ADC_CHANNEL_2
# CCRTAICC_ADC_CHANNEL_3
# CCRTAICC_ADC_CHANNEL_4
# CCRTAICC_ADC_CHANNEL_5
# CCRTAICC_ADC_CHANNEL_6
# " " "
# CCRTAICC_ADC_CHANNEL_58
# CCRTAICC_ADC_CHANNEL_59
# CCRTAICC_ADC_CHANNEL_60
# CCRTAICC_ADC_CHANNEL_61
# CCRTAICC_ADC_CHANNEL_62
# CCRTAICC_ADC_CHANNEL_63
Output: none
Return: _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR                (successful)
# CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN                (library not open)
# CCRTAICC_LIB_INVALID_ARG            (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION         (local region not present)
# CCRTAICC_LIB_NO_RESOURCE             (no free PLL available)
# CCRTAICC_LIB_IO_ERROR                (read error)
# CCRTAICC_LIB_ADC_IS_NOT_ACTIVE       (ADC is not active)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
# CCRTAICC_LIB_CLOCK_NOT_ASSIGNED_TO_ADC (clock not assigned to ADC)
*****/

```

2.2.15 ccrtAICC_ADC_Perform_External_Positive_Calibration()

Use this call to perform an external positive calibration. Prior to calling this function, the ADC inputs must be provided with a positive signal close to +10 Volts, otherwise this call will fail. Additionally, the user can specify a range of channels.

Users need to perform this calibration after they have configured the channels with the appropriate clocks and frequencies, voltage references and channel speed selection. The reason for this is that these selections will have a direct bearing on the calibrated values. Failure to do so will result in inaccurate ADC readings.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Perform_External_Positive_Calibration(void      *Handle,
                                                    _ccrtaicc_adc_channel_t chan_start,
                                                    _ccrtaicc_adc_channel_t chan_end,
                                                    double          ReferenceVoltage)

```

Description: Perform ADC External Positive Calibration


```

Input: void *Handle (handle pointer)
       _ccrtaicc_adc_channel_t chan_start (start channel)
       # CCRTAICC_ADC_CHANNEL_0
       # CCRTAICC_ADC_CHANNEL_1
       # CCRTAICC_ADC_CHANNEL_2
       # CCRTAICC_ADC_CHANNEL_3
       # CCRTAICC_ADC_CHANNEL_4
       # CCRTAICC_ADC_CHANNEL_5
       # CCRTAICC_ADC_CHANNEL_6
       # " " "
       # CCRTAICC_ADC_CHANNEL_58
       # CCRTAICC_ADC_CHANNEL_59
       # CCRTAICC_ADC_CHANNEL_60
       # CCRTAICC_ADC_CHANNEL_61
       # CCRTAICC_ADC_CHANNEL_62
       # CCRTAICC_ADC_CHANNEL_63
       _ccrtaicc_adc_channel_t chan_end (end channel)
       # CCRTAICC_ADC_CHANNEL_0
       # CCRTAICC_ADC_CHANNEL_1
       # CCRTAICC_ADC_CHANNEL_2
       # CCRTAICC_ADC_CHANNEL_3
       # CCRTAICC_ADC_CHANNEL_4
       # CCRTAICC_ADC_CHANNEL_5
       # CCRTAICC_ADC_CHANNEL_6
       # " " "
       # CCRTAICC_ADC_CHANNEL_58
       # CCRTAICC_ADC_CHANNEL_59
       # CCRTAICC_ADC_CHANNEL_60
       # CCRTAICC_ADC_CHANNEL_61
       # CCRTAICC_ADC_CHANNEL_62
       # CCRTAICC_ADC_CHANNEL_63
double ReferenceVoltage (Reference Voltage)
Output: none
Return: _ccrtaicc_lib_error_number_t
       # CCRTAICC_LIB_NO_ERROR (successful)
       # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
       # CCRTAICC_LIB_NOT_OPEN (library not open)
       # CCRTAICC_LIB_INVALID_ARG (invalid argument)
       # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
       # CCRTAICC_LIB_NO_RESOURCE (no free PLL available)
       # CCRTAICC_LIB_IO_ERROR (read error)
       # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
       # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
       # CCRTAICC_LIB_CLOCK_NOT_ASSIGNED_TO_ADC (clock not assigned to ADC)
*****/

```

2.2.16 ccrtaICC_ADC_Perform_Negative_Calibration()

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

Users need to perform this calibration after they have configured the channels with the appropriate clocks and frequencies, voltage references and channel speed selection. The reason for this is that these selections will have a direct bearing on the calibrated values. Failure to do so will result in inaccurate ADC readings.

```

/*****
   _ccrtaicc_lib_error_number_t
   ccrtaICC_ADC_Perform_Negative_Calibration (void *Handle,
                                             _ccrtaicc_adc_channel_t chan_start,
                                             _ccrtaicc_adc_channel_t chan_end)

```

Description: Perform ADC Negative Calibration

```

Input:   void                *Handle          (handle pointer)
        _ccrtaicc_adc_channel_t chan_start  (start channel number)
        # CCRTAICC_ADC_CHANNEL_0
        # CCRTAICC_ADC_CHANNEL_1
        # CCRTAICC_ADC_CHANNEL_2
        # CCRTAICC_ADC_CHANNEL_3
        # CCRTAICC_ADC_CHANNEL_4
        # CCRTAICC_ADC_CHANNEL_5
        # CCRTAICC_ADC_CHANNEL_6
        # " " "
        # CCRTAICC_ADC_CHANNEL_58
        # CCRTAICC_ADC_CHANNEL_59
        # CCRTAICC_ADC_CHANNEL_60
        # CCRTAICC_ADC_CHANNEL_61
        # CCRTAICC_ADC_CHANNEL_62
        # CCRTAICC_ADC_CHANNEL_63
        _ccrtaicc_adc_channel_t chan_end    (end channel number)
        # CCRTAICC_ADC_CHANNEL_0
        # CCRTAICC_ADC_CHANNEL_1
        # CCRTAICC_ADC_CHANNEL_2
        # CCRTAICC_ADC_CHANNEL_3
        # CCRTAICC_ADC_CHANNEL_4
        # CCRTAICC_ADC_CHANNEL_5
        # CCRTAICC_ADC_CHANNEL_6
        # " " "
        # CCRTAICC_ADC_CHANNEL_58
        # CCRTAICC_ADC_CHANNEL_59
        # CCRTAICC_ADC_CHANNEL_60
        # CCRTAICC_ADC_CHANNEL_61
        # CCRTAICC_ADC_CHANNEL_62
        # CCRTAICC_ADC_CHANNEL_63
Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR             (successful)
        # CCRTAICC_LIB_BAD_HANDLE          (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN            (library not open)
        # CCRTAICC_LIB_NO_LOCAL_REGION     (local region not present)
        # CCRTAICC_LIB_NO_RESOURCE         (no free PLL available)
        # CCRTAICC_LIB_IO_ERROR            (read error)
        # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE   (ADC is not active)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
        # CCRTAICC_LIB_CLOCK_NOT_ASSIGNED_TO_ADC (clock not assigned to ADC)
        *****/

```

2.2.17 ccrtAICC_ADC_Perform_Offset_Calibration()

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

Users need to perform this calibration after they have configured the channels with the appropriate clocks and frequencies, voltage references and channel speed selection. The reason for this is that these selections will have a direct bearing on the calibrated values. Failure to do so will result in inaccurate ADC readings.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Perform_Offset_Calibration (void                *Handle,
                                         _ccrtaicc_adc_channel_t chan_start,
                                         _ccrtaicc_adc_channel_t chan_end)

```

Description: Perform ADC Offset Calibration

```

Input:   void                *Handle          (handle pointer)
         _ccrtaicc_adc_channel_t chan_start  (start channel number)
         # CCRTAICC_ADC_CHANNEL_0
         # CCRTAICC_ADC_CHANNEL_1
         # CCRTAICC_ADC_CHANNEL_2
         # CCRTAICC_ADC_CHANNEL_3
         # CCRTAICC_ADC_CHANNEL_4
         # CCRTAICC_ADC_CHANNEL_5
         # CCRTAICC_ADC_CHANNEL_6
         # " " "
         # CCRTAICC_ADC_CHANNEL_58
         # CCRTAICC_ADC_CHANNEL_59
         # CCRTAICC_ADC_CHANNEL_60
         # CCRTAICC_ADC_CHANNEL_61
         # CCRTAICC_ADC_CHANNEL_62
         # CCRTAICC_ADC_CHANNEL_63
         _ccrtaicc_adc_channel_t chan_end    (end channel number)
         # CCRTAICC_ADC_CHANNEL_0
         # CCRTAICC_ADC_CHANNEL_1
         # CCRTAICC_ADC_CHANNEL_2
         # CCRTAICC_ADC_CHANNEL_3
         # CCRTAICC_ADC_CHANNEL_4
         # CCRTAICC_ADC_CHANNEL_5
         # CCRTAICC_ADC_CHANNEL_6
         # " " "
         # CCRTAICC_ADC_CHANNEL_58
         # CCRTAICC_ADC_CHANNEL_59
         # CCRTAICC_ADC_CHANNEL_60
         # CCRTAICC_ADC_CHANNEL_61
         # CCRTAICC_ADC_CHANNEL_62
         # CCRTAICC_ADC_CHANNEL_63
Output:  none
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR             (successful)
         # CCRTAICC_LIB_BAD_HANDLE          (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN            (library not open)
         # CCRTAICC_LIB_NO_LOCAL_REGION     (local region not present)
         # CCRTAICC_LIB_NO_RESOURCE        (no free PLL available)
         # CCRTAICC_LIB_IO_ERROR           (read error)
         # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE  (ADC is not active)
         # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
         # CCRTAICC_LIB_CLOCK_NOT_ASSIGNED_TO_ADC
                                                (clock not assigned to ADC)
*****/

```

2.2.18 ccrtAICC_ADC_Perform_Positive_Calibration()

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

Users need to perform this calibration after they have configured the channels with the appropriate clocks and frequencies, voltage references and channel speed selection. The reason for this is that these selections will have a direct bearing on the calibrated values. Failure to do so will result in inaccurate ADC readings.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Perform_Positive_Calibration (void                *Handle,
                                           _ccrtaicc_adc_channel_t chan_start,
                                           _ccrtaicc_adc_channel_t chan_end)

```

Description: Perform ADC Positive Calibration

```

Input:   void                *Handle          (handle pointer)
         _ccrtaicc_adc_channel_t chan_start  (start channel number)

```

```

# CCRTAICC_ADC_CHANNEL_0
# CCRTAICC_ADC_CHANNEL_1
# CCRTAICC_ADC_CHANNEL_2
# CCRTAICC_ADC_CHANNEL_3
# CCRTAICC_ADC_CHANNEL_4
# CCRTAICC_ADC_CHANNEL_5
# CCRTAICC_ADC_CHANNEL_6
# " " "
# CCRTAICC_ADC_CHANNEL_58
# CCRTAICC_ADC_CHANNEL_59
# CCRTAICC_ADC_CHANNEL_60
# CCRTAICC_ADC_CHANNEL_61
# CCRTAICC_ADC_CHANNEL_62
# CCRTAICC_ADC_CHANNEL_63
_ccrtaicc_adc_channel_t chan_end          (end channel number)
# CCRTAICC_ADC_CHANNEL_0
# CCRTAICC_ADC_CHANNEL_1
# CCRTAICC_ADC_CHANNEL_2
# CCRTAICC_ADC_CHANNEL_3
# CCRTAICC_ADC_CHANNEL_4
# CCRTAICC_ADC_CHANNEL_5
# CCRTAICC_ADC_CHANNEL_6
# " " "
# CCRTAICC_ADC_CHANNEL_58
# CCRTAICC_ADC_CHANNEL_59
# CCRTAICC_ADC_CHANNEL_60
# CCRTAICC_ADC_CHANNEL_61
# CCRTAICC_ADC_CHANNEL_62
# CCRTAICC_ADC_CHANNEL_63
Output: none
Return: _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR                (successful)
# CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN                (library not open)
# CCRTAICC_LIB_NO_LOCAL_REGION         (local region not present)
# CCRTAICC_LIB_NO_RESOURCE             (no free PLL available)
# CCRTAICC_LIB_IO_ERROR                (read error)
# CCRTAICC_LIB_ADC_IS_NOT_ACTIVE       (ADC is not active)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE     (Clock is not active)
# CCRTAICC_LIB_CLOCK_NOT_ASSIGNED_TO_ADC (clock not assigned to ADC)
*****/

```

2.2.19 **ccrtaICC_ADC_Read_Channels()**

This call provides the user an easy method of reading the ADC channels. User can supply a channel mask. If pointer to *adc_csr* is NULL, then the routine itself computes the current ADC configuration. For performance, the user should get the current ADC configuration using the *ccrtaICC_ADC_Get_CSR()* call to get the current settings and pass it to this routine. Hence, if the configuration is not changed, the user can continuously invoke *ccrtaICC_ADC_Read_Channels()* routine without incurring the additional overhead of routine calling the *ccrtaICC_ADC_Get_CSR()* call.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_ADC_Read_Channels(void          *Handle,
                                _ccrtaicc_adc_channel_mask_t ChanMask,
                                _ccrtaicc_adc_csr_t          *adc_csr,
                                ccrtaicc_adc_volts_t         *adc_volts)

```

Description: Read ADC Channels

```

Input: void          *Handle          (Handle pointer)
       _ccrtaicc_adc_channel_mask_t  ChanMask      (specify channel mask)

```

```

# CCRTAICC_ADC_CHANNEL_MASK_0
# CCRTAICC_ADC_CHANNEL_MASK_1
# CCRTAICC_ADC_CHANNEL_MASK_2
# CCRTAICC_ADC_CHANNEL_MASK_3
# CCRTAICC_ADC_CHANNEL_MASK_4
# CCRTAICC_ADC_CHANNEL_MASK_5
# CCRTAICC_ADC_CHANNEL_MASK_6
# " " "
# CCRTAICC_ADC_CHANNEL_MASK_58
# CCRTAICC_ADC_CHANNEL_MASK_59
# CCRTAICC_ADC_CHANNEL_MASK_60
# CCRTAICC_ADC_CHANNEL_MASK_61
# CCRTAICC_ADC_CHANNEL_MASK_62
# CCRTAICC_ADC_CHANNEL_MASK_63
# CCRTAICC_ALL_ADC_CHANNELS_MASK
_ccrtaicc_adc_csr_t          *adc_csr (pointer to ADC csr)
_ccrtaicc_adccsr_update_clock_t  adc_update_clock
# CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_0:
# CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_1:
# CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_2:
# CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_3:
# CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_0:
# CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_1:
# CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_2:
# CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_3:
# CCRTAICC_ADC_UPDATE_NORMAL_EXTERNAL_CLOCK:
# CCRTAICC_ADC_UPDATE_INVERTED_EXTERNAL_CLOCK:
# CCRTAICC_ADC_UPDATE_CLOCK_NONE:
_ccrtaicc_adccsr_speed_select_t  adc_speed_select;
# CCRTAICC_ADC_NORMAL_SPEED:
# CCRTAICC_ADC_HIGH_SPEED:
# CCRTAICC_ADC_SPEED_SELECT_DO_NOT_CHANGE:
_ccrtaicc_adccsr_data_format_t  adc_data_format
# CCRTAICC_ADC_OFFSET_BINARY:
# CCRTAICC_ADC_TWOS_COMPLEMENT:
# CCRTAICC_ADC_DATA_FORMAT_DO_NOT_CHANGE:
_ccrtaicc_adccsr_input_range_t  adc_input_range
# CCRTAICC_ADC_BIPOLAR_5V:
# CCRTAICC_ADC_BIPOLAR_10V:
# CCRTAICC_ADC_UNIPOLAR_5V:
# CCRTAICC_ADC_UNIPOLAR_10V:
# CCRTAICC_ADC_INPUT_RANGE_DO_NOT_CHANGE:
Output: ccrtaicc_adc_volts_t          *adc_volts (pointer to ADC volts)
uint      Raw[CCRTAICC_MAX_ADC_CHANNELS];
double    Float[CCRTAICC_MAX_ADC_CHANNELS];
Return:  _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR              (successful)
# CCRTAICC_LIB_BAD_HANDLE            (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN              (library not open)
# CCRTAICC_LIB_INVALID_ARG           (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION       (local region not present)
# CCRTAICC_LIB_ADC_IS_NOT_ACTIVE     (ADC is not active)
*****/

```

2.2.20 ccrtaICC_ADC_Read_Channels_Calibration()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_ADC_Read_Channels_Calibration(void *Handle,
char *filename)

```

Description: Read ADC Channels Calibration

Input: void *Handle (handle pointer)
Output: char *filename (pointer to filename)
Return: _ccrtaicc_lib_error_number_t
CCRTAICC_LIB_NO_ERROR (successful)
CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
CCRTAICC_LIB_NOT_OPEN (library not open)
CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
CCRTAICC_LIB_CANNOT_OPEN_FILE (cannot open calib. file)

*****/
e.g.

#Date : Fri Dec 14 11:30:00 2018

Table with 4 columns: #Chan, Negative, Offset, Positive. Rows include ch00, ch01, ch02, ch03, . . ., ch61, ch62, ch63.

2.2.21 ccrtaICC_ADC_Reset_Calibration()

This call resets the gain and offset information for all channels to default, i.e. both the positive and negative gains are set to 1 and the offset is set to 0.

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_ADC_Reset_Calibration (void *Handle)

Description: Reset Calibration Data

Input: void *Handle (handle pointer)
Output: none
Return: _ccrtaicc_lib_error_number_t
CCRTAICC_LIB_NO_ERROR (successful)
CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
CCRTAICC_LIB_NOT_OPEN (library not open)
CCRTAICC_LIB_INVALID_ARG (invalid argument)
CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
CCRTAICC_LIB_NO_RESOURCE (no free PLL available)
CCRTAICC_LIB_IO_ERROR (read error)

*****/

2.2.22 ccrtaICC_ADC_Reset_Fifo()

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

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_ADC_Reset_Fifo(void *Handle,
_ccrtaicc_adc_fifo_reset_t activate)

Description: ADC Reset Fifo

Input: void *Handle (handle pointer)
_ccrtaicc_adc_fifo_reset_t activate (activate converter)

```

        # CCRTAICC_ADC_FIFO_ACTIVATE
        # CCRTAICC_ADC_FIFO_RESET
Output: none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG      (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
*****/

```

2.2.23 ccrtaICC_ADC_Set_CSR()

This call sets the ADC control registers for the selected channel group. For accurate results during data collection, the user needs to perform a calibration of the channels once are configured and the associated clocks started.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_ADC_Set_CSR (void          *Handle,
                     _ccrtaicc_adc_mask_t  adc_mask,
                     ccrtaicc_adc_csr_t    *adc_csr)

Description: Set ADC Control and Status information

Input:  void          *Handle (Handle pointer)
        _ccrtaicc_adc_mask_t  adc_mask (selected ADC mask)
        # CCRTAICC_ADC_MASK_0_15
        # CCRTAICC_ADC_MASK_16_31
        # CCRTAICC_ADC_MASK_32_47
        # CCRTAICC_ADC_MASK_48_63
        # CCRTAICC_ALL_ADC_MASK
        _ccrtaicc_adc_csr_t          *adc_csr (pointer to ADC csr)
        _ccrtaicc_adccsr_update_clock_t  adc_update_clock
        # CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_0:
        # CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_1:
        # CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_2:
        # CCRTAICC_ADC_UPDATE_NORMAL_CLOCK_3:
        # CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_0:
        # CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_1:
        # CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_2:
        # CCRTAICC_ADC_UPDATE_INVERTED_CLOCK_3:
        # CCRTAICC_ADC_UPDATE_NORMAL_EXTERNAL_CLOCK:
        # CCRTAICC_ADC_UPDATE_INVERTED_EXTERNAL_CLOCK:
        # CCRTAICC_ADC_UPDATE_CLOCK_NONE:
        _ccrtaicc_adccsr_speed_select_t  adc_speed_select;
        # CCRTAICC_ADC_NORMAL_SPEED:
        # CCRTAICC_ADC_HIGH_SPEED:
        # CCRTAICC_ADC_SPEED_SELECT_DO_NOT_CHANGE:
        _ccrtaicc_adccsr_data_format_t  adc_data_format
        # CCRTAICC_ADC_OFFSET_BINARY:
        # CCRTAICC_ADC_TWOS_COMPLEMENT:
        # CCRTAICC_ADC_DATA_FORMAT_DO_NOT_CHANGE:
        _ccrtaicc_adccsr_input_range_t  adc_input_range
        # CCRTAICC_ADC_BIPOLAR_5V:
        # CCRTAICC_ADC_BIPOLAR_10V:
        # CCRTAICC_ADC_UNIPOLAR_5V:
        # CCRTAICC_ADC_UNIPOLAR_10V:
        # CCRTAICC_ADC_INPUT_RANGE_DO_NOT_CHANGE:

Output: none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)

```

```

# CCRTAICC_LIB_BAD_HANDLE           (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN             (library not open)
# CCRTAICC_LIB_BAD_HANDLE           (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN             (device not open)
# CCRTAICC_LIB_INVALID_ARG         (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION     (local region not present)
# CCRTAICC_LIB_ADC_IS_NOT_ACTIVE   (ADC is not active)
*****/

```

2.2.24 ccrtAICC_ADC_Set_Driver_Read_Mode()

This call sets the current driver ADC 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. If the *read(2)* call fails with the *ENOBUFS* error, an overflow condition has occurred. User will need to reduce the clock speed and/or the number of selected channels until an overflow condition is no longer triggered.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Set_Driver_Read_Mode (void                *Handle,
                                   _ccrtaicc_driver_ADC_read_mode_t  mode)

Description: Select Driver ADC Read Mode

Input:   void                *Handle (Handle pointer)
         _ccrtaicc_driver_ADC_read_mode_t mode (select ADC read mode)
         # CCRTAICC_ADC_PIO_CHANNEL
         # CCRTAICC_ADC_PIO_FIFO

Output:  none

Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR           (successful)
         # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN        (device not open)
         # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
         # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
*****/

```

2.2.25 ccrtAICC_ADC_Set_Fifo_Channel_Select()

This call allows the user to select a set of channels that need to be captured in the ADC Fifo. For ADCs that have the high speed option *CCRTAICC_ADC_HIGH_SPEED* selected, the odd (*unused*) channels for the ADC will *not* be included in the Fifo, even if the odd channels are specified in the channel selection mask.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Set_Fifo_Channel_Select(void        *Handle,
                                       _ccrtaicc_adc_channel_mask_t
                                       adc_fifo_channel_select_mask)

Description: ADC Set Fifo Channel Selection

Input:   void                *Handle (handle pointer)
         _ccrtaicc_adc_channel_mask_t adc_fifo_channel_select_mask
                                               (channel select mask)

         # CCRTAICC_ADC_CHANNEL_MASK_0
         # CCRTAICC_ADC_CHANNEL_MASK_1
         # CCRTAICC_ADC_CHANNEL_MASK_2
         # CCRTAICC_ADC_CHANNEL_MASK_3
         # CCRTAICC_ADC_CHANNEL_MASK_4
         # CCRTAICC_ADC_CHANNEL_MASK_5
         # CCRTAICC_ADC_CHANNEL_MASK_6
         # " " "

```



```

# CCRTAICC_ADC_CHANNEL_MASK_58
# CCRTAICC_ADC_CHANNEL_MASK_59
# CCRTAICC_ADC_CHANNEL_MASK_60
# CCRTAICC_ADC_CHANNEL_MASK_61
# CCRTAICC_ADC_CHANNEL_MASK_62
# CCRTAICC_ADC_CHANNEL_MASK_63
# CCRTAICC_ALL_ADC_CHANNELS_MASK
Output: none
Return: _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR (successful)
# CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN (device not open)
# CCRTAICC_LIB_INVALID_ARG (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
*****/

```

2.2.26 ccrtaICC_ADC_Set_Fifo_Threshold()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_ADC_Set_Fifo_Threshold(void *Handle,
                                uint adc_threshold)

Description: ADC Set Fifo Threshold

Input: void *Handle (handle pointer)
       uint adc_threshold (ADC fifo threshold)
Output: none
Return: _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR (successful)
# CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN (device not open)
# CCRTAICC_LIB_INVALID_ARG (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
*****/

```

2.2.27 ccrtaICC_ADC_Set_Input_Control()

This call allows the user to select whether the inputs to all the ADCs are from an *external* supply or the *calibration* bus. If calibration bus is selected, then the user can select one of several reference voltages or ground reference using the *ccrtaICC_Set_Calibration_CSR()* to direct it as an input to the all the active ADCs.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_ADC_Set_Input_Control (void *Handle,
                                _ccrtaicc_adc_input_control_t adc_input_control)

Description: Set Input Control information for all the ADCs

Input: void *Handle (handle pointer)
       _ccrtaicc_adc_input_control_t
       adc_input_control (control select)
# CCRTAICC_ADC_INPUT_CONTROL_EXTERNAL_SIGNAL
# CCRTAICC_ADC_INPUT_CONTROL_CALIBRATION_BUS
Output: none
Return: _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR (successful)
# CCRTAICC_LIB_NO_LOCAL_REGION (local region error)

```

```

# CCRTAICC_LIB_BAD_HANDLE          (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN            (device not open)
*****/

```

2.2.28 ccrtAICC_ADC_Set_Negative_Cal()

This call allows the user to set the negative calibration data for all the channels by supplying floating point *Float* gains to the call. Users can supply CCRTAICC_DO_NOT_CHANGE as a gain for any channel that should not be changed. Additionally, this call will return the *RAW* value of the gain supplied that is written to the board.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Set_Negative_Cal(void          *Handle,
                                ccrtaicc_adc_cal_t *cal)

```

Description: Set the ADC Negative Calibration data.

```

Input:   void          *Handle      (handle pointer)
         ccrtaicc_adc_cal_t      *cal      (pointer to board cal)
         uint          Raw[CCRTAICC_MAX_ADC_CHANNELS];
         double        Float[CCRTAICC_MAX_ADC_CHANNELS];

```

Output: none

```

Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR          (successful)
         # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN        (library not open)
         # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
         # CCRTAICC_LIB_NO_LOCAL_REGION  (local region not present)
         # CCRTAICC_LIB_NO_RESOURCE     (no free PLL available)
         # CCRTAICC_LIB_IO_ERROR        (read error)
         # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)

```

```

*****/

```

2.2.29 ccrtAICC_ADC_Set_Offset_Cal()

This call allows the user to set the offset calibration data for all the channels by supplying floating point *Float* offset to the call. Users can supply CCRTAICC_DO_NOT_CHANGE as a gain for any channel that should not be changed. Additionally, this call will return the *Raw* value of the offset supplied that is written to the board.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_ADC_Set_Offset_Cal(void          *Handle,
                                ccrtaicc_adc_cal_t *cal)

```

Description: Set the ADC Offset Calibration data.

```

Input:   void          *Handle      (handle pointer)
         ccrtaicc_adc_cal_t      *cal      (pointer to board cal)
         uint          Raw[CCRTAICC_MAX_ADC_CHANNELS];
         double        Float[CCRTAICC_MAX_ADC_CHANNELS];

```

Output: none

```

Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR          (successful)
         # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN        (library not open)
         # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
         # CCRTAICC_LIB_NO_LOCAL_REGION  (local region not present)
         # CCRTAICC_LIB_NO_RESOURCE     (no free PLL available)
         # CCRTAICC_LIB_IO_ERROR        (read error)
         # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)

```

```

*****/

```

2.2.30 `ccrtaICC_ADC_Set_Positive_Cal()`

This call allows the user to set the positive calibration data for all the channels by supplying floating point *Float* gains to the call. Users can supply `CCRTAICC_DO_NOT_CHANGE` as a gain for any channel that should not be changed. Additionally, this call will return the *Raw* value of the gain supplied that is written to the board.

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtaICC_ADC_Set_Positive_Cal(void          *Handle,
                                ccrtaicc_adc_cal_t *cal)

Description: Set the ADC Positive Calibration data.

Input:      void          *Handle      (handle pointer)
            ccrtaicc_adc_cal_t *cal    (pointer to board cal)
            uint          Raw[CCRTAICC_MAX_ADC_CHANNELS];
            double        Float[CCRTAICC_MAX_ADC_CHANNELS];

Output:     none

Return:     _ccrtaicc_lib_error_number_t
            # CCRTAICC_LIB_NO_ERROR          (successful)
            # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
            # CCRTAICC_LIB_NOT_OPEN        (library not open)
            # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
            # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
            # CCRTAICC_LIB_NO_RESOURCE     (no free PLL available)
            # CCRTAICC_LIB_IO_ERROR        (read error)
            # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
*****/
```

2.2.31 `ccrtaICC_ADC_Write_Channels_Calibration()`

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

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtaICC_ADC_Write_Channels_Calibration(void *Handle,
                                           char *filename)

Description: Write Channels Calibration

Input:      void *Handle      (handle pointer)
Output:     char *filename    (pointer to filename)
Return:     _ccrtaicc_lib_error_number_t
            # CCRTAICC_LIB_NO_ERROR          (successful)
            # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
            # CCRTAICC_LIB_NOT_OPEN        (library not open)
            # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
            # CCRTAICC_LIB_CANNOT_OPEN_FILE (cannot open calib. file)
            # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
            # CCRTAICC_LIB_ADC_IS_NOT_ACTIVE (ADC is not active)
*****/
```

e.g.

```
#Date          : Fri Dec 14 11:30:00 2018

#Chan  Negative          Offset          Positive
#====  =====          =====          =====
ch00:  1.000178198330104351043701172  -0.00022888183593750000  1.000139905605465173721313477
ch01:  0.999954220838844776153564453  -0.00007629394531250000  0.999967454932630062103271484
ch02:  1.000127669423818588256835938  -0.00015258789062500000  1.000118218362331390380859375
ch03:  0.999876655638217926025390625  0.00000000000000000000  0.999875877983868122100830078
. . .
ch61:  1.000136931426823139190673828  -0.00007629394531250000  1.000138226430863142013549805
ch62:  1.000080410391092300415039062  -0.00007629394531250000  1.000093937385827302932739258
ch63:  1.000272549688816070556640625  -0.00007629394531250000  1.000277046114206314086914062
```

2.2.32 ccrtAICC_Add_Irq()

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

```
/*
 * int ccrtAICC_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 ccrtAICC_Remove_Irq(), then this call adds it back.
 *
 * Input:      void *Handle          (Handle pointer)
 * Output:     none
 * Return:     _ccrtaicc_lib_error_number_t
 *              # CCRTAICC_LIB_NO_ERROR          (successful)
 *              # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
 *              # CCRTAICC_LIB_NOT_OPEN        (library not open)
 *              # CCRTAICC_LIB_IOCTL_FAILED    (driver ioctl call failed)
 */
```

2.2.33 ccrtAICC_BoardExpirationTimeRemaining()

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

```
/*
 * _ccrtaicc_lib_error_number_t
 * ccrtAICC_BoardExpirationTimeRemaining(void *Handle,
 *                                       time_t *SecondsToExpire,
 *                                       ccrtAICC_date_string_t *GmtDateTimeString,
 *                                       ccrtAICC_date_string_t *LocalDateTimeString,
 *                                       _ccrtaicc_firmware_state *FirmwareState)
 *
 * Description: Number of seconds to expire on a restricted card
 *
 * Input:      void *Handle          (Handle pointer)
 * Output:     time_t *SecondsToExpire (seconds to expire)
 *              ccrtAICC_date_string_t *GmtDateTimeString (GMT date/time
 *                                                         string)
 *              char date[CCRTAICC_DATE_TIME_STRING_SIZE]
 *              ccrtAICC_date_string_t *LocalDateTimeString (Local date/time
 *                                                         string)
 *              char date[CCRTAICC_DATE_TIME_STRING_SIZE]
 *              _ccrtaicc_firmware_state *FirmwareState (Firmware State)
 *              # CCRTAICC_FIRMWARE_STATE_UNRESTRICTED
 *              # CCRTAICC_FIRMWARE_STATE_RESTRICTED
 *              # CCRTAICC_FIRMWARE_STATE_EXPIRED
 * Return:     _ccrtaicc_lib_error_number_t
 *              # CCRTAICC_LIB_NO_ERROR          (successful)
 */
```

```

# CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN (library not open)
*****/

```

Mandatory arguments to the call are **Handle* and **SecondsToExpire*. Rest of the arguments are optional and be set to *NULL*.

SecondsToExpire* – If the board has an expiration date, this call will return the number of seconds this card can be used before it expires. *Once the card has expired, this call will not be reached as the device open will fail with an authorization error.***

If the board has no expiration date, this call will return zero as the number of seconds.

**GmtDateTimeString* – If the board has an expiration date, this ascii GMT date representation of the expiration date is available in this variable if it is not NULL

**LocalDateTimeString* – If the board has an expiration date, this ascii Local date representation of the expiration date is available in this variable if it is not NULL

**FirmwareState* – This returns the current state of the installed firmware. I can be one of:

- CCRTAICC_FIRMWARE_STATE_UNRESTRICTED. This firmware has no restrictions.
- CCRTAICC_FIRMWARE_STATE_RESTRICTED. This firmware has restrictions. It is possible that and expiration date restriction is not present.
- CCRTAICC_FIRMWARE_STATE_EXPIRED. This firmware has restrictions. One of the restrictions is the expiration date which has expired. Typically, you may not see this state as the utility will fail during the open with an authentication error.

2.2.34 ccrtAICC_Clear_Cable_Fault()

This call clears any latched fault conditions that were created due to overvoltage

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clear_Cable_Fault (void *Handle,
                           ccrtaicc_cable_fault_csr_t *CableFaultCSR)

```

Description: Clear Cable Fault

Input: void *Handle (handle pointer)
ccrtaicc_cable_fault_csr_t *CableFaultCSR (pointer to cable fault CSR)

```

_ccrtaicc_cable_fault_group_mask_t LatchedFaultStatus
# CCRTAICC_CABLE_FAULT_GROUP_00_03_MASK
# CCRTAICC_CABLE_FAULT_GROUP_04_07_MASK
# CCRTAICC_CABLE_FAULT_GROUP_08_11_MASK
# CCRTAICC_CABLE_FAULT_GROUP_12_15_MASK
# CCRTAICC_CABLE_FAULT_GROUP_16_19_MASK
# CCRTAICC_CABLE_FAULT_GROUP_20_23_MASK
# CCRTAICC_CABLE_FAULT_GROUP_24_27_MASK
# CCRTAICC_CABLE_FAULT_GROUP_28_31_MASK
# CCRTAICC_CABLE_FAULT_GROUP_32_35_MASK
# CCRTAICC_CABLE_FAULT_GROUP_36_39_MASK
# CCRTAICC_CABLE_FAULT_GROUP_40_43_MASK
# CCRTAICC_CABLE_FAULT_GROUP_44_47_MASK
# CCRTAICC_CABLE_FAULT_GROUP_48_51_MASK
# CCRTAICC_CABLE_FAULT_GROUP_52_55_MASK
# CCRTAICC_CABLE_FAULT_GROUP_56_59_MASK

```

```

        # CCRTAICC_CABLE_FAULT_GROUP_60_63_MASK
        # CCRTAICC_ALL_CABLE_FAULT_GROUPS_MASK
Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region error)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
*****/

```

2.2.35 ccrtaicc_Clear_Driver_Error()

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

```

/*****
  _ccrtaicc_lib_error_number_t ccrtaicc_Clear_Driver_Error(void *Handle)

Description: Clear any previously generated driver related error.

Input:   void *Handle           (Handle pointer)
Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_IOCTL_FAILED      (driver ioctl call failed)
*****/

```

2.2.36 ccrtaicc_Clear_Lib_Error()

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

```

/*****
  _ccrtaicc_lib_error_number_t ccrtaicc_Clear_Lib_Error(void *Handle)

Description: Clear any previously generated library related error.

Input:   void *Handle           (Handle pointer)
Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
*****/

```

2.2.37 ccrtaicc_Clock_Generator_Disable_Outputs()

This call is used to disable the clock generator outputs. No data collection can occur until the clock outputs are re-enabled.

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtaicc_Clock_Generator_Disable_Outputs (void *Handle)

Description: Disable Clock Generator Outputs

Input:   void *Handle           (Handle pointer)
Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
*****/

```

```

# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.38 ccrtAICC_Clock_Generator_Enable_Outputs()

This call is used to enable the clock generator outputs if they have been previously disabled by the *ccrtAICC_Clock_Generator_Disable_Outputs()* call.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Generator_Enable_Outputs (void *Handle)

Description: Enable Clock Generator Outputs

Input:   void *Handle           (Handle pointer)
Output:  none
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR           (successful)
         # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN          (device not open)
         # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.39 ccrtAICC_Clock_Generator_Soft_Reset()

Perform a soft clock reset on all the output clocks.

```

/*****
_ccrtaicc_lib_error_number_t ccrtAICC_Clock_Generator_Soft_Reset(void *Handle)

Description: Perform Soft Reset to Clock Generator

Input:   void *Handle           (Handle pointer)
Output:  none
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR           (successful)
         # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN          (device not open)
         # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.40 ccrtAICC_Clock_Get_Generator_CSR()

Return the clock generator control and status register.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Get_Generator_CSR (void *Handle,
                                  ccrtaicc_clkgen_csr_t *CgCsr)

Description: Get Generator Control and Status information

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

_ccrtaicc_clkgen_interface_t interface
# CCRTAICC_CLOCK_GENERATOR_INTERFACE_IDLE
# CCRTAICC_CLOCK_GENERATOR_INTERFACE_BUSY
_ccrtaicc_clkgen_output_t output
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_DISABLE
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_ENABLE
_ccrtaicc_clkgen_state_t state
# CCRTAICC_CLOCK_GENERATOR_ACTIVE

```

```

        # CCRTAICC_CLOCK_GENERATOR_RESET
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE             (no/bad handler
                                           supplied)
        # CCRTAICC_LIB_NOT_OPEN              (device not open)
        # CCRTAICC_LIB_INVALID_ARG           (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION       (local region not
                                           present)
*****/

```

2.2.41 ccrtaicc_clock_get_generator_info()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaicc_clock_get_generator_info (void                *Handle,
                                   _ccrtaicc_clock_generator_output_t WhichOutput,
                                   ccrtaicc_clock_generator_info_t *CgInfo)

```

Description: Get Clock Generator Information

```

Input:  void                *Handle    (Handle pointer)
        _ccrtaicc_clock_generator_output_t WhichOutput (select output)
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_9

Output: ccrtaicc_clock_generator_info_t *CgInfo (pointer to clock
                                                generator info)
        __u64                M_divider_num
        __u32                M_divider_den
        __u64                N_divider_num
        __u32                N_divider_den
        __u32                R_divider_value
        __u32                R_divider
        _ccrtaicc_cg_zero_delay_t ZeroDelay
        # CCRTAICC_CG_ZERO_DELAY_MODE
        # CCRTAICC_CG_NORMAL_MODE
        _ccrtaicc_cg_stat_ctrl_voltsel_t Voltage_select
        # CCRTAICC_CG_VOLTAGE_SELECT_1_8V
        # CCRTAICC_CG_VOLTAGE_SELECT_3_3V
        _ccrtaicc_cg_input_xaxb_extclk_sel_t Input_xaxb_selection
        # CCRTAICC_CG_INPUT_XAXB_USE_CRYSTAL
        # CCRTAICC_CG_INPUT_XAXB_USE_EXTCLK_SOURCE
        _ccrtaicc_cg_xaxb_power_down_t Input_xaxb_power
        # CCRTAICC_CG_XAXB_POWER_DOWN
        # CCRTAICC_CG_XAXB_DO_NOT_POWER_DOWN
        ccrtaicc_clkgen_csr_t Clkcsr
        _ccrtaicc_clkgen_interface_t interface
        # CCRTAICC_CLOCK_GENERATOR_INTERFACE_IDLE
        # CCRTAICC_CLOCK_GENERATOR_INTERFACE_BUSY
        _ccrtaicc_clkgen_output_t output
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_DISABLE
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_ENABLE
        _ccrtaicc_clkgen_state_t state

```



```

# CCRTAICC_CLOCK_GENERATOR_ACTIVE
# CCRTAICC_CLOCK_GENERATOR_RESET
ccrtaicc_clkgen_output_config_t          Config
  _ccrtaicc_cg_outcfg_force_rdiv2_t     force_rdiv2
  # CCRTAICC_CG_OUTPUT_CONFIG_DONT_FORCE_RDIV2
  # CCRTAICC_CG_OUTPUT_CONFIG_FORCE_RDIV2
  _ccrtaicc_cg_outcfg_enable_t          enable
  # CCRTAICC_CG_OUTPUT_CONFIG_DISABLE
  # CCRTAICC_CG_OUTPUT_CONFIG_ENABLE
  _ccrtaicc_cg_outcfg_shutdown_t        shutdown
  # CCRTAICC_CG_OUTPUT_CONFIG_POWER_UP
  # CCRTAICC_CG_OUTPUT_CONFIG_SHUTDOWN
ccrtaicc_clkgen_output_format_t          Format
  _ccrtaicc_cg_outfmt_cmos_drive_t      cmos_drive
  # CCRTAICC_CG_OUTPUT_FORMAT_CMOS_DRIVE_LVDS
  # CCRTAICC_CG_OUTPUT_FORMAT_CMOS_DRIVE_CMOS
  _ccrtaicc_cg_outfmt_disable_state_t   disable_state
  # CCRTAICC_CG_OUTPUT_FORMAT_DISABLE_LOW
  # CCRTAICC_CG_OUTPUT_FORMAT_DISABLE_HIGH
  _ccrtaicc_cg_outfmt_sync_t            sync
  # CCRTAICC_CG_OUTPUT_FORMAT_SYNC_DISABLE
  # CCRTAICC_CG_OUTPUT_FORMAT_SYNC_ENABLE
  _ccrtaicc_cg_outfmt_format_t          format
  # CCRTAICC_CG_OUTPUT_FORMAT_FORMAT_LVDS
  # CCRTAICC_CG_OUTPUT_FORMAT_FORMAT_CMOS
ccrtaicc_clkgen_output_mode_t            Mode
  _ccrtaicc_cg_outmode_amplitude_t      amplitude
  # CCRTAICC_CG_OUTPUT_AMPLITUDE_CMOS
  # CCRTAICC_CG_OUTPUT_AMPLITUDE_LVDS
  _ccrtaicc_cg_outmode_common_t         common
  # CCRTAICC_CG_OUTPUT_COMMON_CMOS
  # CCRTAICC_CG_OUTPUT_COMMON_LVDS
  # CCRTAICC_CG_OUTPUT_COMMON_LVPECL
ccrtaicc_clkgen_output_mux_t             Mux
  _ccrtaicc_cg_outmux_inversion_t       inversion
  # CCRTAICC_CG_OUTPUT_MUX_COMPLEMENTARY
  # CCRTAICC_CG_OUTPUT_MUX_IN_PHASE
  # CCRTAICC_CG_OUTPUT_MUX_INVERTED
  # CCRTAICC_CG_OUTPUT_MUX_OUT_OF_PHASE
  _ccrtaicc_cg_outmux_ndiv_select_t     ndiv_mux
  # CCRTAICC_CG_OUTPUT_MUX_NDIV_0
  # CCRTAICC_CG_OUTPUT_MUX_NDIV_1
  # CCRTAICC_CG_OUTPUT_MUX_NDIV_2
  # CCRTAICC_CG_OUTPUT_MUX_NDIV_3
  # CCRTAICC_CG_OUTPUT_MUX_NDIV_4
ccrtaicc_clkgen_input_clock_enable_t     Input_clock_enable
  _ccrtaicc_cg_input_clock_enable_t     input_0_clock
  # CCRTAICC_CG_INPUT_CLOCK_DISABLE
  # CCRTAICC_CG_INPUT_CLOCK_ENABLE
  _ccrtaicc_cg_input_clock_enable_t     input_1_clock
  # CCRTAICC_CG_INPUT_CLOCK_DISABLE
  # CCRTAICC_CG_INPUT_CLOCK_ENABLE
  _ccrtaicc_cg_input_clock_enable_t     input_2_clock
  # CCRTAICC_CG_INPUT_CLOCK_DISABLE
  # CCRTAICC_CG_INPUT_CLOCK_ENABLE
  _ccrtaicc_cg_input_clock_enable_t     input_fb_clock
  # CCRTAICC_CG_INPUT_CLOCK_DISABLE
  # CCRTAICC_CG_INPUT_CLOCK_ENABLE
ccrtaicc_clkgen_input_clock_select_t     Input_clock_select
  _ccrtaicc_cg_input_clock_select_control_t control
  # CCRTAICC_CG_INPUT_CLOCK_SELECT_PIN_CONTROL
  # CCRTAICC_CG_INPUT_CLOCK_SELECT_REG_CONTROL
  _ccrtaicc_cg_input_clock_select_register_t select

```

```

# CCRTAICC_CG_INPUT_CLOCK_SELECT_IN0
# CCRTAICC_CG_INPUT_CLOCK_SELECT_IN1
# CCRTAICC_CG_INPUT_CLOCK_SELECT_IN2
# CCRTAICC_CG_INPUT_CLOCK_SELECT_INXAXB
ccrtaicc_pdiv_all_info_t          Pdiv_info
__u64                             Pfb_divider
ccrtaicc_pdiv_info_t            P0
__u64                             Divider
  _ccrtaicc_cg_pdiv_enable_t     Enable
    # CCRTAICC_CG_PDIV_DISABLE
    # CCRTAICC_CG_PDIV_ENABLE
  _ccrtaicc_cg_pdiv_input_state_t State
    # CCRTAICC_CG_PDIV_INPUT_UNUSED
    # CCRTAICC_CG_PDIV_INPUT_DISABLED
    # CCRTAICC_CG_PDIV_INPUT_SELECTED
ccrtaicc_pdiv_info_t            P1
__u64                             Divider
  _ccrtaicc_cg_pdiv_enable_t     Enable
    # CCRTAICC_CG_PDIV_DISABLE
    # CCRTAICC_CG_PDIV_ENABLE
  _ccrtaicc_cg_pdiv_input_state_t State
    # CCRTAICC_CG_PDIV_INPUT_UNUSED
    # CCRTAICC_CG_PDIV_INPUT_DISABLED
    # CCRTAICC_CG_PDIV_INPUT_SELECTED
ccrtaicc_pdiv_info_t            P2
__u64                             Divider
  _ccrtaicc_cg_pdiv_enable_t     Enable
    # CCRTAICC_CG_PDIV_DISABLE
    # CCRTAICC_CG_PDIV_ENABLE
  _ccrtaicc_cg_pdiv_input_state_t State
    # CCRTAICC_CG_PDIV_INPUT_UNUSED
    # CCRTAICC_CG_PDIV_INPUT_DISABLED
    # CCRTAICC_CG_PDIV_INPUT_SELECTED
ccrtaicc_pdiv_info_t            Pxaxb
__u64                             Divider
  _ccrtaicc_cg_pdiv_enable_t     Enable
    # CCRTAICC_CG_PDIV_DISABLE
    # CCRTAICC_CG_PDIV_ENABLE
  _ccrtaicc_cg_pdiv_input_state_t State
    # CCRTAICC_CG_PDIV_INPUT_UNUSED
    # CCRTAICC_CG_PDIV_INPUT_DISABLED
    # CCRTAICC_CG_PDIV_INPUT_SELECTED
int                               Which_Pdiv_Selected
int                               P_Divider
long double                       OutputClockFrequency;
# <valid positive output clock frequency>
# CCRTAICC_CLOCK_ERROR_INVALID_P_DIVIDER
# CCRTAICC_CLOCK_ERROR_VCO_CLOCK_NOT_IN_RANGE
# CCRTAICC_CLOCK_ERROR_N_DIVIDER_NOT_IN_RANGE
# CCRTAICC_CLOCK_ERROR_P_DIVIDER_NOT_IN_RANGE
# CCRTAICC_CLOCK_ERROR_R_DIVIDER_NOT_IN_RANGE
# CCRTAICC_CLOCK_ERROR_INVALID_CLOCK_FREQUENCY
Return: _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR           (successful)
# CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN          (device not open)
# CCRTAICC_LIB_INVALID_ARG       (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.42 ccrtAICC_Clock_Get_Generator_Input_Clock_Enable()

This call returns the status of all the input clocks.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Get_Generator_Input_Clock_Enable (void *Handle,
                                                ccrtaicc_clkgen_input_clock_enable_t *InputClockEnable)

Description: Return the Clock Generator Input Clock Enable

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

_ccrtaicc_cg_input_clock_enable_t input_0_clock
# CCRTAICC_CG_INPUT_CLOCK_DISABLE
# CCRTAICC_CG_INPUT_CLOCK_ENABLE
_ccrtaicc_cg_input_clock_enable_t input_1_clock
# CCRTAICC_CG_INPUT_CLOCK_DISABLE
# CCRTAICC_CG_INPUT_CLOCK_ENABLE
_ccrtaicc_cg_input_clock_enable_t input_2_clock
# CCRTAICC_CG_INPUT_CLOCK_DISABLE
# CCRTAICC_CG_INPUT_CLOCK_ENABLE
_ccrtaicc_cg_input_clock_enable_t input_fb_clock
# CCRTAICC_CG_INPUT_CLOCK_DISABLE
# CCRTAICC_CG_INPUT_CLOCK_ENABLE
Return:  _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR (successful)
# CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN (device not open)
# CCRTAICC_LIB_INVALID_ARG (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/
```

2.2.43 ccrtAICC_Clock_Get_Generator_Input_Clock_Select()

This call returns the input clock selection.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Get_Generator_Input_Clock_Select (void *Handle,
                                                ccrtaicc_clkgen_input_clock_select_t *ClkSel)

Description: Get Input Clock Selection

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

_ccrtaicc_cg_input_clock_select_control_t control;
# CCRTAICC_CG_INPUT_CLOCK_SELECT_PIN_CONTROL
# CCRTAICC_CG_INPUT_CLOCK_SELECT_REG_CONTROL
_ccrtaicc_cg_input_clock_select_register_t select;
# CCRTAICC_CG_INPUT_CLOCK_SELECT_IN0
# CCRTAICC_CG_INPUT_CLOCK_SELECT_IN1
# CCRTAICC_CG_INPUT_CLOCK_SELECT_IN2
# CCRTAICC_CG_INPUT_CLOCK_SELECT_INXAXB
Return:  _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR (successful)
# CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN (device not open)
# CCRTAICC_LIB_INVALID_ARG (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION (local region error)
*****/
```

```

# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE          (Clock is not active)
*****

```

2.2.44 ccrtAICC_Clock_Get_Generator_Input_Clock_Status()

The call returns the input clock status.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Get_Generator_Input_Clock_Status (void *Handle,
                                                ccrtaicc_clkgen_input_clock_status_t *ClkStatus)

```

Description: Get Input Clock Status

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

```

_ccrtaicc_cg_calibration_status_t calstat
# CCRTAICC_CG_STATUS_DEVICE_IS_NOT_CALIBRATING
# CCRTAICC_CG_STATUS_DEVICE_IS_CALIBRATING
_ccrtaicc_cg_lol_pll_locked_t PLL_locked
# CCRTAICC_CG_STATUS_LOL_PLL_LOCKED
# CCRTAICC_CG_STATUS_LOL_PLL_NOT_LOCKED
_ccrtaicc_cg_smbus_timeout_error_t SMBUS_timeout
# CCRTAICC_CG_STATUS_LOL_SMBUS_NOT_TIMEDOUT
# CCRTAICC_CG_STATUS_LOL_SMBUS_TIMEDOUT
_ccrtaicc_cg_los_signal_present_t input_signal
# CCRTAICC_CG_STATUS_LOS_SIGNAL_PRESENT
# CCRTAICC_CG_STATUS_LOS_SIGNAL_NOT_PRESENT
_ccrtaicc_cg_los_alarm_t input_0_clock
# CCRTAICC_CG_LOS_INPUT_CLOCK_PRESENT
# CCRTAICC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
_ccrtaicc_cg_los_alarm_t input_1_clock
# CCRTAICC_CG_LOS_INPUT_CLOCK_PRESENT
# CCRTAICC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
_ccrtaicc_cg_los_alarm_t input_2_clock
# CCRTAICC_CG_LOS_INPUT_CLOCK_PRESENT
# CCRTAICC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
_ccrtaicc_cg_los_alarm_t input_fb_clock
# CCRTAICC_CG_LOS_INPUT_CLOCK_PRESENT
# CCRTAICC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
_ccrtaicc_cg_losxaxb_signal_present_t input_xaxb_clock
# CCRTAICC_CG_LOS_INPUT_CLOCK_PRESENT
# CCRTAICC_CG_LOS_INPUT_CLOCK_NOT_PRESENT
Return: _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR (successful)
# CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN (device not open)
# CCRTAICC_LIB_INVALID_ARG (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****

```

2.2.45 ccrtAICC_Clock_Get_Generator_M_Divider()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Get_Generator_M_Divider (void *Handle,
                                         _u64 *Numerator,
                                         _u32 *Denominator,
                                         long double *Value)

```

Description: Return Clock Generator M-Divider Numerator and Denominator

```

Input:  void                *Handle          (Handle pointer)
Output:  __u64              *Numerator     (pointer to Numerator)
        __u32              *Denominator    (pointer to Denominator)
        long double        *Value          (pointer to Value)
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION    (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.46 ccrtaicc_clock_get_generator_n_divider()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaicc_clock_get_generator_n_divider (void                *Handle,
                                       _ccrtaicc_clock_generator_divider_t WhichDivider,
                                       __u64                    *Numerator,
                                       __u32                    *Denominator,
                                       long double              *Value)

```

Description: Return Clock Generator N-Divider Numerator and Denominator

```

Input:  void                *Handle          (Handle pointer)
        _ccrtaicc_clock_generator_divider_t WhichDivider (select divider)
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_N0
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_N1
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_N2
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_N3
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_N4
Output:  __u64              *Numerator     (pointer to Numerator)
        __u32              *Denominator    (pointer to Denominator)
        long double        *Value          (pointer to Value)
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION    (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.47 ccrtaicc_clock_get_generator_output_config()

Return the clock generator output configuration for the selected output.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaicc_clock_get_generator_output_config (void                *Handle,
                                           _ccrtaicc_clock_generator_output_t WhichOutput,
                                           ccrtaicc_clkgen_output_config_t *OutCfg)

```

Description: Return Clock Generator Output Configuration

```

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

```

```

# CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_9
Output: ccrtaiicc_clkgen_output_config_t *OutCfg (pointer to output config)
        _ccrtaiicc_cg_outcfg_force_rdiv2_t force_rdiv2
        # CCRTAICC_CG_OUTPUT_CONFIG_DONT_FORCE_RDIV2
        # CCRTAICC_CG_OUTPUT_CONFIG_FORCE_RDIV2
        _ccrtaiicc_cg_outcfg_enable_t enable
        # CCRTAICC_CG_OUTPUT_CONFIG_DISABLE
        # CCRTAICC_CG_OUTPUT_CONFIG_ENABLE
        _ccrtaiicc_cg_outcfg_shutdown_t shutdown
        # CCRTAICC_CG_OUTPUT_CONFIG_POWER_UP
        # CCRTAICC_CG_OUTPUT_CONFIG_SHUTDOWN
Return: _ccrtaiicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR (successful)
        # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN (device not open)
        # CCRTAICC_LIB_INVALID_ARG (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****

```

2.2.48 ccrtAICC_Clock_Get_Generator_Output_Format()

Return the clock generator output format for the selected output.

```

/*****
_ccrtaiicc_lib_error_number_t
ccrtAICC_Clock_Get_Generator_Output_Format (void *Handle,
        _ccrtaiicc_clock_generator_output_t WhichOutput,
        ccrtaiicc_clkgen_output_format_t *OutFmt)

```

Description: Return Clock Generator Output Format

```

Input: void *Handle (Handle pointer)
        _ccrtaiicc_clock_generator_output_t WhichOutput (select output)
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_9
Output: ccrtaiicc_clkgen_output_format_t *OutFmt (pointer to output format)
        _ccrtaiicc_cg_outfmt_cmos_drive_t cmos_drive
        # CCRTAICC_CG_OUTPUT_FORMAT_CMOS_DRIVE_LVDS
        # CCRTAICC_CG_OUTPUT_FORMAT_CMOS_DRIVE_CMOS
        _ccrtaiicc_cg_outfmt_disable_state_t disable_state
        # CCRTAICC_CG_OUTPUT_FORMAT_DISABLE_LOW
        # CCRTAICC_CG_OUTPUT_FORMAT_DISABLE_HIGH
        _ccrtaiicc_cg_outfmt_sync_t sync
        # CCRTAICC_CG_OUTPUT_FORMAT_SYNC_DISABLE
        # CCRTAICC_CG_OUTPUT_FORMAT_SYNC_ENABLE

```

```

        _ccrtaicc_cg_outfmt_format_t          format
        # CCRTAICC_CG_OUTPUT_FORMAT_FORMAT_LVDS
        # CCRTAICC_CG_OUTPUT_FORMAT_FORMAT_CMOS
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR              (successful)
        # CCRTAICC_LIB_BAD_HANDLE            (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN              (device not open)
        # CCRTAICC_LIB_INVALID_ARG           (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION       (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
*****/

```

2.2.49 ccrtaICC_Clock_Get_Generator_Output_Mode()

Return the clock generator output mode for the selected output.

```

/*****
    _ccrtaicc_lib_error_number_t
    ccrtaICC_Clock_Get_Generator_Output_Mode (void          *Handle,
                                               _ccrtaicc_clock_generator_output_t WhichOutput,
                                               ccrtaicc_clkgen_output_mode_t      *OutMode)

Description: Return Clock Generator Output Mode

Input:  void          *Handle      (Handle pointer)
        _ccrtaicc_clock_generator_output_t WhichOutput (select output)
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_9

Output: ccrtaicc_clkgen_output_mode_t *OutMode (pointer to output
        amplitude/common mode)
        _ccrtaicc_cg_outmode_amplitude_t amplitude
        # CCRTAICC_CG_OUTPUT_AMPLITUDE_CMOS
        # CCRTAICC_CG_OUTPUT_AMPLITUDE_LVDS
        _ccrtaicc_cg_outmode_common_t common
        # CCRTAICC_CG_OUTPUT_COMMON_CMOS
        # CCRTAICC_CG_OUTPUT_COMMON_LVDS
        # CCRTAICC_CG_OUTPUT_COMMON_LVPECL

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR              (successful)
        # CCRTAICC_LIB_BAD_HANDLE            (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN              (device not open)
        # CCRTAICC_LIB_INVALID_ARG           (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION       (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE   (Clock is not active)
*****/

```

2.2.50 ccrtaICC_Clock_Get_Generator_Output_Mux()

Return the clock generator output mux for the selected output.

```

/*****
    _ccrtaicc_lib_error_number_t
    ccrtaICC_Clock_Get_Generator_Output_Mux (void          *Handle,
                                               _ccrtaicc_clock_generator_output_t WhichOutput,
                                               ccrtaicc_clkgen_output_mux_t      *OutMux)

```

Description: Return Clock Generator Output Mux

```
Input: void *Handle (Handle pointer)
       _ccrtaicc_clock_generator_output_t WhichOutput (select output)
       # CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
       # CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
       # CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
       # CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
       # CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
       # CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
       # CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
       # CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
       # CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
       # CCRTAICC_CLOCK_GENERATOR_OUTPUT_9

Output: ccrtaicc_clkgen_output_mux_t *OutMux (pointer to output
       inversion/N-divider mux)
       _ccrtaicc_cg_outmux_inversion_t inversion
       # CCRTAICC_CG_OUTPUT_MUX_COMPLEMENTARY
       # CCRTAICC_CG_OUTPUT_MUX_IN_PHASE
       # CCRTAICC_CG_OUTPUT_MUX_INVERTED
       # CCRTAICC_CG_OUTPUT_MUX_OUT_OF_PHASE
       _ccrtaicc_cg_outmux_ndiv_select_t ndiv_mux
       # CCRTAICC_CG_OUTPUT_MUX_NDIV_0
       # CCRTAICC_CG_OUTPUT_MUX_NDIV_1
       # CCRTAICC_CG_OUTPUT_MUX_NDIV_2
       # CCRTAICC_CG_OUTPUT_MUX_NDIV_3
       # CCRTAICC_CG_OUTPUT_MUX_NDIV_4

Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR (successful)
        # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN (device not open)
        # CCRTAICC_LIB_INVALID_ARG (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/
```

2.2.51 ccrtaicc_clock_get_generator_p_divider()

Return the clock generator P-Divider.

```
/*
*****
_ccrtaicc_lib_error_number_t
ccrtaicc_clock_get_generator_p_divider (void *Handle,
                                         _ccrtaicc_clock_generator_divider_t WhichDivider,
                                         __u64 *Divider)
*****
*/
```

Description: Return Clock Generator P-Divider

```
Input: void *Handle (Handle pointer)
       _ccrtaicc_clock_generator_divider_t WhichDivider (select divider)
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P0
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P1
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P2
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_PFB
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_PXAXB

Output: __u64 *Divider (pointer to
                       Divider)

Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR (successful)
        # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN (device not open)
        # CCRTAICC_LIB_INVALID_ARG (invalid argument)
```



```

# CCRTAICC_LIB_NO_LOCAL_REGION      (local region not present)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.52 ccrtAICC_Clock_Get_Generator_P_Divider_Enable()

Return the clock generator P-Divider Enable state.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaicc_Clock_Get_Generator_P_Divider_Enable (void          *Handle,
                                               _ccrtaicc_clock_generator_divider_t WhichDivider,
                                               _ccrtaicc_cg_pdiv_enable_t      *Pdiv_Enable)

Description: Return Clock Generator P-Divider Enable

Input:   void          *Handle      (Handle pointer)
         _ccrtaicc_clock_generator_divider_t WhichDivider (select divider)
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P0
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P1
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P2
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_PXAXB

Output:  _ccrtaicc_cg_pdiv_enable_t *Pdiv_Enable (pointer to enable
                                                flag)

         # CCRTAICC_CG_PDIV_DISABLE
         # CCRTAICC_CG_PDIV_ENABLE

Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR      (successful)
         # CCRTAICC_LIB_BAD_HANDLE    (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN      (device not open)
         # CCRTAICC_LIB_INVALID_ARG   (invalid argument)
         # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
         # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.53 ccrtAICC_Clock_Get_Generator_R_Divider()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaicc_Clock_Get_Generator_R_Divider (void          *Handle,
                                         _ccrtaicc_clock_generator_divider_t WhichDivider,
                                         __u32          *Divider)

Description: Return Clock Generator R-Divider

Input:   void          *Handle      (Handle pointer)
         _ccrtaicc_clock_generator_divider_t WhichDivider (select divider)
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R0
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R1
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R2
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R3
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R4
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R5
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R6
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R7
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R8
         # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R9

Output:  __u32          *Divider    (pointer to Divider)

Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR      (successful)
         # CCRTAICC_LIB_BAD_HANDLE    (no/bad handler supplied)

```

```

# CCRTAICC_LIB_NOT_OPEN          (device not open)
# CCRTAICC_LIB_INVALID_ARG       (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.54 ccrtAICC_Clock_Get_Generator_Revision()

Return the clock generator revision information.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Get_Generator_Revision (void          *Handle,
                                       ccrtaicc_clock_revision_t *Revision)

Description: Return Clock Generator Revision

Input:   void          *Handle      (Handle pointer)
Output:  ccrtaicc_clock_revision_t *Revision  (pointer to Divider)
         _ccrtaicc_cg_die_revision_t   DieRevision
         # CCRTAICC_CG_SILICON_REVISION_A0
         # CCRTAICC_CG_SILICON_REVISION_A1
         _convert_base_part_number_t   BasePartNumber;
         _convert_base_part_number_t
         u_short BPN
         u_char  NChar[2]
         _ccrtaicc_cg_clock_speed_grade_t  ClockSpeedGrade;
         # CCRTAICC_CG_CLOCK_SPEED_GRADE_A
         # CCRTAICC_CG_CLOCK_SPEED_GRADE_B
         # CCRTAICC_CG_CLOCK_SPEED_GRADE_C
         # CCRTAICC_CG_CLOCK_SPEED_GRADE_D
         _ccrtaicc_cg_clock_revision_t   ClockRevision;
         # CCRTAICC_CG_CLOCK_REVISION_A
         # CCRTAICC_CG_CLOCK_REVISION_B
         # CCRTAICC_CG_CLOCK_REVISION_C
         # CCRTAICC_CG_CLOCK_REVISION_D
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR          (successful)
         # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN        (device not open)
         # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
         # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
         # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.55 ccrtAICC_Clock_Get_Generator_Value()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Get_Generator_Value (void  *Handle,
                                    int    address,
                                    u_char *value)

Description: Return the value of the specified Clock Generator register.

Input:   void          *Handle      (Handle pointer)
         int           address      (clock gen address to display)
Output:  u_char        *value;      (pointer to value)
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR          (successful)
         # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)

```

```

# CCRTAICC_LIB_NOT_OPEN          (device not open)
# CCRTAICC_LIB_INVALID_ARG      (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION  (local region not present)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.56 ccrtAICC_Clock_Get_Generator_Voltage_Select()

Return the clock generator Voltage Selection.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Get_Generator_Voltage_Select (void          *Handle,
                                             _ccrtaicc_cg_stat_ctrl_voltsel_t *VoltSel)

Description: Return the Clock Generator Voltage Selection

Input:   void          *Handle (Handle pointer)
Output:  _ccrtaicc_cg_stat_ctrl_voltsel_t *VoltSel (pointer to voltage select)
         # CCRTAICC_CG_VOLTAGE_SELECT_1_8V
         # CCRTAICC_CG_VOLTAGE_SELECT_3_3V

Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR          (successful)
         # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN        (device not open)
         # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
         # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
         # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.57 ccrtAICC_Clock_Get_Generator_Zero_Delay()

Return the clock generator Zero Delay status.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Get_Generator_Zero_Delay (void          *Handle,
                                           _ccrtaicc_cg_zero_delay_t *ZeroDelay)

Description: Return the Clock Generator Zero Delay setting.

Input:   void          *Handle (Handle pointer)
Output:  _ccrtaicc_cg_zero_delay_t *ZeroDelay (pointer to zero delay)
         # CCRTAICC_CG_ZERO_DELAY_MODE
         # CCRTAICC_CG_NORMAL_MODE

Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR          (successful)
         # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN        (device not open)
         # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
         # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
         # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.58 ccrtAICC_Clock_ReturnOutputFrequency()

This call does not return the actual programmed frequency but instead returns the expected output frequency that would be generated if the specified user input parameters are supplied.

```

/*****
long double
ccrtAICC_Clock_ReturnOutputFrequency(double InputClock,
                                       long double Mdiv_value,

```

```

long double Ndiv_value,
double Pdiv_value,
double Rdiv_value)

Description: Return output frequency

Input:  double      InputClock (input clock frequency in Hz)
        long double Mdiv_value (M-Divider value)
        long double Ndiv_value (N-Divider value)
        double      Pdiv_value (P-Divider value)
        double      Rdiv_value (R-Divider value)
Output: none
Return: long double      returned frequency
*****/

```

2.2.59 ccrtAICC_Clock_Set_Generator_CSR()

This call sets the clock generator control and status register.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaicc_Clock_Set_Generator_CSR (void      *Handle,
                                ccrtaicc_clkgen_csr_t *CgCsr)

Description: Set Clock Generator Control and Status information

Input:  void      *Handle      (Handle pointer)
        ccrtaicc_clkgen_csr_t *CgCsr      (pointer to clock generator csr)
        _ccrtaicc_clkgen_output_t      output
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_DISABLE
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_ENABLE
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_DO_NOT_CHANGE
        _ccrtaicc_clkgen_state_t      state
        # CCRTAICC_CLOCK_GENERATOR_ACTIVE
        # CCRTAICC_CLOCK_GENERATOR_RESET
        # CCRTAICC_CLOCK_GENERATOR_STATE_DO_NOT_CHANGE

Output: none
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR      (successful)
        # CCRTAICC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN      (device not open)
        # CCRTAICC_LIB_INVALID_ARG   (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
*****/

```

2.2.60 ccrtAICC_Clock_Set_Generator_Input_Clock_Enable()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaicc_Clock_Set_Generator_Input_Clock_Enable (void      *Handle,
                                                ccrtaicc_clkgen_input_clock_enable_t *InputClockEnable)

Description: Set Clock Generator Input Clock Enable

Input:  void      *Handle      (Handle pointer)
        ccrtaicc_clkgen_input_clock_enable_t *InputClockEnable (pointer to input clock enable)
        _ccrtaicc_cg_input_clock_enable_t      input_0_clock

```

```

        # CCRTAICC_CG_INPUT_CLOCK_DISABLE
        # CCRTAICC_CG_INPUT_CLOCK_ENABLE
    _ccrtaicc_cg_input_clock_enable_t    input_1_clock
        # CCRTAICC_CG_INPUT_CLOCK_DISABLE
        # CCRTAICC_CG_INPUT_CLOCK_ENABLE
    _ccrtaicc_cg_input_clock_enable_t    input_2_clock
        # CCRTAICC_CG_INPUT_CLOCK_DISABLE
        # CCRTAICC_CG_INPUT_CLOCK_ENABLE
    _ccrtaicc_cg_input_clock_enable_t    input_fb_clock
        # CCRTAICC_CG_INPUT_CLOCK_DISABLE
        # CCRTAICC_CG_INPUT_CLOCK_ENABLE

Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.61 ccrtaICC_Clock_Set_Generator_Input_Clock_Select()

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

```

/*****
    _ccrtaicc_lib_error_number_t
    ccrtaICC_Clock_Set_Generator_Input_Clock_Select (void          *Handle,
                                                    ccrtaicc_clkgen_input_clock_select_t *ClkSel)

Description: Set Clock Generator Input Clock Selection

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

        _ccrtaicc_cg_input_clock_select_control_t    control;
        # CCRTAICC_CG_INPUT_CLOCK_SELECT_PIN_CONTROL
        # CCRTAICC_CG_INPUT_CLOCK_SELECT_REG_CONTROL
    _ccrtaicc_cg_input_clock_select_register_t    select;
        # CCRTAICC_CG_INPUT_CLOCK_SELECT_IN0
        # CCRTAICC_CG_INPUT_CLOCK_SELECT_IN1
        # CCRTAICC_CG_INPUT_CLOCK_SELECT_IN2
        # CCRTAICC_CG_INPUT_CLOCK_SELECT_INXAXB

Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.62 ccrtaICC_Clock_Set_Generator_M_Divider()

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

```

/*****

```

```

_ccrtaicc_lib_error_number_t
ccrtaicc_Clock_Set_Generator_M_Divider (void    *Handle,
                                         __u64  Numerator,
                                         __u32  Denominator,
                                         int     Update)

```

Description: Set Clock Generator M-Divider Numerator and Denominator

```

Input:  void          *Handle          (Handle pointer)
        __u64        Numerator        (Numerator)
        __u32        Denominator      (Denominator)
        int          Update           (True=Update)

Output: none

Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR       (successful)
        # CCRTAICC_LIB_BAD_HANDLE     (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN       (device not open)
        # CCRTAICC_LIB_INVALID_ARG    (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)

```

*****/

2.2.63 ccrtaicc_Clock_Set_Generator_N_Divider()

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

```

/*****
ccrtaicc_Clock_Set_Generator_N_Divider()
_ccrtaicc_lib_error_number_t
ccrtaicc_Clock_Set_Generator_N_Divider (void          *Handle,
                                         _ccrtaicc_clock_generator_divider_t WhichDivider,
                                         __u64        Numerator,
                                         __u32        Denominator,
                                         int          Update)

```

Description: Set Clock Generator N-Divider Numerator and Denominator

```

Input:  void          *Handle          (Handle pointer)
        _ccrtaicc_clock_generator_divider_t WhichDivider (select divider)
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_N0
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_N1
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_N2
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_N3
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_N4
        __u64        Numerator        (Numerator)
        __u32        Denominator      (Denominator)
        int          Update           (True=Update)

Output: none

Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR       (successful)
        # CCRTAICC_LIB_BAD_HANDLE     (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN       (device not open)
        # CCRTAICC_LIB_INVALID_ARG    (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)

```

*****/

2.2.64 ccrtAICC_Clock_Set_Generator_Output_Config()

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

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_Clock_Set_Generator_Output_Config (void          *Handle,
                                             _ccrtaicc_clock_generator_output_t WhichOutput,
                                             ccrtaicc_clkgen_output_config_t   *OutCfg)

Description: Set Clock Generator Output Configuration

Input:   void          *Handle          (Handle pointer)
         _ccrtaicc_clock_generator_output_t WhichOutput (select output)
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_9
         ccrtaicc_clkgen_output_config_t *OutCfg (pointer to output config)
         _ccrtaicc_cg_outcfg_force_rdiv2_t force_rdiv2
         # CCRTAICC_CG_OUTPUT_CONFIG_DONT_FORCE_RDIV2
         # CCRTAICC_CG_OUTPUT_CONFIG_FORCE_RDIV2
         _ccrtaicc_cg_outcfg_enable_t enable
         # CCRTAICC_CG_OUTPUT_CONFIG_DISABLE
         # CCRTAICC_CG_OUTPUT_CONFIG_ENABLE
         _ccrtaicc_cg_outcfg_shutdown_t shutdown
         # CCRTAICC_CG_OUTPUT_CONFIG_POWER_UP
         # CCRTAICC_CG_OUTPUT_CONFIG_SHUTDOWN

Output:  none
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR          (successful)
         # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN        (device not open)
         # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
         # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
         # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/
```

2.2.65 ccrtAICC_Clock_Set_Generator_Output_Format()

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

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_Clock_Set_Generator_Output_Format (void          *Handle,
                                             _ccrtaicc_clock_generator_output_t WhichOutput,
                                             ccrtaicc_clkgen_output_format_t   *OutFmt)

Description: Set Clock Generator Output Format

Input:   void          *Handle          (Handle pointer)
         _ccrtaicc_clock_generator_output_t WhichOutput (select output)
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
```

```

# CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
# CCRTAICC_CLOCK_GENERATOR_OUTPUT_9
ccrtaicc_clkgen_output_format_t          *OutFmt          (pointer to
                                          output format)

_ccrtaicc_cg_outfmt_cmos_drive_t        cmos_drive
# CCRTAICC_CG_OUTPUT_FORMAT_CMOS_DRIVE_LVDS
# CCRTAICC_CG_OUTPUT_FORMAT_CMOS_DRIVE_CMOS
_ccrtaicc_cg_outfmt_disable_state_t      disable_state
# CCRTAICC_CG_OUTPUT_FORMAT_DISABLE_LOW
# CCRTAICC_CG_OUTPUT_FORMAT_DISABLE_HIGH
_ccrtaicc_cg_outfmt_sync_t              sync
# CCRTAICC_CG_OUTPUT_FORMAT_SYNC_DISABLE
# CCRTAICC_CG_OUTPUT_FORMAT_SYNC_ENABLE
_ccrtaicc_cg_outfmt_format_t            format
# CCRTAICC_CG_OUTPUT_FORMAT_FORMAT_LVDS
# CCRTAICC_CG_OUTPUT_FORMAT_FORMAT_CMOS

Output:  none
Return:  _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR                  (successful)
# CCRTAICC_LIB_BAD_HANDLE                 (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN                   (device not open)
# CCRTAICC_LIB_INVALID_ARG                (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION            (local region not present)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE        (Clock is not active)
*****/

```

2.2.66 ccrtAICC_Clock_Set_Generator_Output_Mode()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Set_Generator_Output_Mode (void          *Handle,
                                          _ccrtaicc_clock_generator_output_t WhichOutput,
                                          ccrtaicc_clkgen_output_mode_t *OutMode)

Description: Set Clock Generator Output Mode

Input:  void          *Handle          (Handle pointer)
        _ccrtaicc_clock_generator_output_t WhichOutput (select output)
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_9
        ccrtaicc_clkgen_output_mode_t *OutMode          (pointer to
                                                          output mode)
        _ccrtaicc_cg_outmode_amplitude_t amplitude
        # CCRTAICC_CG_OUTPUT_AMPLITUDE_CMOS

```



```

        # CCRTAICC_CG_OUTPUT_AMPLITUDE_LVDS
    _ccrtaicc_cg_outmode_common_t          common
        # CCRTAICC_CG_OUTPUT_COMMON_CMOS
        # CCRTAICC_CG_OUTPUT_COMMON_LVDS
        # CCRTAICC_CG_OUTPUT_COMMON_LVPECL
Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.67 ccrtaICC_Clock_Set_Generator_Output_Mux()

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

```

/*****
    _ccrtaicc_lib_error_number_t
    ccrtaICC_Clock_Set_Generator_Output_Mux (void          *Handle,
                                             _ccrtaicc_clock_generator_output_t WhichOutput,
                                             ccrtaicc_clkgen_output_mux_t      *OutMux)

Description: Set Clock Generator Output Mux

Input:  void          *Handle      (Handle pointer)
        _ccrtaicc_clock_generator_output_t WhichOutput (select output)
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_9
        ccrtaicc_clkgen_output_mux_t      *OutMux (pointer to output
                                                    inversion/N-divider mux)
        _ccrtaicc_cg_outmux_inversion_t  inversion
        # CCRTAICC_CG_OUTPUT_MUX_COMPLEMENTARY
        # CCRTAICC_CG_OUTPUT_MUX_IN_PHASE
        # CCRTAICC_CG_OUTPUT_MUX_INVERTED
        # CCRTAICC_CG_OUTPUT_MUX_OUT_OF_PHASE
        _ccrtaicc_cg_outmux_ndiv_select_t ndiv_mux
        # CCRTAICC_CG_OUTPUT_MUX_NDIV_0
        # CCRTAICC_CG_OUTPUT_MUX_NDIV_1
        # CCRTAICC_CG_OUTPUT_MUX_NDIV_2
        # CCRTAICC_CG_OUTPUT_MUX_NDIV_3
        # CCRTAICC_CG_OUTPUT_MUX_NDIV_4
Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.68 ccrtAICC_Clock_Set_Generator_P_Divider()

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

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_Clock_Set_Generator_P_Divider (void          *Handle,
                                           _ccrtaicc_clock_generator_divider_t WhichDivider,
                                           _u64          Divider,
                                           int          Update)

Description: Set Clock Generator R-Divider

Input:  void          *Handle          (Handle pointer)
        _ccrtaicc_clock_generator_divider_t WhichDivider (select divider)
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P0
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P1
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P2
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_PFB
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_PXAXB
        _u64          Divider          (Divider)
        int          Update            (True=Update)

Output: none
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN         (device not open)
        # CCRTAICC_LIB_INVALID_ARG      (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/
```

2.2.69 ccrtAICC_Clock_Set_Generator_P_Divider_Enable()

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

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_Clock_Set_Generator_P_Divider_Enable (void          *Handle,
                                                  _ccrtaicc_clock_generator_divider_t WhichDivider,
                                                  _ccrtaicc_cg_pdiv_enable_t   Pdiv_Enable)

Description: Set Clock Generator P-Divider Enable

Input:  void          *Handle          (Handle pointer)
        _ccrtaicc_clock_generator_divider_t WhichDivider (select divider)
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P0
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P1
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_P2
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_PXAXB
        _ccrtaicc_cg_pdiv_enable_t   Pdiv_Enable (enable flag)
        # CCRTAICC_CG_PDIV_DISABLE
        # CCRTAICC_CG_PDIV_ENABLE

Output: none
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN         (device not open)
*****/
```

```

# CCRTAICC_LIB_INVALID_ARG (invalid argument)
# CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.70 ccrtAICC_Clock_Set_Generator_R_Divider()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Set_Generator_R_Divider (void *Handle,
                                         _ccrtaicc_clock_generator_divider_t WhichDivider,
                                         __u32 Divider)

Description: Set Clock Generator R-Divider

Input: void *Handle (Handle pointer)
       _ccrtaicc_clock_generator_divider_t WhichDivider (select divider)
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R0
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R1
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R2
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R3
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R4
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R5
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R6
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R7
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R8
       # CCRTAICC_CLOCK_GENERATOR_DIVIDER_R9
       __u32 Divider (Divider)

Output: none
Return: _ccrtaicc_lib_error_number_t
       # CCRTAICC_LIB_NO_ERROR (successful)
       # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
       # CCRTAICC_LIB_NOT_OPEN (device not open)
       # CCRTAICC_LIB_INVALID_ARG (invalid argument)
       # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
       # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.71 ccrtAICC_Clock_Set_Generator_Value()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Clock_Set_Generator_Value (void *Handle,
                                     int address,
                                     u_char value)

Description: Set the value of the specified Clock Generator register.

Input: void *Handle (Handle pointer)
       int address (clock gen address to set)
       u_char value

```

```

        u_char          value;          (value to write)
Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.72 ccrtaICC_Clock_Set_Generator_Voltage_Select()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_Clock_Set_Generator_Voltage_Select (void          *Handle,
                                             _ccrtaicc_cg_stat_ctrl_voltsel_t VoltSel)

Description: Set Clock Generator voltage selection

Input:  void          *Handle      (Handle pointer)
        _ccrtaicc_cg_stat_ctrl_voltsel_t VoltSel      (voltage selection)
        # CCRTAICC_CG_VOLTAGE_SELECT_1_8V
        # CCRTAICC_CG_VOLTAGE_SELECT_3_3V

Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
        # CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)

*****/

```

2.2.73 ccrtaICC_Clock_Set_Generator_Zero_Delay()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_Clock_Set_Generator_Zero_Delay (void          *Handle,
                                           _ccrtaicc_cg_zero_delay_t ZeroDelay)

Description: Set Clock Generator Zero Delay selection

Input:  void          *Handle      (Handle pointer)
        _ccrtaicc_cg_zero_delay_t      ZeroDelay      (zero delay selection)
        # CCRTAICC_CG_ZERO_DELAY_MODE
        # CCRTAICC_CG_NORMAL_MODE

Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)

*****/

```

```

# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****

```

2.2.74 **ccrtAICC_Close()**

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

```

/*****
_ccrtaicc_lib_error_number_t ccrtaICC_Close(void *Handle)

Description: Close a previously opened device.

Input:   void *Handle           (Handle pointer)
Output:  none
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR      (successful)
         # CCRTAICC_LIB_BAD_HANDLE   (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN     (device not open)
*****/

```

2.2.75 **ccrtAICC_Compute_All_Output_Clocks()**

This call does not program the clock outputs but instead returns to the user whether the board can be programmed with the user selected output clock frequencies. Additionally, useful information is returned to the user in a structure for each clock that was computed.

```

/*****

ccrtaICC_Compute_All_Output_Clocks()

Description: Compute All Output Clocks

Input:   void *Handle           (Handle pointer)
         double InputClockFrequency (Input clock frequency)
         ccrtaicc_compute_all_output_clocks_t *AllClocks (Pointer to all output clocks info)
         ccrtaicc_compute_single_output_clock_t *Clock
         long double DesiredFrequency
         double DesiredTolerancePPT
Output:  ccrtaicc_compute_all_output_clocks_t *AllClocks
         (Pointer to returned output clocks info)
         _u32 NumberOfNdividers
         ccrtaicc_compute_single_output_clock_t *Clock
         _ccrtaicc_clock_generator_output_t OutputClock
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
         # CCRTAICC_CLOCK_GENERATOR_OUTPUT_9
         double InputClockFrequency
         long double FrequencyDeviation
         int FrequencyFound
         long double ActualFrequency
         double ActualTolerancePPT
         _u64 Mdiv_Numerator
         _u32 Mdiv_Denominator
         _u64 Ndiv_Numerator
*****/

```

```

    __u32
    _ccrtaicc_cg_outmux_ndiv_select_t
    # CCRTAICC_CG_OUTPUT_MUX_NDIV_0
    # CCRTAICC_CG_OUTPUT_MUX_NDIV_1
    # CCRTAICC_CG_OUTPUT_MUX_NDIV_2
    # CCRTAICC_CG_OUTPUT_MUX_NDIV_3
    # CCRTAICC_CG_OUTPUT_MUX_NDIV_4
    __u32
    __u32
    __u32
Return:  _ccrtaicc_lib_error_number_t
    # CCRTAICC_LIB_NO_ERROR
    # CCRTAICC_LIB_BAD_HANDLE
    # CCRTAICC_LIB_NOT_OPEN
    # CCRTAICC_LIB_NO_LOCAL_REGION
    # CCRTAICC_LIB_IO_ERROR
    # CCRTAICC_LIB_N_DIVIDERS_EXCEEDED
    # CCRTAICC_LIB_CANNOT_COMPUTE_OUTPUT_FREQ
    # CCRTAICC_LIB_INVALID_ARG
*****

```

Ndiv_Denominator
 Ndiv_ToUse

 Rdiv_value
 Rdivider
 Pdivider

 (successful)
 (no/bad handler supplied)
 (library not open)
 (local region error)
 (device not ready)
 (number of N-Dividers exceeded)
 (cannot compute output freq)
 (invalid argument)

2.2.76 ccrtaICC_Convert_Physmem2avmm_Address()

This call is used to supply the user with an Avalon equivalent Address for the supplied Physical DMA memory. This Avalon equivalent address can then be supplied to the DMA engine to perform DMA operations.

```

/*****
    _ccrtaicc_lib_error_number_t
    ccrtaICC_Convert_Physmem2avmm_Address(void
    uint
    uint
    *Handle,
    *PhysDmaMemPtr,
    *AvalonAddress)

```

Description: Get the converted value of Physical DMA memory to Avalon address to be supplied as address for DMA operations.

```

Input:  void
        uint
        *Handle
        *PhysDmaMemPtr
Output: uint
        *AvalonAddress
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR
        # CCRTAICC_LIB_BAD_HANDLE
        # CCRTAICC_LIB_NOT_OPEN
        # CCRTAICC_LIB_INVALID_ARG
        # CCRTAICC_LIB_AVALON_TRANSLATION_TABLE
        # CCRTAICC_LIB_ADDRESS_RANGE_ERROR
*****

```

(Handle pointer)
 (pointer to physical DMA memory)
 (pointer to Avalon Address).
 (successful)
 (no/bad handler supplied)
 (library not open)
 (invalid argument)
 (avalon translation table error)
 (address range error)

2.2.77 ccrtaICC_Create_UserProcess()

Typically reads from h/w take a finite time to complete. If the user has a process that is time critical and needs to read the latest data faster, they may use a new approach called Hyper-Drive. In this case, the user defines a thread with this call, which continuously reads the data from the board and holds the latest values. The user process can then access this latest data at substantially faster rates. The two drawbacks to this approach is that the excessive bus access is made and dedicated CPUs are required.

This call is used to create this User Process looping thread which can be controlled by the user via the returned handle. (This is an experimental API for debugging and testing).

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_Create_UserProcess(void                *Handle,
                                _ccrtaicc_UserFunction_t *UFunc,
                                _ccrtaicc_UserFunction_t **UFuncHandle)

Description: Create a User Process for user defined processing

Input:   void                *Handle        (Handle pointer)
         _ccrtaicc_UserFunction_t *UFunc    (pointer to user
                                                information structure)

Output:  _ccrtaicc_UserFunction_t **UFuncHandle (pointer to user function
                                                struct handle)

Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR            (successful)
         # CCRTAICC_LIB_BAD_HANDLE         (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN          (device not open)
         # CCRTAICC_LIB_NO_RESOURCE       (cannot allocate memory)
         # CCRTAICC_LIB_INTERNAL_ERROR    (pthread attr failed)
         # CCRTAICC_LIB_THREAD_CREATE_FAILED (failed to create thread)
*****/

typedef struct
{
    int Magic;
    void (*UserFunction) (void *hdl);
    pthread_t UserFunction_Thread_id;
    pid_t Pid;
    pthread_mutex_t lock; /* lock this structure */
    pthread_cond_t wait; /* wait for command */
    pthread_mutex_t cmd_lock; /* lock this structure */
    pthread_cond_t cmd_wait; /* wait for command */
    pthread_mutex_t user_lock; /* lock this structure */
    pthread_cond_t user_wait; /* wait for command */
    pthread_mutex_t user_mem_lock; /* lock this structure */
    pthread_cond_t user_mem_wait; /* wait for command */
    volatile int cpuAffinity; /* CPU on which Thread
                                will run */
    volatile int cpuCount; /* no. of cpus to run on
                                starting at base */
    volatile void *Handle;
    volatile void **Args;
    volatile int SchedulePolicy;
    volatile int SchedulePriority;
    volatile int ScheduleSelf; /* 1=(Use
                                SchedulePriority-
                                1),0=no change */
    volatile ccrtaicc_uf_action_t Action;
    volatile ccrtaicc_uf_state_t State;
    volatile int CommandPending;
    volatile void *Next_UserFunction;
    volatile unsigned int long long RunCount;
    volatile int Pause;
} _ccrtaicc_UserFunction_t;

```

2.2.78 ccrtaICC_DataToVolts()

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

data supplied in *us_data* must not be greater than the hardware resolution bits *CCRTAICC_ADC_RESOLUTION_BITS* supported by the board. Data greater than this will be masked out.

```

/*****
  double ccrtAICC_DataToVolts(int us_data, ccrtAICC_volt_convert_t *conv)

  Description: Convert Data to volts

  Input:   int          us_data          (data to convert)
          ccrtAICC_volt_convert_t      *conv          (pointer to
                                                    conversion struct)
          double        VoltageRange     (maximum voltage
                                                    range)
          _ccrtAICC_csr_dataformat_t    Format         (format)
          # CCRTAICC_OFFSET_BINARY
          # CCRTAICC_TWOS_COMPLEMENT
          ccrtAICC_bool  BiPolar         (bi-polar)
          # CCRTAICC_TRUE
          # CCRTAICC_FALSE
          int            ResolutionBits  (Number of
                                                    resolution bits)

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

```

2.2.79 ccrtAICC_Destroy_AllUserProcess()

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

```

/*****
  _ccrtAICC_lib_error_number_t ccrtAICC_Destroy_AllUserProcess(void *Handle)

  Description: Destroy all created user processes

  Input:   void          *Handle          (Handle pointer)
  Output:  none
  Return:  _ccrtAICC_lib_error_number_t
          # CCRTAICC_LIB_NO_ERROR          (successful)
          # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
*****/

```

2.2.80 ccrtAICC_Destroy_UserProcess()

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

```

/*****
  _ccrtAICC_lib_error_number_t ccrtAICC_Destroy_UserProcess(void *Handle,
                                                            _ccrtAICC_UserFunction_t **UFuncHandle)

  Description: Destroy an already created user process

  Input:   void          *Handle          (Handle pointer)
          _ccrtAICC_UserFunction_t      **UFuncHandle  (pointer to user handle)
  Output:  none
  Return:  _ccrtAICC_lib_error_number_t
          # CCRTAICC_LIB_NO_ERROR          (successful)
          # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
*****/

```


2.2.81 ccrtAICC_Disable_Pci_Interrupts()

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

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_Disable_Pci_Interrupts (void                                *Handle,
                                   _ccrtaicc_all_interrupts_mask interrupt_mask)

Description: Disable interrupts being generated by the board.

Input:   void                                *Handle      (Handle pointer)
         _ccrtaicc_all_interrupts_mask      interrupt_mask (interrupt mask)
         # CCRTAICC_DMA0_INTMASK
         # CCRTAICC_DMA1_INTMASK
         # CCRTAICC_MSGDMA_INTMASK
         # CCRTAICC_ADC_FIFO_INTMASK
         # CCRTAICC_ALL_DMA_INTMASK
         # CCRTAICC_ALL_ANALOG_INTMASK
         # CCRTAICC_DMA_ANALOG_INTMASK
         # CCRTAICC_ALL_INTMASK

Output:  none
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR                (successful)
         # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN                (device not open)
         # CCRTAICC_LIB_IOCTL_FAILED            (driver ioctl call
                                                failed)
*****/
```

2.2.82 ccrtAICC_DMA_Configure()

The purpose of this call is configure a DMA engine to be ready for commencing DMA.

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_DMA_Configure(void                                *Handle,
                          ccrtaicc_dma_engine_t            DMAEngineNo,
                          uint                               AvMM_FromAddr,
                          uint                               AvMM_ToAddr,
                          uint                               DMASize)

Description: Configure DMA Engine

Input:   void                                *Handle      (Handle pointer)
         ccrtaicc_dma_engine_t DMAEngineNo (select DMA engine)
         # CCRTAICC_DMA0
         # CCRTAICC_DMA1
         uint                               AvMM_FromAddr (Avalon Memory Converted Source
                                                           Address)
         uint                               AvMM_ToAddr   (Avalon Memory Converted
                                                           Destination Address)
         uint                               DMASize        (DMA transfer size in bytes)

Output:  none
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR                (successful)
         # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN                (library not open)
         # CCRTAICC_LIB_INVALID_ARG             (invalid argument)
         # CCRTAICC_LIB_DMA_ACCESS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                                (DMA access not allowed for
                                                selected address)
*****/
```

*****/

2.2.83 ccrtAICC_DMA_Fire()

The purpose of this call is to initiate an already configured DMA engine.

/*****

```
_ccrtAICC_lib_error_number_t
ccrtAICC_DMA_Fire(void                *Handle,
                  ccrtAICC_dma_engine_t DMAEngineNo,
                  ccrtAICC_bool        UseInterrupts,
                  ccrtAICC_bool        WaitForCompletion,
                  int                   DmaControl)
```

Description: Start DMA Engine

```
Input:  void                *Handle                (Handle pointer)
        ccrtAICC_dma_engine_t DMAEngineNo        (select DMA engine)
        # CCRTAICC_DMA0
        # CCRTAICC_DMA1
        ccrtAICC_bool        UseInterrupts        (Enable Interrupt flag)
        # CCRTAICC_TRUE
        # CCRTAICC_FALSE
        ccrtAICC_bool        WaitForCompletion    (Wait for Completion Flag)
        # CCRTAICC_TRUE
        # CCRTAICC_FALSE
        int                   DmaControl          (DMA control flags)
        # CCRTAICC_DMA_CONTROL_RCON              (read constant)
        # CCRTAICC_DMA_CONTROL_WCON              (write constant)
        # CCRTAICC_DMA_CONTROL_INCREMENT         (increment)

Output: none
Return: _ccrtAICC_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                  (no error)
        # CCRTAICC_LIB_BAD_HANDLE                (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                  (library not open)
        # CCRTAICC_LIB_INVALID_ARG              (invalid argument)
        # CCRTAICC_LIB_IOCTL_FAILED             (ioctl failed)
        # CCRTAICC_LIB_DMA_FAILED               (DMA failed)
```

*****/

2.2.84 ccrtAICC_Enable_Pci_Interrupts()

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

/*****

```
_ccrtAICC_lib_error_number_t
ccrtAICC_Enable_Pci_Interrupts (void                *Handle,
                                _ccrtAICC_all_interrupts_mask interrupt_mask)
```

Description: Enable interrupts being generated by the board.

```
Input:  void                *Handle                (Handle pointer)
        _ccrtAICC_all_interrupts_mask interrupt_mask (interrupt mask)
        # CCRTAICC_DMA0_INTMASK
        # CCRTAICC_DMA1_INTMASK
        # CCRTAICC_MSGDMA_INTMASK
        # CCRTAICC_ADC_FIFO_INTMASK
        # CCRTAICC_ALL_DMA_INTMASK
        # CCRTAICC_ALL_ANALOG_INTMASK
        # CCRTAICC_DMA_ANALOG_INTMASK
        # CCRTAICC_ALL_INTMASK

Output: none
```

```

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_IOCTL_FAILED      (driver ioctl call failed)
*****/

```

2.2.85 ccrtaICC_Fast_Memcpy()

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

```

/*****
ccrtaICC_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:     _ccrtaicc_lib_error_number_t
            # CCRTAICC_LIB_NO_ERROR        (successful)
            # CCRTAICC_LIB_BAD_HANDLE      (no/bad handler supplied)
            # CCRTAICC_LIB_NOT_OPEN        (device not open)
*****/

```

2.2.86 ccrtaICC_Fast_Memcpy_Unlocked()

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

```

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

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

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

```

2.2.87 ccrtAICC_Fast_Memcpy_Unlocked_FIFO()

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

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

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

Input:  volatile void *Source           (pointer to source buffer)
        int           SizeInWords       (transfer size in words)
        int           PioControl         (PIO Control)
        # CCRTAICC_PIO_CONTROL_RCON     (read constant)
        # CCRTAICC_PIO_CONTROL_WCON     (write constant)
        # CCRTAICC_PIO_CONTROL_INCREMENT (read/write increment)
        int           DoubleWordSupport (double word support flag)
        # CCRTAICC_FALSE                 (h/w double word transfers not
        # CCRTAICC_TRUE                   supported)
        # CCRTAICC_TRUE                   (h/w double word transfers
        # CCRTAICC_TRUE                   supported)

Output: volatile void *Destination     (pointer to destination buffer)
Return: none
*/
```

2.2.88 ccrtAICC_Fraction_To_Hex()

This converts a fractional decimal to a hexadecimal value.

```
/*
int
ccrtAICC_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)
*/
```

2.2.89 ccrtAICC_Get_All_Boards_Driver_Info()

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

```
/*
ccrtAICC_Get_All_Boards_Driver_Info()
_ccrtaicc_lib_error_number_t ccrtAICC_Get_All_Boards_Driver_Info(
                               void           *Handle,
                               ccrtaicc_all_boards_driver_info *all_boards_info)

Description: Get device information from driver for all boards.
*/
```

```

Input: void *Handle (Handle pointer)
Output: ccrtaiicc_driver_info_t *all_boards_info (info struct pointer)
char version[12]
char built[32]
char module_name[16]
int board_index
int table_index
char board_desc[32]
int bus
int slot
int func
int vendor_id
int sub_vendor_id
int sub_device_id
union {
    u_int BoardInfo
    ccrtaiicc_boardinfo_t BInfo
        u_char Function
        u_char Type
        u_short Id
}
union {
    u_int FirmwareDate
    ccrtaiicc_firmware_date_t FmDate
        u_short Year
        u_char Day
        u_char Month
}
union {
    u_int FirmwareRevision
    ccrtaiicc_firmware_revision_t FmRev
        u_short Minor
        u_short Major
}
int msi_support
int irqlevel
double calibration_10v_reference_voltage
double calibration_5v_reference_voltage
int driver_dma_size

// DMA
ccrtaiicc_driver_dma_info_t dma_info
short num_trans_tbl_entries
int avalon_page_bits
int avalon_page_size
int tx_interface_base
int dma_max_engines
int dma_max_burst_size
int dma_max_transactions
int dma_max_size[CCRTAICC_DMA_MAX_ENGINES]
int dma_width_in_bytes[CCRTAICC_DMA_MAX_ENGINES]
int dma_fire_command[CCRTAICC_DMA_MAX_ENGINES]

// Interrupt
ccrtaiicc_driver_int_t interrupt
uint InterruptsOccurredMask
uint WakeupInterruptMask
int timeout_seconds
int DmaControl

long long unsigned count
long long unsigned dma_count[CCRTAICC_DMA_MAX_ENGINES]
long long unsigned MsgDma_count; // Modular Scatter-Gather DMA

```

```

int                                     Ccrtaiicc_Max_Region

// Memory Region
ccrtaiicc_dev_region_t                 mem_region[CCRTAICC_MAX_REGION]
    uint physical_address
    uint size
    uint flags
    uint *virtual_address

// ADC
ccrtaiicc_driver_adc_info_t           adc_info
    double adc_max_voltage_range
    int     number_of_adcs
    int     number_of_adc_channels
    int     number_of_adc_resolutionbits
    _ccrtaiicc_adc_channel_mask_t
        all_adc_channels_mask
    int     max_adc_fifo_threshold
    int     max_adc_high_speed_frequency
    int     max_adc_normal_speed_frequency

// SDRAM
ccrtaiicc_driver_sdrām_info_t         sdrām_info
    int                                     sdrām_max_size_in_words
    _ccrtaiicc_clock_generator_output_t sdrām_output_clock
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_9
    double                                 sdrām_output_clock_frequency

// CLOCK
ccrtaiicc_driver_clock_info_t         clock_info
    _ccrtaiicc_cg_input_clock_select_register_t default_input_clock
        - CCRTAICC_CG_INPUT_CLOCK_SELECT_IN0
        - CCRTAICC_CG_INPUT_CLOCK_SELECT_IN1
        - CCRTAICC_CG_INPUT_CLOCK_SELECT_IN2
        - CCRTAICC_CG_INPUT_CLOCK_SELECT_INXAXB
    double                                 default_input_clock_frequency
    double                                 default_clock_tolerance_ppt

// SPROM
ccrtaiicc_sprām_header_t              sprām_header
    u_int32_t board_serial_number
    u_short   sprām_revision

// Chip Temperature (this board does NOT return a chip temperature)
char                                     fpga_chip_temperature

char                                     double_word_support

union {
    u_int                                     FirmwareTime
    ccrtaiicc_firmware_time_t               FmTime
        u_char Second
        u_char Minute
        u_char Hour
}

```

```

        u_char  unused
    }
    union {
        u_int          FirmwareFlavorCode
        ccrtAICC_firmware_option_code_t  FmOptionCode
        u_char  C0
        u_char  C1
        u_char  C2
        u_char  C3
    }

    u_short          RunLevelSectorNumber
    char             FirmwareReloadFailed
    char             MultiFirmwareSupport

    union {
        u_int          Dummy_time_t[2]
        time_t         DriverLoadCurrentTime
    }

    u_int32_t        FirmwareBoardSerialNumber
    u_int32_t        MaxMsgDmaDescriptors
    u_int32_t        MaxMsgDmaSize
    u_int32_t        MsgDmaWidthInBytes
    u_int32_t        MaxMsgDmaFifoSize
    u_int32_t        FpgawbRevision
    Return:  _ccrtAICC_lib_error_number_t
            # CCRTAICC_LIB_NO_ERROR          (successful)
            # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
            # CCRTAICC_LIB_NOT_OPEN        (device not open)
            # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
            # CCRTAICC_LIB_IOCTL_FAILED    (driver ioctl call failed)
    *****/

```

2.2.90 ccrtAICC_Get_Board_CSR()

This call returns information from the board status register.

```

/*****
_ccrtAICC_lib_error_number_t
ccrtAICC_Get_Board_CSR (void          *Handle,
                        ccrtAICC_board_csr_t  *bcsr)

Description: Get Board Control and Status information

Input:  void          *Handle      (Handle pointer)
Output: ccrtAICC_board_csr_t      *bcsr      (pointer to board csr)
        _ccrtAICC_bcsr_identify_board_t  identify_board
        # CCRTAICC_BCSR_IDENTIFY_BOARD_DISABLE
        # CCRTAICC_BCSR_IDENTIFY_BOARD_ENABLE

Return:  _ccrtAICC_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN        (device not open)
        # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION  (local region not present)
*****/

```

2.2.91 ccrtAICC_Get_Board_Info()

This call returns the board id, the board type and the firmware revision level for the selected board. This board id is 0x9277 and board type is 0x1 or 0x9278 with a board type of 0x2.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Get_Board_Info (void *Handle,
                        ccrtaicc_board_info_t *binfo)
Description: Get Board Information

Input: void *Handle (Handle pointer)
Output: ccrtaicc_board_info_t *binfo (pointer to board info)
        int vendor_id
        int sub_vendor_id
        int sub_device_id
        ccrtaicc_boardinfo_t BInfo
        u_char Function
        u_char Type
        u_short Id
        ccrtaicc_firmware_date_t FmDate
        u_short Year
        u_char Day
        u_char Month
        ccrtaicc_firmware_revision_t FmRev
        u_short Minor
        u_short Major
        ccrtaicc_sprom_header_t sprom_header
        u_int32_t board_serial_number
        u_short sprom_revision
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR (successful)
        # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN (device not open)
        # CCRTAICC_LIB_INVALID_ARG (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
*****/

```

2.2.92 ccrtAICC_Get_Cable_Fault_CSR()

This call returns the cable fault and the latched cable fault status.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Get_Cable_Fault_CSR (void *Handle,
                              ccrtaicc_cable_fault_csr_t *CableFaultCSR)

Description: Get Cable Fault Control & Status information

Input: void *Handle (handle pointer)
Output: ccrtaicc_cable_fault_csr_t *CableFaultCSR (pointer to cable fault
        CSR)
        _ccrtaicc_cable_fault_group_mask_t LatchedFaultStatus
        # CCRTAICC_CABLE_FAULT_GROUP_00_03_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_04_07_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_08_11_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_12_15_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_16_19_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_20_23_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_24_27_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_28_31_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_32_35_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_36_39_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_40_43_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_44_47_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_48_51_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_52_55_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_56_59_MASK

```



```

        # CCRTAICC_CABLE_FAULT_GROUP_60_63_MASK
        # CCRTAICC_ALL_CABLE_FAULT_GROUPS_MASK
    _ccrtaicc_cable_fault_group_mask_t      FaultStatus
        # CCRTAICC_CABLE_FAULT_GROUP_00_03_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_04_07_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_08_11_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_12_15_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_16_19_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_20_23_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_24_27_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_28_31_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_32_35_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_36_39_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_40_43_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_44_47_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_48_51_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_52_55_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_56_59_MASK
        # CCRTAICC_CABLE_FAULT_GROUP_60_63_MASK
        # CCRTAICC_ALL_CABLE_FAULT_GROUPS_MASK
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_NO_LOCAL_REGION         (local region error)
        # CCRTAICC_LIB_INVALID_ARG            (invalid argument)
    *****/

```

2.2.93 ccrtaicc_Get_Calibration_CSR()

This call returns the current calibration control and status register.

```

/*****
    ccrtaicc_Get_Calibration_CSR()

    Description: Get Calibration Control and Status Register

    Input:   void                *Handle      (Handle pointer)
    Output:  ccrtaicc_calibration_csr_t *CalCSR  (pointer to calibration CSR)
            _ccrtaicc_calbus_control_t BusControl (bus control)
            # CCRTAICC_CB_GROUND
            # CCRTAICC_CB_POSITIVE_10V_REFERENCE
            # CCRTAICC_CB_NEGATIVE_10V_REFERENCE
            # CCRTAICC_CB_POSITIVE_5V_REFERENCE
            # CCRTAICC_CB_NEGATIVE_5V_REFERENCE
            # CCRTAICC_CB_POSITIVE_2V_REFERENCE
            # CCRTAICC_CB_BUS_OPEN

    Return:  _ccrtaicc_lib_error_number_t
            # CCRTAICC_LIB_NO_ERROR                (successful)
            # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
            # CCRTAICC_LIB_NOT_OPEN                (device not open)
            # CCRTAICC_LIB_INVALID_ARG            (invalid argument)
            # CCRTAICC_LIB_NO_LOCAL_REGION         (local region not present)
    *****/

```

2.2.94 ccrtaicc_Get_Driver_Error()

This call returns the last error generated by the driver.

```

/*****
    _ccrtaicc_lib_error_number_t
    ccrtaicc_Get_Driver_Error (void                *Handle,
                               ccrtaicc_user_error_t *ret_err)

```

Description: Get the last error generated by the driver.

```
Input:   void                *Handle          (Handle pointer)
Output:  ccrtaiicc_user_error_t *ret_err      (error struct pointer)
        uint error;          (error number)
        char name[CCRTAICC_ERROR_NAME_SIZE] (error name used in driver)
        char desc[CCRTAICC_ERROR_DESC_SIZE] (error description)
Return:  _ccrtaiicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR            (successful)
        # CCRTAICC_LIB_BAD_HANDLE         (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN           (device not open)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_IOCTL_FAILED      (driver ioctl call failed)
*****/

#define CCRTAICC_ERROR_NAME_SIZE    64
#define CCRTAICC_ERROR_DESC_SIZE    128

typedef struct _ccrtaiicc_user_error_t
{
    uint error;                /* error number */
    char name[CCRTAICC_ERROR_NAME_SIZE]; /* error name used in driver */
    char desc[CCRTAICC_ERROR_DESC_SIZE]; /* error description */
} ccrtaiicc_user_error_t;

enum
{
    CCRTAICC_SUCCESS = 0,
    CCRTAICC_INVALID_PARAMETER,
    CCRTAICC_DMA_TIMEOUT,
    CCRTAICC_OPERATION_CANCELLED,
    CCRTAICC_RESOURCE_ALLOCATION_ERROR,
    CCRTAICC_INVALID_REQUEST,
    CCRTAICC_FAULT_ERROR,
    CCRTAICC_BUSY,
    CCRTAICC_ADDRESS_IN_USE,
    CCRTAICC_USER_INTERRUPT_TIMEOUT,
    CCRTAICC_DMA_INCOMPLETE,
    CCRTAICC_DATA_UNDERFLOW,
    CCRTAICC_DATA_OVERFLOW,
    CCRTAICC_IO_FAILURE,
    CCRTAICC_OPERATION_NOT_SUPPORTED,
    CCRTAICC_ADC_FIFO_THRESHOLD_TIMEOUT,
    CCRTAICC_INTERRUPT_HANDLER_NOT_ENABLED,
    CCRTAICC_FIRMWARE_RELOAD_FAILED,
    CCRTAICC_DEVICE_AUTHORIZATION_FAILED,
};
```

2.2.95 ccrtAICC_Get_Driver_Info()

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

```
/******
_ccrtaiicc_lib_error_number_t
ccrtAICC_Get_Driver_Info (void                *Handle,
                        ccrtaiicc_driver_info_t *info)

Description: Get device information from driver.

Input:   void                *Handle          (Handle pointer)
Output:  ccrtaiicc_driver_info_t *info      (info struct pointer)
        char                version[12]
```

```

char          built[32]
char          module_name[16]
int           board_index
int           table_index
char          board_desc[32]
int           bus
int           slot
int           func
int           vendor_id
int           sub_vendor_id
int           sub_device_id
union {
    u_int BoardInfo
    ccrtaiicc_boardinfo_t          BInfo
        u_char Function
        u_char Type
        u_short Id
}
union {
    u_int FirmwareDate
    ccrtaiicc_firmware_date_t      FmDate
        u_short Year
        u_char Day
        u_char Month
}
union {
    u_int FirmwareRevision
    ccrtaiicc_firmware_revision_t  FmRev
        u_short Minor
        u_short Major
}
int           msi_support
int           irqlevel
double        calibration_10v_reference_voltage
double        calibration_5v_reference_voltage
int           driver_dma_size

// DMA
ccrtaiicc_driver_dma_info_t        dma_info
short   num_trans_tbl_entries
int     avalon_page_bits
int     avalon_page_size
int     tx_interface_base
int     dma_max_engines
int     dma_max_burst_size
int     dma_max_transactions
int     dma_max_size[CCRTAICC_DMA_MAX_ENGINES]
int     dma_width_in_bytes[CCRTAICC_DMA_MAX_ENGINES]
int     dma_fire_command[CCRTAICC_DMA_MAX_ENGINES]

// Interrupt
ccrtaiicc_driver_int_t            interrupt
uint   InterruptsOccurredMask
uint   WakeupInterruptMask
int     timeout_seconds
int     DmaControl

long long unsigned   count
long long unsigned   dma_count[CCRTAICC_DMA_MAX_ENGINES]
long long unsigned   MsgDma_count;           // Modular Scatter-Gather DMA

int           Ccrtaiicc_Max_Region

```

```

// Memory Region
ccrtaicc_dev_region_t          mem_region[CCRTAICC_MAX_REGION]
    uint physical_address
    uint size
    uint flags
    uint *virtual_address

// ADC
ccrtaicc_driver_adc_info_t      adc_info
    double adc_max_voltage_range
    int     number_of_adcs
    int     number_of_adc_channels
    int     number_of_adc_resolutionbits
    _ccrtaicc_adc_channel_mask_t
        all_adc_channels_mask
    int     max_adc_fifo_threshold
    int     max_adc_high_speed_frequency
    int     max_adc_normal_speed_frequency

// SDRAM
ccrtaicc_driver_sdram_info_t    sdram_info
    int     sdram_max_size_in_words
    _ccrtaicc_clock_generator_output_t sdram_output_clock
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
        - CCRTAICC_CLOCK_GENERATOR_OUTPUT_9
    double sdram_output_clock_frequency

// CLOCK
ccrtaicc_driver_clock_info_t    clock_info
    _ccrtaicc_cg_input_clock_select_register_t default_input_clock
        - CCRTAICC_CG_INPUT_CLOCK_SELECT_IN0
        - CCRTAICC_CG_INPUT_CLOCK_SELECT_IN1
        - CCRTAICC_CG_INPUT_CLOCK_SELECT_IN2
        - CCRTAICC_CG_INPUT_CLOCK_SELECT_INXAXB
    double default_input_clock_frequency
    double default_clock_tolerance_ppt

// SPROM
ccrtaicc_sprom_header_t        sprom_header
    u_int32_t board_serial_number
    u_short   sprom_revision

// Chip Temperature (this board does NOT return a chip temperature)
char fpga_chip_temperature

char double_word_support

union {
    u_int     FirmwareTime
    ccrtaicc_firmware_time_t FmTime
        u_char Second
        u_char Minute
        u_char Hour
        u_char unused
}
union {

```

```

        u_int          FirmwareFlavorCode
        ccrtaiicc_firmware_option_code_t FmOptionCode
        u_char  C0
        u_char  C1
        u_char  C2
        u_char  C3
    }

    u_short          RunLevelSectorNumber
    char             FirmwareReloadFailed
    char             MultiFirmwareSupport

    union {
        u_int          Dummy_time_t[2]
        time_t         DriverLoadCurrentTime
    }

    u_int32_t        FirmwareBoardSerialNumber
    u_int32_t        MaxMsgDmaDescriptors
    u_int32_t        MaxMsgDmaSize
    u_int32_t        MsgDmaWidthInBytes
    u_int32_t        MaxMsgDmaFifoSize
    u_int32_t        FpgawbRevision
    Return:  _ccrtaiicc_lib_error_number_t
            # CCRTAICC_LIB_NO_ERROR          (successful)
            # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
            # CCRTAICC_LIB_NOT_OPEN        (device not open)
            # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
            # CCRTAICC_LIB_IOCTL_FAILED    (driver ioctl call failed)
    *****/

```

2.2.96 ccrtAICC_Get_External_Clock_CSR()

This call returns the current external clock connections.

```

/*****
    _ccrtaiicc_lib_error_number_t
    ccrtAICC_Get_External_Clock_CSR (void          *Handle,
                                     ccrtaiicc_external_clock_csr_t *ExtClkCSR)

    Description: Get External Clock Control & Status information

    Input:  void          *Handle          (handle pointer)
    Output: ccrtaiicc_external_clock_csr_t *ExtClkCSR (pointer to external clock
                                                    CSR)

            ccrtaiicc_bool          InputClockPresent
            ccrtaiicc_bool          OutputClockPresent
            _ccrtaiicc_external_clock_output_select_t ClockOutputSelect;
            # CCRTAICC_EXCLOS_CLOCK_GENERATOR_OUTPUT_0
            # CCRTAICC_EXCLOS_CLOCK_GENERATOR_OUTPUT_1
            # CCRTAICC_EXCLOS_CLOCK_GENERATOR_OUTPUT_2
            # CCRTAICC_EXCLOS_CLOCK_GENERATOR_OUTPUT_3
            # CCRTAICC_EXCLOS_EXTERNAL_CLOCK_INPUT
            # CCRTAICC_EXCLOS_NO_CLOCK

    Return:  _ccrtaiicc_lib_error_number_t
            # CCRTAICC_LIB_NO_ERROR          (successful)
            # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
            # CCRTAICC_LIB_NOT_OPEN        (device not open)
            # CCRTAICC_LIB_NO_LOCAL_REGION (local region error)
            # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
    *****/

```

2.2.97 ccrtAICC_Get_Interrupt_Status()

This call returns the current status of the various interrupts.

```

/*****
  _ccrtAICC_lib_error_number_t
  ccrtAICC_Get_Interrupt_Status (void          *Handle,
                                ccrtAICC_interrupt_t *intr)

  Description: Get Interrupt Status information

  Input:   void          *Handle          (handle pointer)
  Output:  ccrtAICC_interrupt_t *intr      (pointer to interrupt status)
           _ccrtAICC_intsta_adc_t
           # CCRTAICC_INT_ADC_FIFO_THRESHOLD_NONE
           # CCRTAICC_INT_ADC_FIFO_THRESHOLD_OCCURRED
  Return:  _ccrtAICC_lib_error_number_t
           # CCRTAICC_LIB_NO_ERROR          (successful)
           # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
           # CCRTAICC_LIB_NOT_OPEN         (device not open)
           # CCRTAICC_LIB_NO_LOCAL_REGION   (local region error)
           # CCRTAICC_LIB_INVALID_ARG      (invalid argument)
  *****/

```

2.2.98 ccrtAICC_Get_Interrupt_Timeout_Seconds()

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

```

/*****
  _ccrtAICC_lib_error_number_t
  ccrtAICC_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:  _ccrtAICC_lib_error_number_t
           # CCRTAICC_LIB_NO_ERROR          (successful)
           # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
           # CCRTAICC_LIB_NOT_OPEN         (device not open)
           # CCRTAICC_LIB_INVALID_ARG      (invalid argument)
           # CCRTAICC_LIB_NO_LOCAL_REGION   (local region not present)
           # CCRTAICC_LIB_IOCTL_FAILED     (ioctl error)
  *****/

```

2.2.99 ccrtAICC_Get_Lib_Error()

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

```

/*****
  _ccrtAICC_lib_error_number_t
  ccrtAICC_Get_Lib_Error (void          *Handle,
                          ccrtAICC_lib_error_t *lib_error)

  Description: Get last error generated by the library.

  Input:   void          *Handle          (Handle pointer)

```

```

Output:  ccrtaiicc_lib_error_t *lib_error          (error struct pointer)
         uint error                               (last error number)
         char name[CCRTAICC_LIB_ERROR_NAME_SIZE] (last error name)
         char desc[CCRTAICC_LIB_ERROR_DESC_SIZE] (last error description)
         int  line_number                         (last error line number in
                                                lib)
         char function[CCRTAICC_LIB_ERROR_FUNC_SIZE]
                                                (library function in
                                                error)

Return:  _ccrtaiicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR                 (successful)
         # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN                (device not open)
         # CCRTAICC_LIB_AUTHORIZATION_FAILURE   (device authorization failure)
*****
typedef struct _ccrtaiicc_lib_error_t {
    uint    error;                               /* lib error number */
    char    name[CCRTAICC_LIB_ERROR_NAME_SIZE]; /* error name used in lib */
    char    desc[CCRTAICC_LIB_ERROR_DESC_SIZE]; /* error description */
    int     line_number;                         /* line number in library */
    char    function[CCRTAICC_LIB_ERROR_FUNC_SIZE];
                                                /* library function */
} ccrtaiicc_lib_error_t;

```

Possible library errors:

```

CCRTAICC_LIB_NO_ERROR                = 0, // Successful
CCRTAICC_LIB_INVALID_ARG             = -1, // Invalid argument
CCRTAICC_LIB_ALREADY_OPEN            = -2, // Already open
CCRTAICC_LIB_OPEN_FAILED              = -3, // Open failed
CCRTAICC_LIB_BAD_HANDLE               = -4, // Bad handle
CCRTAICC_LIB_NOT_OPEN                 = -5, // Device not opened
CCRTAICC_LIB_MMAP_SELECT_FAILED       = -6, // Mmap selection failed
CCRTAICC_LIB_MMAP_FAILED              = -7, // Mmap failed
CCRTAICC_LIB_MUNMAP_FAILED            = -8, // Munmap failed
CCRTAICC_LIB_NOT_MAPPED               = -9, // Not mapped
CCRTAICC_LIB_ALREADY_MAPPED           = -10, // Device already mapped
CCRTAICC_LIB_IOCTL_FAILED             = -11, // Device IOCTL failed
CCRTAICC_LIB_IO_ERROR                 = -12, // I/O error
CCRTAICC_LIB_INTERNAL_ERROR           = -13, // Internal library error
CCRTAICC_LIB_NOT_IMPLEMENTED          = -14, // Call not implemented
CCRTAICC_LIB_LOCK_FAILED              = -15, // Failed to get lib lock
CCRTAICC_LIB_NO_LOCAL_REGION           = -16, // Local region not present
CCRTAICC_LIB_NO_CONFIG_REGION         = -17, // Config region not present
CCRTAICC_LIB_NO_SOLUTION_FOUND        = -18, // No solution found
CCRTAICC_LIB_NO_RESOURCE               = -19, // Resource not available
CCRTAICC_LIB_CANNOT_OPEN_FILE         = -20, // Cannot open file
CCRTAICC_LIB_DMA_BUSY                 = -21, // DMA busy
CCRTAICC_LIB_AVALON_TRANSLATION_TABLE = -22, // Avalon translation table
error
CCRTAICC_LIB_ADDRESS_RANGE_ERROR      = -23, // Physical DMA address
exceeds memory size
CCRTAICC_LIB_NO_SPACE_IN_TABLE        = -24, // No space available to
allocate any more physical
memory
CCRTAICC_LIB_CANNOT_ALLOCATE_PHYS_MEM = -25, // Cannot allocate physical
memory
CCRTAICC_LIB_DMA_FAILED                = -26, // DMA failed
CCRTAICC_LIB_THREAD_CREATE_FAILED     = -27, // Thread Creation failed
CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE      = -28, // Clock Generator is not
active
CCRTAICC_LIB_CANNOT_COMPUTE_OUTPUT_FREQ = -29, // Cannot compute output

```

```

frequency
CCRTAICC_LIB_N_DIVIDERS_EXCEEDED      = -30, // Number of N-Dividers
exceeded
CCRTAICC_LIB_CLOCK_GENERATION_FAILED  = -31, // Clock generation failed
CCRTAICC_LIB_CALIBRATION_RANGE_ERROR  = -32, // Calibration voltage out of
range
CCRTAICC_LIB_BAD_DATA_IN_CAL_FILE     = -33, // Bad data in calibration
file
CCRTAICC_LIB_VOLTAGE_NOT_IN_RANGE     = -34, // Voltage not in range
CCRTAICC_LIB_ADC_IS_NOT_ACTIVE        = -35, // ADC is not active
CCRTAICC_LIB_SDRAM_IS_NOT_ACTIVE      = -36, // SDRAM is not active
CCRTAICC_LIB_SDRAM_INITIALIZATION_FAILED = -37, // SDRAM initialization failed
CCRTAICC_LIB_SERIAL_PROM_FAILURE     = -38, // Serial PROM failure -
malfunction or not present
CCRTAICC_LIB_SERIAL_PROM_BUSY        = -39, // Serial PROM busy
CCRTAICC_LIB_SERIAL_PROM_WRITE_PROTECTED = -40, // Serial PROM is write
protected
CCRTAICC_LIB_AUTHORIZATION_FAILURE    = -41, // Failure to authorize
opening of device
CCRTAICC_LIB_INTHDLR_CREATE_FAILURE   = -42, // Interrupt handler creation
failure
CCRTAICC_LIB_INTHDLR_ALREADY_RUNNING  = -43, // Interrupt handler already
running
CCRTAICC_LIB_NO_FREE_DESCRIPTOR_AVAILABLE = -44, // No free descriptors
available
CCRTAICC_LIB_ERROR_IN_DESCRIPTOR_LIST  = -45, // Error in descriptor list
CCRTAICC_LIB_MSGDMA_NOT_SUPPORTED     = -46, // Modular Scatter-Gather DMA
not supported
CCRTAICC_LIB_MSGDMA_ACCESS_NOT_ALLOWED_FOR_SELECTED_ADDRESS = -47, // MSGDMA access not allowed
for selected address
CCRTAICC_LIB_NOT_OWNER_OF_MSGDMA      = -48, // Not Owner of Modular
Scatter-Gather DMA
CCRTAICC_LIB_MSGDMA_IN_USE            = -49, // Modular Scatter-Gather DMA
In Use
CCRTAICC_LIB_CLOCK_NOT_ASSIGNED_TO_ADC = -50, // Clock generator is not
assigned to an ADC
CCRTAICC_LIB_SERIAL_PROM_NOT_PRESENT  = -51, // Serial PROM not present
CCRTAICC_LIB_DMA_ACCESS_NOT_ALLOWED_FOR_SELECTED_ADDRESS = -52, // DMA access not allowed for
selected address
CCRTAICC_LIB_SDRAM_NOT_SUPPORTED      = -53, // SDRAM not supported

```

2.2.100 ccrtAICC_Get_Library_Info()

This call returns useful library information to the user.

```

/*****

```

```

    _ccrtAICC_lib_error_number_t
    ccrtAICC_Get_Library_Info (void          *Handle,
                               ccrtAICC_library_info_t *info)

```

Description: Get library information

```

Input:   void          *Handle      (Handle pointer)
Output:  ccrtAICC_library_info_t   *info      (info struct pointer)
        int           fp;

        ccrtAICC_local_ctrl_data_t *local_ptr;
        -- structure in ccrtAICC_user.h
void
int     *munmap_local_ptr;
        local_mmap_size;

```



```

ccrtaicc_config_local_data_t      *config_ptr;
    -- structure in ccrtaicc_user.h
void                               *munmap_config_ptr;
int                               config_mmap_size;

ccrtaicc_user_phys_mem_t
    PhysMem[CCRTAICC_MAX_AVALON_NUM_TRANS_TBL_ENTRIES];
    -- structure in ccrtaicc_user.h

ccrtaicc_driver_library_common_t  *driver_lib_ptr;
    -- structure in ccrtaicc_user.h
void                               *munmap_driver_lib_ptr;
int                               driver_lib_mmap_size;
uint                              UserPid;

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN         (device not open)
        # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
*****/

```

2.2.101 ccrtaICC_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 *ccrtaicc_user.h* include file that is supplied with the driver.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_Get_Mapped_Config_Ptr (void          *Handle,
                               ccrtaicc_config_local_data_t  **config_ptr)

Description: Get mapped configuration pointer.

Input:  void          *Handle          (Handle pointer)
Output: ccrtaicc_config_local_data_t  **config_ptr      (config struct ptr)
        -- structure in ccrtaicc_user.h
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN         (device not open)
        # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
        # CCRTAICC_LIB_NO_CONFIG_REGION (config region not present)
*****/

```

2.2.102 ccrtaICC_Get_Mapped_Driver_Library_Ptr()

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

```

/*****
ccrtaICC_Get_Mapped_Driver_Library_Ptr()
_ccrtaicc_lib_error_number_t
ccrtaICC_Get_Mapped_Driver_Library_Ptr (void          *Handle,
                                       ccrtaicc_driver_library_common_t  **driver_lib_ptr)

Description: Get mapped Driver/Library structure pointer.

```

```

Input:  void                                *Handle          (Handle pointer)
Output: ccrtaiicc_driver_library_common_t **driver_lib_ptr (driver_lib
                                             struct ptr)

        -- structure in ccrtaiicc_user.h
Return: _ccrtaiicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR              (successful)
        # CCRTAICC_LIB_BAD_HANDLE           (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN             (device not open)
        # CCRTAICC_LIB_INVALID_ARG         (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION      (local region not present)

*****/

```

2.2.103 ccrtAICC_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 *ccrtaiicc_user.h* include file that is supplied with the driver.

```

/*****
_ccrtaiicc_lib_error_number_t
ccrtAICC_Get_Mapped_Local_Ptr (void          *Handle,
                               ccrtaiicc_local_ctrl_data_t **local_ptr)

Description: Get mapped local pointer.

Input:  void                                *Handle          (Handle pointer)
Output: ccrtaiicc_local_ctrl_data_t **local_ptr (local struct ptr)
        -- structure in ccrtaiicc_user.h
Return: _ccrtaiicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR              (successful)
        # CCRTAICC_LIB_BAD_HANDLE           (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN             (device not open)
        # CCRTAICC_LIB_INVALID_ARG         (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION      (local region not present)

*****/

```

2.2.104 ccrtAICC_Get_Open_File_Descriptor()

When the library *ccrtAICC_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.

```

/*****
_ccrtaiicc_lib_error_number_t
ccrtAICC_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: _ccrtaiicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR              (successful)

*****/

```

```

# CCRTAICC_LIB_BAD_HANDLE      (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN        (device not open)
# CCRTAICC_LIB_INVALID_ARG     (invalid argument)
*****/

```

2.2.105 ccrtAICC_Get_Physical_Memory()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Get_Physical_Memory (void          *Handle,
                             ccrtaicc_user_phys_mem_t *phys_mem)

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

Input:   void          *Handle          (Handle pointer)
         ccrtaicc_user_phys_mem_t *phys_mem      (mem struct pointer)
         void          *mmaped_user_mem_ptr (mmaped user virtual
                                         memory)

         uint          entry_num_in_tran_table
                                         (entry number in translation table)

Output:  ccrtaicc_user_phys_mem_t *phys_mem      (mem struct pointer)
         uint          user_pid
         void          *phys_mem_ptr
         void          *driver_virt_mem_ptr
         void          *mmaped_user_mem_ptr
         uint          phys_mem_size
         uint          phys_mem_size_freed
         uint          entry_num_in_tran_table
         uint          num_of_entries_used

Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR          (successful)
         # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN         (device not open)
         # CCRTAICC_LIB_INVALID_ARG      (invalid argument)
         # CCRTAICC_LIB_IOCTL_FAILED     (driver ioctl call failed)
*****/

```

2.2.106 ccrtAICC_Get_RunCount_UserProcess()

This call returns to the user a count of the number of times the User Process has entered.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Get_RunCount_UserProcess(void          *UFuncHandle,
                                   unsigned int long long *RunCount)

Description: Get run count in user process

Input:   void          *UFuncHandle (UF Handle pointer))
Output:  unsigned int long long *RunCount (pointer to run count)
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR          (successful)
         # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
*****/

```

2.2.107 ccrtAICC_Get_TestBus_Control()

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

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_Get_TestBus_Control (void          *Handle,
                               _ccrtaicc_testbus_control_t *test_control)

  Description: Return the value of the Test Bus control information

  Input:   void          *Handle          (handle pointer)
  Output:  _ccrtaicc_testbus_control_t
          *test_control (pointer to control select)
          # CCRTAICC_TBUS_CONTROL_OPEN
          # CCRTAICC_TBUS_CONTROL_CAL_BUS
  Return:  _ccrtaicc_lib_error_number_t
          # CCRTAICC_LIB_NO_ERROR          (successful)
          # CCRTAICC_LIB_NO_LOCAL_REGION   (local region error)
          # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
          # CCRTAICC_LIB_NOT_OPEN        (device not open)
  *****/

```

2.2.108 ccrtAICC_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.

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_Get_Value (void          *Handle,
                    CCRTAICC_CONTROL cmd,
                    void          *value)

  Description: Return the value of the specified board register.

  Input:   void          *Handle          (Handle pointer)
          CCRTAICC_CONTROL cmd          (register definition)
          -- structure in ccrtaicc_lib.h
  Output:  void          *value;         (pointer to value)
  Return:  _ccrtaicc_lib_error_number_t
          # CCRTAICC_LIB_NO_ERROR          (successful)
          # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
          # CCRTAICC_LIB_NOT_OPEN        (device not open)
          # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
          # CCRTAICC_LIB_NO_LOCAL_REGION  (local region not present)
  *****/

```

2.2.109 ccrtAICC_Hex_To_Fraction()

This call converts a hexadecimal value to a fractional decimal.

```

/*****
  double
  ccrtAICC_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)
  *****/

```

2.2.110 ccrtAICC_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.

```
/*
_ccrtaicc_lib_error_number_t
ccrtAICC_Identify_Board (void          *Handle,
                        _ccrtaicc_identify_t Identify)

Description: Identify the board by setting the front LED

Input:  void          *Handle          (Handle pointer)
        _ccrtaicc_identify_t Identify (Identify board settings)
        # CCRTAICC_IDENTIFY_OFF      (turn off flashing)
        # CCRTAICC_IDENTIFY_ON       (turn on flashing)
        # Number of seconds to flash

Output: none
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR       (successful)
        # CCRTAICC_LIB_BAD_HANDLE     (no/bad handler supplied)
        # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTAICC_LIB_NOT_OPEN       (device not open)
        # CCRTAICC_LIB_INVALID_ARG    (invalid argument)
*/
```

2.2.111 ccrtAICC_Initialize_Board()

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

```
/*
_ccrtaicc_lib_error_number_t
ccrtAICC_Initialize_Board (void *Handle)

Description: Initialize the board.

Input:  void          *Handle          (Handle pointer)
Output: none
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR       (successful)
        # CCRTAICC_LIB_BAD_HANDLE     (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN       (device not open)
        # CCRTAICC_LIB_IOCTL_FAILED   (driver ioctl call failed)
        # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
*/
```

2.2.112 ccrtAICC_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 is not available, this call will fail, at which point the user will need to issue the *ccrtAICC_Munmap_Physical_Memory()* API call to remove any previously allocated physical memory.

```
/*
_ccrtaicc_lib_error_number_t
ccrtAICC_MMap_Physical_Memory (void          *Handle,
                               int           size,
                               ccrtaicc_user_phys_mem_t *phys_mem)

Description: Allocate a physical DMA memory for size bytes.

Input:  void          *Handle          (Handle pointer)
        int           size            (size in bytes)
        ccrtaicc_user_phys_mem_t *phys_mem
*/
```

```

Output:  ccrtaiicc_user_phys_mem_t *phys_mem          (mem struct pointer)
        uint                      user_pid
        void                      *phys_mem_ptr
        void                      *driver_virt_mem_ptr
        void                      *mmaped_user_mem_ptr
        uint                      phys_mem_size
        uint                      phys_mem_size_freed
        uint                      entry_num_in_tran_table
        uint                      num_of_entries_used

Return:  _ccrtaiicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN        (device not open)
        # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
        # CCRTAICC_LIB_MMAP_SELECT_FAILED (mmap selection failed)
        # CCRTAICC_LIB_MMAP_FAILED     (mmap failed)
        # CCRTAICC_LIB_NO_SPACE_IN_TABLE (no space in phys memory table)
*****/

```

2.2.113 ccrtAICC_MsgDma_Configure_ADC_Fifo()

This call in conjunction with the *ccrtaiicc_MsgDma_Fire_Adc_Fifo()* API provides a fast method to extract samples that are present in the ADC Fifo. Due to the capability of this card to support a large number of channels at a very high rate, it is recommended to use this modular scatter-gather DMA instead of the regular DMA engine to perform the sample extraction. This configuration API need only be called once with a destination address and number of samples to be collected. As long as the configuration has not changed, the user can continuously collect samples from the ADC with help of the *ccrtaiicc_MsgDma_Fire_Adc_Fifo()* API at very high transfer rates. If the user wishes to change the number of samples extracted, they will need to reissue this configuration API with the understanding that the overhead to setup the modular scatter-gather DMA engine is significant.

```

/*****
_ccrtaiicc_lib_error_number_t
ccrtAICC_MsgDma_Configure_ADC_Fifo (void      *Handle,
                                   void      *PciDmaMemory,
                                   uint      NumberOfSamples,
                                   _ccrtaiicc_msgdma_descriptors_id_t
                                   *LastDescriptorId)

```

Description: Configure Modular Scatter-Gather MSG DMA ADC Fifo descriptor

```

Input:  void      *Handle          (Handle pointer)
        void      *PciDmaMemory   (Virtual PCI DMA Memory pointer)
        uint      NumberOfSamples  (number of FIFO samples to read)

```

```

Output:  _ccrtaiicc_msgdma_descriptors_id_t
        *LastDescriptorId (pointer to last descriptor id)

```

```

Return:  _ccrtaiicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN        (device not open)
        # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
        # CCRTAICC_LIB_NO_FREE_DESCRIPTOR_AVAILABLE
                                          (no free descriptors
                                          available)
        # CCRTAICC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA
                                          not supported)
        # CCRTAICC_LIB_NOT_OWNER_OF_MSGDMA (not owner of modular
                                          scatter-gather)

```

```

*****/

```

2.2.114 ccrtAICC_MsgDma_Configure_Descriptor()

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

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

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

DMA transfers can occur from either of the following:

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

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

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

```
/*
_ccrtaicc_lib_error_number_t
ccrtAICC_MsgDma_Configure_Descriptor (void *Handle,
_ccrtaicc_msgdma_descriptors_id_t *DescriptorID,
_ccrtaicc_msgdma_descriptor_t *Descriptor,
_ccrtaicc_msgdma_descriptors_id_t AttachToDescriptorID)
```

Description: Configure Modular Scatter-Gather DMA descriptor

```
Input: void *Handle (Handle pointer)
_ccrtaicc_msgdma_descriptors_id_t *DescriptorID
      (Set to NULL or valid ID)
      # 0 (let function find a free ID)
      # CCRTAICC_MSGDMA_DESCRIPTOR_ID_1 ...
      CCRTAICC_MSGDMA_DESCRIPTOR_ID_31
_ccrtaicc_msgdma_descriptor_t *Descriptor (pointer to descriptor)
      __u64 ReadAddress
      __u64 WriteAddress
      __u32 Length
_ccrtaicc_msgdma_descriptors_id_t AttachToDescriptorID
      (Attach to descriptor ID)
      # CCRTAICC_MSGDMA_DESCRIPTOR_ID_1 ...
      CCRTAICC_MSGDMA_DESCRIPTOR_ID_31
Output: _ccrtaicc_msgdma_descriptors_id_t *DescriptorID (returned ID)
```

```

        # CCRTAICC_MSGDMA_DESCRIPTOR_ID_1 ...
        CCRTAICC_MSGDMA_DESCRIPTOR_ID_31
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_INVALID_ARG             (invalid argument)
        # CCRTAICC_LIB_NO_FREE_DESCRIPTOR_AVAILABLE
                                                (no free descriptors available)
        # CCRTAICC_LIB_MSG_DMA_NOT_SUPPORTED (modular scatter-gather
                                                DMA not supported)
        # CCRTAICC_LIB_MSG_DMA_READS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                                (MSG DMA Reads not allowed
                                                for selected address)
*****/

```

2.2.115 ccrtaICC_MsgDma_Configure_Single()

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

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

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

```

/*****
   _ccrtaicc_lib_error_number_t
   ccrtaICC_MsgDma_Configure_Single (void                *Handle,
                                     ccrtaicc_msgdma_descriptor_t *Descriptor)

Description: Configure Single Modular Scatter-Gather DMA descriptor

Input:   void                *Handle (Handle pointer)
         ccrtaicc_msgdma_descriptor_t *Descriptor (pointer to descriptor)
         __u64 ReadAddress
         __u64 WriteAddress
         __u32 Length

Output:  none

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_INVALID_ARG             (invalid argument)
        # CCRTAICC_LIB_DMA_BUSY                (DMA Busy, cannot be reset)
        # CCRTAICC_LIB_MSG_DMA_NOT_SUPPORTED (modular scatter-gather DMA
                                                not supported)
        # CCRTAICC_LIB_MSG_DMA_READS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
                                                (MSG DMA Reads not allowed
                                                for selected address)
*****/

```


2.2.116 ccrtAICC_MsgDma_Fire()

This call commences a scatter-gather DMA operation that has been previously configured and setup by the *ccrtAICC_MsgDma_Configure()* and *ccrtAICC_MsgDma_Setup()* call.

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

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

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

- CCRTAICC_MSGD_DESC_CONTROL_GO
- CCRTAICC_MSGD_DESC_CONTROL_OWNED_BY_HW

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

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

```
/*
*****
_ccrtaicc_lib_error_number_t
_ccrtaicc_MsgDma_Fire (void *Handle,
                        _ccrtaicc_msgdma_descriptors_id_t StartDescriptorID,
                        _ccrtaicc_msgdma_descriptors_id_mask_t DescriptorIDMask,
                        int ControlWord,
                        _ccrtaicc_msgdma_descriptors_id_t
                        LastIdForInterrupts)
```

Description: Fire Modular Scatter-Gather DMA descriptor

```
Input: void *Handle (Handle pointer)
        _ccrtaicc_msgdma_descriptors_id_t StartDescriptorID (Set to
        # 0 valid ID)
        # CCRTAICC_MSGDMA_DESCRIPTOR_ID_1 ... (don't set start descriptor ID
        CCRTAICC_MSGDMA_DESCRIPTOR_ID_31 in prefetcher)
        _ccrtaicc_msgdma_descriptors_id_mask_t DescriptorIDMask
        # CCRTAICC_MSGDMA_DESCRIPTOR_ID_1_MASK ... (descriptor ID mask)
        CCRTAICC_MSGDMA_DESCRIPTOR_ID_31_MASK
        # CCRTAICC_MSGDMA_DESCRIPTOR_ID_ALL_MASK
        int ControlWord
        # CCRTAICC_MSGD_DESC_CONTROL_GO
        # CCRTAICC_MSGD_DESC_CONTROL_OWNED_BY_HW
        _ccrtaicc_msgdma_descriptors_id_t LastIdForInterrupts (Set 0 or
```

```

                                                    Last ID for interrupts)
# 0
# CCRTAICC_MSGDMA_DESCRIPTOR_ID_1 ...
  CCRTAICC_MSGDMA_DESCRIPTOR_ID_31
Output: none
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR           (successful)
         # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
         # CCRTAICC_LIB_DMA_FAILED        (DMA failed)
         # CCRTAICC_LIB_DMA_BUSY          (DMA busy)
*****/

```

2.2.117 **ccrtaICC_MsgDma_Fire_ADC_Fifo()**

Once the user has configured the modular scatter-gather DMA engine for ADC Fifo extraction using the *ccrtaICC_MsgDma_Configure_ADC_Fifo()* API, the user will need to use this *ccrtaICC_MsgDma_Fire_ADC_Fifo()* API to extract the samples that have collected in the ADC. Prior to issuing this call, the user will need to ensure that sufficient samples have been collected in the FIFO, otherwise, the call will return an empty FIFO with *0xBAADBEEF* as its data. Once sufficient samples have collected in the FIFO, the user will need to immediately invoke the *ccrtaICC_MsgDma_Fire_ADC_Fifo()* API, otherwise it is possible that an overflow condition would occur and samples would be lost. At this point, the user will need to reset the FIFO (clear the samples) and resume data collection operation once again.

2.2.118 **ccrtaICC_MsgDma_Fire_Single()**

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

```

/*****
  _ccrtaicc_lib_error_number_t
  _ccrtaICC_MsgDma_Fire_Single (void *Handle,
                               int UseInterrupts)

Description: Fire Single Modular Scatter-Gather DMA descriptor

Input:   void *Handle           (Handle pointer)
         int UseInterrupts      (Use interrupts flag)
         # CCRTAICC_TRUE
         # CCRTAICC_FALSE

Output:  none
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR           (successful)
         # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
         # CCRTAICC_LIB_DMA_FAILED        (DMA failed)
         # CCRTAICC_LIB_DMA_BUSY          (DMA busy)
*****/

```

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

2.2.119 **ccrtaICC_MsgDma_Free_Descriptor()**

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

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtaICC_MsgDma_Free_Descriptor (void *Handle,
                                   _ccrtaicc_msgdma_descriptors_id_mask_t DescriptorIDMask)

```

Description: Free Modular Scatter-Gather DMA descriptor

```
Input:   void                                     *Handle (Handle pointer)
        _ccrtaicc_msgdma_descriptors_id_mask_t DescriptorIDMask
                                                (descriptor ID mask)
        # CCRTAICC_MSGDMA_DESCRIPTOR_ID_1_MASK ...
        CCRTAICC_MSGDMA_DESCRIPTOR_ID_31_MASK
        # CCRTAICC_MSGDMA_DESCRIPTOR_ID_ALL_MASK

Output:  none

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE             (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN              (device not open)
        # CCRTAICC_LIB_INVALID_ARG           (invalid argument)
        # CCRTAICC_LIB_MSG_DMA_NOT_SUPPORTED (modular scatter-gather DMA
                                                not supported)

*****/
```

2.2.120 ccrtaICC_MsgDma_Get_Descriptor()

This call returns information on the selected descriptor.

```
/*
   _ccrtaicc_lib_error_number_t
   ccrtaICC_MsgDma_Get_Descriptor (void                                     *Handle,
                                   _ccrtaicc_msgdma_descriptors_id_t DescriptorID,
                                   ccrtaicc_msgdma_descriptor_t         *Descriptor,
                                   __u64                                 *DescriptorAddress)
*/
```

Description: Get Modular Scatter-Gather DMA Descriptor

```
Input:   void                                     *Handle (Handle pointer)
        _ccrtaicc_msgdma_descriptors_id_t DescriptorID (descriptor ID)
        # CCRTAICC_MSGDMA_DESCRIPTOR_ID_1 ...
        CCRTAICC_MSGDMA_DESCRIPTOR_ID_31

Output:  ccrtaicc_msgdma_descriptor_t *Descriptor (pointer to descriptor)
        __u64 ReadAddress
        __u64 WriteAddress
        __u32 Length
        __u32 Control
        __u32 ReadBurstCount
        __u32 WriteBurstCount
        __u32 ReadStride
        __u32 WriteStride
        __u32 ActualBytesTransferred
        __u32 Status
        __u64 *DescriptorAddress (descriptor address)

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE             (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN              (device not open)
        # CCRTAICC_LIB_INVALID_ARG           (invalid argument)
        # CCRTAICC_LIB_MSG_DMA_NOT_SUPPORTED (modular scatter-gather DMA
                                                not supported)

*****/
```

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

2.2.121 ccrtaICC_MsgDma_Get_Dispatcher_CSR()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_MsgDma_Get_Dispatcher_CSR (void                                *Handle,
                                   ccrtaicc_msgdma_dispatcher_t      *Dispatcher)

Description: Get Modular Scatter-Gather DMA Dispatcher CSR

Input:   void                                *Handle (Handle pointer)
Output:  ccrtaicc_msgdma_dispatcher_t      *Dispatcher (pointer to dispatcher)
        __u32 Status
            # CCRTAICC_MSGD_DISP_STATUS_IRQ                :IRQ
            # CCRTAICC_MSGD_DISP_STATUS_STOPPED_ETERM      :Stopped on Early
                Termination
            # CCRTAICC_MSGD_DISP_STATUS_STOPPED_ERROR      :Stopped on Error
            # CCRTAICC_MSGD_DISP_STATUS_RESETTING         :Resetting
            # CCRTAICC_MSGD_DISP_STATUS_STOPPED            :Stopped
            # CCRTAICC_MSGD_DISP_STATUS_RESP_BUF_FULL      :Response Buffer
                Full
            # CCRTAICC_MSGD_DISP_STATUS_RESP_BUF_EMPTY    :Response Buffer
                Empty
            # CCRTAICC_MSGD_DISP_STATUS_DESC_BUF_FULL      :Descriptor Buffer
                Full
            # CCRTAICC_MSGD_DISP_STATUS_DESC_BUF_EMPTY    :Descriptor Buffer
                Empty
            # CCRTAICC_MSGD_DISP_STATUS_BUSY               :Busy
        __u32 Control
            # CCRTAICC_MSGD_DISP_CONTROL_STOP_DESC        :Stop Descriptors
            # CCRTAICC_MSGD_DISP_CONTROL_INT_ENA_MASK     :Global Interrupt
                Enable Mask
            # CCRTAICC_MSGD_DISP_CONTROL_STOP_ETERM       :Stop on Early
                Termination
            # CCRTAICC_MSGD_DISP_CONTROL_STOP_ON_ERROR    :Stop on Error
            # CCRTAICC_MSGD_DISP_CONTROL_RESET_DISP      :Reset Dispatcher
            # CCRTAICC_MSGD_DISP_CONTROL_STOP_DISP        :Stop Dispatcher
        __u32 ReadFillLevel
        __u32 WriteFillLevel
        __u32 ResponseFillLevel
        __u32 ReadSequenceNumber
        __u32 WriteSequenceNumber
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                          (successful)
        # CCRTAICC_LIB_BAD_HANDLE                        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                          (device not open)
        # CCRTAICC_LIB_INVALID_ARG                       (invalid argument)
        # CCRTAICC_LIB_MSG_DMA_NOT_SUPPORTED             (modular scatter-gather DMA
                not supported)
*****/

```

2.2.122 ccrtAICC_MsgDma_Get_Prefetcher_CSR()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_MsgDma_Get_Prefetcher_CSR (void                                *Handle,
                                   ccrtaicc_msgdma_prefetcher_t      *Prefetcher)

Description: Get Modular Scatter-Gather DMA Prefetcher CSR

Input:   void                                *Handle (Handle pointer)
Output:  ccrtaicc_msgdma_prefetcher_t      *Prefetcher (pointer to prefetcher)
        __u32 Status
            # CCRTAICC_MSGD_PREF_STATUS_IRQ                :IRQ Occurred

```

```

    __u32 Control
    # CCRTAICC_MSGD_PREF_CONTROL_PARK_MODE      :Park Mode
    # CCRTAICC_MSGD_PREF_CONTROL_INT_ENA_MASK   :Global Interrupt
                                                Enable Mask
    # CCRTAICC_MSGD_PREF_CONTROL_RESET          :Reset Prefetcher
                                                Core
    # CCRTAICC_MSGD_PREF_CONTROL_DESC_POLL_EN   :Descriptor Polling
                                                Enable
    # CCRTAICC_MSGD_PREF_CONTROL_RUN            :Start Descriptor
                                                fetching operation

    __u64 NextDescriptorPointer
    __u32 DescriptorPollingFrequency
Return:  __ccrtaicc_lib_error_number_t
    # CCRTAICC_LIB_NO_ERROR                     (successful)
    # CCRTAICC_LIB_BAD_HANDLE                   (no/bad handler supplied)
    # CCRTAICC_LIB_NOT_OPEN                     (device not open)
    # CCRTAICC_LIB_INVALID_ARG                  (invalid argument)
    # CCRTAICC_LIB_MSG_DMA_NOT_SUPPORTED        (modular scatter-gather DMA
                                                not supported)
*****/

```

2.2.123 ccrtaICC_MsgDma_Release()

This *ccrtaICC_MsgDma_Release()* API call is used to free up the Modular Scatter-Gather DMA resource that has been reserved by the *ccrtaICC_MsgDma_Seize()* API.

```

/*****
    __ccrtaicc_lib_error_number_t  ccrtaICC_MsgDma_Release (void *Handle)

Description: Release MsgDMA operation for others to use

Input:      void                      *Handle (Handle pointer)
Output:     none
Return:     __ccrtaicc_lib_error_number_t
    # CCRTAICC_LIB_NO_ERROR             (successful)
    # CCRTAICC_LIB_BAD_HANDLE           (no/bad handler supplied)
    # CCRTAICC_LIB_MSGDMA_NOT_SUPPORTED (modular scatter-gather DMA
                                        not supported)
    # CCRTAICC_LIB_NOT_OWNER_OF_MSGDMA (not owner of modular
                                        scatter-gather)
*****/

```

2.2.124 ccrtaICC_MsgDma_Seize()

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

```

/*****
    __ccrtaicc_lib_error_number_t ccrtaICC_MsgDma_Seize (void *Handle)

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

Input:      void                      *Handle (Handle pointer)
Output:     none

```

```

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_MSGDMA_NOT_SUPPORTED    (modular scatter-gather DMA
                                                not supported)
        # CCRTAICC_LIB_MSGDMA_IN_USE          (modular scatter-gather DMA in
                                                use)
*****/

```

2.2.125 ccrtaICC_MsgDma_Setup()

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

```

/*****
    _ccrtaicc_lib_error_number_t
    ccrtaICC_MsgDma_Setup (void                *Handle,
                        _ccrtaicc_msgdma_descriptors_id_t StartDescriptorID,
                        int ForceReset,
                        ccrtaicc_msgdma_active_descriptors_info_t
                        *ActiveDescriptorsInfo)

Description: Setup MsgDMA Dispatcher and Prefetcher

Input:  void                *Handle (Handle pointer)
        _ccrtaicc_msgdma_descriptors_id_t *StartDescriptorID (Set
                                # CCRTAICC_MSGDMA_DESCRIPTOR_ID_1 ...
                                CCRTAICC_MSGDMA_DESCRIPTOR_ID_31
                                to valid ID)
        int ForceReset

Output: ccrtaicc_msgdma_active_descriptors_info_t *ActiveDescriptorsInfo;
        _ccrtaicc_msgdma_descriptors_id_t ID
        # CCRTAICC_MSGDMA_DESCRIPTOR_ID_1 ...
        CCRTAICC_MSGDMA_DESCRIPTOR_ID_31
        _ccrtaicc_msgdma_descriptors_id_mask_t Mask
        # CCRTAICC_MSGDMA_DESCRIPTOR_ID_1_MASK ...
        CCRTAICC_MSGDMA_DESCRIPTOR_ID_31_MASK
        # CCRTAICC_MSGDMA_DESCRIPTOR_ID_ALL_MASK
        __u32 NumberOfDescriptors
        __u32 TotalBytes

Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_INVALID_ARG            (invalid argument)
        # CCRTAICC_LIB_DMA_BUSY                (DMA Busy, cannot be
                                                reset)
        # CCRTAICC_LIB_ERROR_IN_DESCRIPTOR_LIST (invalid descriptor list)
        # CCRTAICC_LIB_MSG_DMA_NOT_SUPPORTED    (modular scatter-gather
                                                DMA not supported)
*****/

```

2.2.126 ccrtaICC_Munmap_Physical_Memory()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Munmap_Physical_Memory (void    *Handle,
                                void    *mmaped_user_mem_ptr)

Description: Unmap a previously mapped physical DMA memory.

Input:   void          *Handle          (Handle pointer)
Output:  void          *mmaped_user_mem_ptr (virtual memory pointer)
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR          (successful)
        # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN        (device not open)
        # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
        # CCRTAICC_LIB_MUNMAP_FAILED   (failed to un-map memory)
        # CCRTAICC_LIB_NOT_MAPPED     (memory not mapped)
*****/

```

2.2.127 ccrtAICC_NanoDelay()

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

```

/*****
void
ccrtAICC_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
*****/

```

2.2.128 ccrtAICC_Open()

This is the first call that needs to be issued by a user to open a device and access the board through the rest of the API calls. What is returned is a handle to a *void pointer* that is supplied as an argument to the other API calls. The *Board_Number* is a valid board number [0.9] that is associated with a physical card. There must exist a character special file */dev/ccrtaicc<Board_Number>* for the call to be successful. One character special file is created for each board found when the driver is successfully loaded.

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

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

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_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: _ccrtaicc_lib_error_number_t
 # CCRTAICC_LIB_NO_ERROR (successful)
 # CCRTAICC_LIB_INVALID_ARG (invalid argument)
 # CCRTAICC_LIB_ALREADY_OPEN (device already opened)
 # CCRTAICC_LIB_OPEN_FAILED (device open failed)
 # CCRTAICC_LIB_ALREADY_MAPPED (memory already mmapmed)
 # CCRTAICC_LIB_MMAP_SELECT_FAILED (mmap selection failed)
 # CCRTAICC_LIB_MMAP_FAILED (mmap failed)

*****/

2.2.129 ccrtaICC_Pause_UserProcess()

This call causes a running User Process to sleep for user specified micro-seconds.

```
/*  
  _ccrtaicc_lib_error_number_t  
  ccrtaICC_Pause_UserProcess(void *UFuncHandle,  
                              int usleep)
```

Description: Pause running user process

Input: void *UFuncHandle (UF Handle pointer)
 int usleep (micro-seconds sleep)

Output: none

Return: _ccrtaicc_lib_error_number_t
 # CCRTAICC_LIB_NO_ERROR (successful)
 # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)

*****/

2.2.130 ccrtaICC_Program_All_Output_Clocks()

This is the main call to program all the output clocks with a single call. All existing clock activity is stopped and replaced with the new clocks selection. Though the user can select the Input Clock Frequency with this call, it is expected that they will use the default CCRTAICC_DEFAULT_INPUT_CLOCK_FREQUENCY value.

The input clock can be one of:

CCRTAICC_CG_INPUT_CLOCK_SELECT_IN0 → 10MHz TCXO (Temperature Compensated Oscillator Clock).
CCRTAICC_CG_INPUT_CLOCK_SELECT_IN1 → External Input
CCRTAICC_CG_INPUT_CLOCK_SELECT_IN2 → FPGA Supplied
CCRTAICC_CG_INPUT_CLOCK_SELECT_INXAXB → Not used

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

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

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


```

/*****
ccrtAICC_Program_All_Output_Clocks()
_ccrtaicc_lib_error_number_t
ccrtAICC_Program_All_Output_Clocks(void          *Handle,
                                     double        InputClockFrequency,
                                     _ccrtaicc_cg_input_clock_select_register_t InputClockSel,
                                     ccrtaicc_compute_all_output_clocks_t *AllClocks,
                                     int          ProgramClocks,
                                     int          ActivateClocks)

```

Description: Program All Output Clocks

```

Input:  void          *Handle
        double        InputClockFrequency
        _ccrtaicc_cg_input_clock_select_register_t InputClockSel
        # CCRTAICC_CG_INPUT_CLOCK_SELECT_IN0
        # CCRTAICC_CG_INPUT_CLOCK_SELECT_IN1
        # CCRTAICC_CG_INPUT_CLOCK_SELECT_IN2
        # CCRTAICC_CG_INPUT_CLOCK_SELECT_INXAXB
        ccrtaicc_compute_all_output_clocks_t *AllClocks (pointer to
        ccrtaicc_compute_single_output_clock_t *Clock (Pointer to
        long double        DesiredFrequency (returned output clock info)
        double        DesiredTolerancePPT
        int          ProgramClocks (program
        int          ActivateClocks (1=activate
        Output:  ccrtaicc_compute_all_output_clocks_t *AllClocks (Pointer to
        ccrtaicc_compute_single_output_clock_t *Clock (Pointer to
        _ccrtaicc_clock_generator_output_t OutputClock
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_0
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_1
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_2
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_3
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_4
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_5
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_6
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_7
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_8
        # CCRTAICC_CLOCK_GENERATOR_OUTPUT_9
        double        InputClockFrequency
        long double   FrequencyDeviation
        int          FrequencyFound
        long double   ActualFrequency
        double        ActualTolerancePPT
        __u64        Mdiv_Numerator
        __u32        Mdiv_Denominator
        __u64        Ndiv_Numerator
        __u32        Ndiv_Denominator
        _ccrtaicc_cg_outmux_ndiv_select_t Ndiv_ToUse
        # CCRTAICC_CG_OUTPUT_MUX_NDIV_0
        # CCRTAICC_CG_OUTPUT_MUX_NDIV_1
        # CCRTAICC_CG_OUTPUT_MUX_NDIV_2
        # CCRTAICC_CG_OUTPUT_MUX_NDIV_3
        # CCRTAICC_CG_OUTPUT_MUX_NDIV_4

```

```

        __u32
        __u32
        __u32
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_NO_LOCAL_REGION    (local region error)
        # CCRTAICC_LIB_IO_ERROR          (device not ready)
        # CCRTAICC_LIB_N_DIVIDERS_EXCEEDED (number of N-Dividers
        exceeded)
        # CCRTAICC_LIB_CANNOT_COMPUTE_OUTPUT_FREQ (cannot compute
        output freq)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_CLOCK_GENERATION_FAILED (clock generation
        failed)
*****/

```

2.2.131 ccrtaICC_Read()

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

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

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

```

/*****
    _ccrtaicc_lib_error_number_t
    ccrtaICC_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:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_BAD_HANDLE        (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN          (device not open)
        # CCRTAICC_LIB_IO_ERROR          (read failed)
*****/

```

2.2.132 ccrtaICC_Reload_Firmware()

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

```

/*****
    ccrtaICC_Reload_Firmware()

```

Description: This call power-cycles the board which in turn forces it to reload its firmware. Typically, this is called after a new firmware has been installed in the board. This saves the need to perform a system reboot after a firmware installation.

Input: void *Handle (Handle pointer)
Output: none
Return: _ccrtaicc_lib_error_number_t
CCRTAICC_LIB_NO_ERROR (successful)
CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
CCRTAICC_LIB_NOT_OPEN (device not open)
CCRTAICC_LIB_IOCTL_FAILED (driver ioctl call failed)
*****/

2.2.133 ccrtaICC_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.

```
/*  
_ccrtaicc_lib_error_number_t  
ccrtaICC_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: _ccrtaicc_lib_error_number_t
CCRTAICC_LIB_NO_ERROR (successful)
CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
CCRTAICC_LIB_NOT_OPEN (device not open)
CCRTAICC_LIB_IOCTL_FAILED (driver ioctl call failed)
*****/

2.2.134 ccrtaICC_Reset_Board()

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

```
/*  
_ccrtaicc_lib_error_number_t  
ccrtaICC_Reset_Board (void *Handle)
```

Description: Reset the board.

Input: void *Handle (Handle pointer)
Output: none
Return: _ccrtaicc_lib_error_number_t
CCRTAICC_LIB_NO_ERROR (successful)
CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
CCRTAICC_LIB_NOT_OPEN (device not open)
CCRTAICC_LIB_IOCTL_FAILED (driver ioctl call failed)
CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
*****/

2.2.135 ccrtAICC_Reset_Clock()

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

```
/*
  _ccrtaicc_lib_error_number_t
  ccrtAICC_Reset_Clock (void *Handle,
                       int activate)

  Description: Perform Hardware Clock Reset

  Input:   void *Handle (Handle pointer)
           int activate (1=activate after reset)

  Output:  none

  Return:  _ccrtaicc_lib_error_number_t
           # CCRTAICC_LIB_NO_ERROR (successful)
           # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
           # CCRTAICC_LIB_NOT_OPEN (device not open)
           # CCRTAICC_LIB_IOCTL_FAILED (driver ioctl call failed)
           # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
  */
```

2.2.136 ccrtAICC_Resume_UserProcess()

Use this call to resume an already paused User Process.

```
/*
  _ccrtaicc_lib_error_number_t
  ccrtAICC_Resume_UserProcess (void *UFuncHandle)

  Description: Resume paused running user process

  Input:   void *UFuncHandle (UF Handle pointer)

  Output:  none

  Return:  _ccrtaicc_lib_error_number_t
           # CCRTAICC_LIB_NO_ERROR (successful)
           # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
  */
```

2.2.137 ccrtAICC_Return_Board_Info_Description()

Return board information description

```
/*
  char *
  ccrtAICC_Return_Board_Info_Description (_ccrtaicc_board_function_t
                                         BoardFunction)

  Description: Return Board Information Description

  Input:   _ccrtaicc_board_function_t BoardFunction (board function)
           # CCRTAICC_BOARD_FUNCTION_AICC
           # CCRTAICC_BOARD_FUNCTION_BASE_LEVEL
           # CCRTAICC_BOARD_FUNCTION_UNDEFINED

  Output:  none

  Return:  char *BoardFuncDesc (board function
                                description)
  */
```

2.2.138 ccrtAICC_SDRAM_Activate() **

Currently, SDRAM is not supported by this hardware.

This call must be the first call to activate the SDRAM. Without activation, all other calls will fail. The user can also use this call to return the current state of the SDRAM without any change.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_SDRAM_Activate (void                *Handle,
                        _ccrtaicc_sdr
                        _ccrtaicc_sdr
                        _ccrtaicc_sdr
                        *current_state)

Description: Activate/DeActivate SDRAM module

Input:   void                *Handle          (Handle pointer)
         _ccrtaicc_sdr
         # CCRTAICC_SDRAM_ALL_DISABLE
         # CCRTAICC_SDRAM_ALL_ENABLE
         # CCRTAICC_SDRAM_ALL_ENABLE_DO_NOT_CHANGE
Output:  _ccrtaicc_sdr
         # CCRTAICC_SDRAM_ALL_DISABLE
         # CCRTAICC_SDRAM_ALL_ENABLE
Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR              (successful)
         # CCRTAICC_LIB_BAD_HANDLE           (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN            (device not open)
         # CCRTAICC_LIB_INVALID_ARG         (invalid argument)
         # CCRTAICC_LIB_NO_LOCAL_REGION     (local region not present)
         # CCRTAICC_LIB_SDRAM_INITIALIZATION_FAILED (SDRAM init failed)
         # CCRTAICC_LIB_SDRAM_NOT_SUPPORTED (SDRAM not supported)
*****/

```

2.2.139 ccrtaICC_SDRAM_Get_CSR() **

Currently, SDRAM is not supported by this hardware.

This call returns the SDRAM control and status register information.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_SDRAM_Get_CSR (void                *Handle,
                       ccrtaicc_sdr
                       *sdr
                       *sdr)

Description: Get SDRAM Control and Status information

Input:   void                *Handle          (Handle pointer)
Output:  ccrtaicc_sdr
         *sdr                 (pointer to SDRAM csr)
         _ccrtaicc_sdr
         # CCRTAICC_SDRAM_READ_AUTO_INCREMENT_DISABLE
         # CCRTAICC_SDRAM_READ_AUTO_INCREMENT_ENABLE
         _ccrtaicc_sdr
         # CCRTAICC_SDRAM_WRITE_AUTO_INCREMENT_DISABLE
         # CCRTAICC_SDRAM_WRITE_AUTO_INCREMENT_ENABLE
         _ccrtaicc_sdr
         # CCRTAICC_SDRAM_READ_TIMEOUT_DID_NOT_OCCUR
         # CCRTAICC_SDRAM_READ_TIMEOUT_OCCURRED
         _ccrtaicc_sdr
         # CCRTAICC_SDRAM_CALIBRATION_FAIL_RESET
         # CCRTAICC_SDRAM_CALIBRATION_FAIL_SET
         _ccrtaicc_sdr
         # CCRTAICC_SDRAM_CALIBRATION_PASS_RESET
         # CCRTAICC_SDRAM_CALIBRATION_PASS_SET
         _ccrtaicc_sdr
         # CCRTAICC_SDRAM_INITIALIZATION_NOT_COMPLETE
         # CCRTAICC_SDRAM_INITIALIZATION_COMPLETE
*****/

```

```

Return:  _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR                (successful)
         # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
         # CCRTAICC_LIB_NOT_OPEN                (device not open)
         # CCRTAICC_LIB_INVALID_ARG             (invalid argument)
         # CCRTAICC_LIB_NO_LOCAL_REGION         (local region not present)
         # CCRTAICC_LIB_SDRAM_IS_NOT_ACTIVE     (SDRAM is not active)
*****/

```

2.2.140 ccrtaICC_SDRAM_Read() **

Currently, SDRAM is not supported by this hardware.

This call provided the user the ability to read any portion of the SDRAM. Its range is from 1 to 0x10000000 (256Mwords). Offset to this routine is only set if it is 0 or greater. Maximum offset is 0x0FFFFFFF. If offset is negative, then the read commences from the last read location.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_SDRAM_Read (void      *Handle,
                    u_int32_t *Buf,
                    int       Offset,
                    u_int32_t Size,
                    u_int32_t *Words_read)

Description: Perform a SDRAM read operation.

Input:  void      *Handle      (Handle pointer)
        int       Offset      (word offset into SDRAM)
        u_int32_t Size        (size of buffer in words)
Output: u_int32_t *Buf        (pointer to buffer)
        u_int32_t *Words_read (words read)

Return: _ccrtaicc_lib_error_number_t
         # CCRTAICC_LIB_NO_ERROR                (successful)
         # CCRTAICC_LIB_NO_LOCAL_REGION         (local region not present)
         # CCRTAICC_LIB_SDRAM_IS_NOT_ACTIVE     (SDRAM is not active)
*****/

```

2.2.141 ccrtaICC_SDRAM_Set_CSR() **

Currently, SDRAM is not supported by this hardware.

This call sets the SDRAM control and status register.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_SDRAM_Set_CSR (void      *Handle,
                      ccrtaicc_sdram_csr_t *sdram_csr)

Description: Set SDRAM Control and Status information

Input:  void      *Handle      (Handle pointer)
        _ccrtaicc_sdram_read_auto_increment_t  read_auto_increment;
        # CCRTAICC_SDRAM_READ_AUTO_INCREMENT_DISABLE
        # CCRTAICC_SDRAM_READ_AUTO_INCREMENT_ENABLE
        _ccrtaicc_sdram_write_auto_increment_t  write_auto_increment;
        # CCRTAICC_SDRAM_WRITE_AUTO_INCREMENT_DISABLE
        # CCRTAICC_SDRAM_WRITE_AUTO_INCREMENT_ENABLE
        _ccrtaicc_sdram_read_timeout_t         read_timeout;
        # CCRTAICC_SDRAM_READ_TIMEOUT_DID_NOT_OCCUR
        # CCRTAICC_SDRAM_READ_TIMEOUT_OCCURRED
        _ccrtaicc_sdram_calibration_fail_t     calibration_failed;

```

```

        # CCRTAICC_SDRAM_CALIBRATION_FAIL_RESET
        # CCRTAICC_SDRAM_CALIBRATION_FAIL_SET
        _ccrtaicc_sdrām_calibration_pass_t      calibration_passed;
        # CCRTAICC_SDRAM_CALIBRATION_PASS_RESET
        # CCRTAICC_SDRAM_CALIBRATION_PASS_SET
        _ccrtaicc_sdrām_initilization_done_t    initialization_done;
        # CCRTAICC_SDRAM_INITIALIZATION_NOT_COMPLETE
        # CCRTAICC_SDRAM_INITIALIZATION_COMPLETE

Output:  none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_INVALID_ARG            (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION        (local region not present)
        # CCRTAICC_LIB_SDRAM_IS_NOT_ACTIVE    (SDRAM is not active)
*****

```

2.2.142 ccrtaICC_SDRAM_Write() **

Currently, SDRAM is not supported by this hardware.

This call provided the user the ability to write to any portion of the SDRAM. Its range is from 1 to 0x10000000 (256Mwords). Offset to this routine is only set if it is 0 or greater. Maximum offset is 0x0FFFFFFF. If offset is negative, then the write commences from the last written location.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_SDRAM_Write (void          *Handle,
                      u_int32_t     *Buf,
                      int            Offset,
                      u_int32_t     Size,
                      u_int32_t     *Words_written)

Description: Perform a SDRAM write operation.

Input:   void          *Handle      (Handle pointer)
         u_int32_t     *Buf         (pointer to buffer)
         int           Offset       (word offset into SDRAM)
         u_int32_t     Size         (size of buffer in words)
Output:  u_int32_t     *Words_written (words written)
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_NO_LOCAL_REGION        (local region not present)
        # CCRTAICC_LIB_SDRAM_IS_NOT_ACTIVE    (SDRAM is not active)
*****

```

2.2.143 ccrtaICC_Set_Board_CSR()

This call is used to set the board control register.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_Set_Board_CSR (void          *Handle,
                       ccrtaicc_board_csr_t *bcsr)

Description: Set Board Control and Status information

Input:   void          *Handle      (Handle pointer)
         ccrtaicc_board_csr_t *bcsr  (pointer to board csr)
         _ccrtaicc_bcsr_identify_board_t identify_board
         # CCRTAICC_BCSR_IDENTIFY_BOARD_DISABLE
         # CCRTAICC_BCSR_IDENTIFY_BOARD_ENABLE

```

```

                                # CCRTAICC_BCSR_IDENTIFY_BOARD_DO_NOT_CHANGE
Output: none
Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_INVALID_ARG             (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION         (local region not present)
*****

```

2.2.144 ccrtaICC_Set_Calibration_CSR()

This call sets the current calibration control and status register.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_Set_Calibration_CSR (void                                *Handle,
                             ccrtaicc_calibration_csr_t *CalCSR)

Description: Set Calibration Control and Status Register

Input:  void                                *Handle      (Handle pointer)
        ccrtaicc_calibration_csr_t *CalCSR      (pointer to calibration CSR)
        _ccrtaicc_calbus_control_t BusControl  (bus control)
        # CCRTAICC_CB_GROUND
        # CCRTAICC_CB_POSITIVE_10V_REFERENCE
        # CCRTAICC_CB_NEGATIVE_10V_REFERENCE
        # CCRTAICC_CB_POSITIVE_5V_REFERENCE
        # CCRTAICC_CB_NEGATIVE_5V_REFERENCE
        # CCRTAICC_CB_POSITIVE_2V_REFERENCE
        # CCRTAICC_CB_BUS_OPEN

Return:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_INVALID_ARG             (invalid argument)
        # CCRTAICC_LIB_NO_LOCAL_REGION         (local region not present)
*****

```

2.2.145 ccrtaICC_Set_External_Clock_CSR()

This call sets the External Clock Input.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_Set_External_Clock_CSR (void                                *Handle,
                                 ccrtaicc_external_clock_csr_t *ExtClkCSR)

Description: Set External Clock Control & Status

Input:  void                                *Handle      (handle pointer)
        ccrtaicc_external_clock_csr_t *ExtClkCSR  (pointer to external clock
                                                    CSR)
        _ccrtaicc_external_clock_output_select_t ClockOutputSelect;
        # CCRTAICC_EXCLOS_CLOCK_GENERATOR_OUTPUT_0
        # CCRTAICC_EXCLOS_CLOCK_GENERATOR_OUTPUT_1
        # CCRTAICC_EXCLOS_CLOCK_GENERATOR_OUTPUT_2
        # CCRTAICC_EXCLOS_CLOCK_GENERATOR_OUTPUT_3
        # CCRTAICC_EXCLOS_EXTERNAL_CLOCK_INPUT
        # CCRTAICC_EXCLOS_NO_CLOCK
        # CCRTAICC_EXCLOS_DO_NOT_CHANGE

Output: none
Return:  _ccrtaicc_lib_error_number_t

```



```

# CCRTAICC_LIB_NO_ERROR                (successful)
# CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN                (device not open)
# CCRTAICC_LIB_NO_LOCAL_REGION        (local region error)
# CCRTAICC_LIB_INVALID_ARG            (invalid argument)
*****/

```

2.2.146 ccrtAICC_Set_Interrupt_Status()

This call sets/clears the various interrupts.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Set_Interrupt_Status (void          *Handle,
                              ccrtaicc_interrupt_t *intr)

Description: Set Interrupt Status

Input:  void          *Handle          (handle pointer)
        ccrtaicc_interrupt_t *intr    (pointer to interrupt status)
        _ccrtaicc_intsta_adc_t
        # CCRTAICC_INT_ADC_FIFO_THRESHOLD_NONE
        # CCRTAICC_INT_ADC_FIFO_THRESHOLD_RESET
        # CCRTAICC_INT_ADC_FIFO_THRESHOLD_DO_NOT_CHANGE

Output: none
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_NO_LOCAL_REGION        (local region error)
        # CCRTAICC_LIB_INVALID_ARG            (invalid argument)

*****/

```

2.2.147 ccrtAICC_Set_Interrupt_Timeout_Seconds()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_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: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR                (successful)
        # CCRTAICC_LIB_BAD_HANDLE              (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN                (device not open)
        # CCRTAICC_LIB_INVALID_ARG            (invalid argument)

*****/

```

2.2.148 ccrtAICC_Set_TestBus_Control()

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

```

/*****

```

```

_ccrtaicc_lib_error_number_t
ccrtaicc_Set_TestBus_Control (void *Handle,
                             _ccrtaicc_testbus_control_t test_control)

Description: Set the value of the Test Bus control information

Input:  void *Handle (handle pointer)
Output: _ccrtaicc_testbus_control_t
        test_control (control select)
        # CCRTAICC_TBUS_CONTROL_OPEN
        # CCRTAICC_TBUS_CONTROL_CAL_BUS
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR (successful)
        # CCRTAICC_LIB_NO_LOCAL_REGION (local region error)
        # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN (device not open)
*****/

```

2.2.149 ccrtaicc_Set_Value()

This call allows the advanced user to set the writable board registers. The actual data written will depend on the command register information that is requested. Refer to the hardware manual for more information on what can be written to.

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaicc_Set_Value (void *Handle,
                   CCRTAICC_CONTROL cmd,
                   void *value)

Description: Set the value of the specified board register.

Input:  void *Handle (Handle pointer)
        CCRTAICC_CONTROL cmd (register definition)
        -- structure in ccrtaicc_lib.h
        void *value (pointer to value to be set)
Output: none
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR (successful)
        # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN (device not open)
        # CCRTAICC_LIB_INVALID_ARG (invalid argument)
*****/

```

2.2.150 ccrtaicc_SPROM_Read() **

Currently, SPROM is not supported by this hardware.

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.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaicc_SPROM_Read(void *Handle,
                   ccrtaicc_sprom_rw_t *spr)

Description: Read Serial Prom for specified number of words

Input:  void *Handle (handle pointer)

```

```

        ccrtAICC_sprom_rw_t   *spr           (pointer to struct)
        u_short word_offset
        u_short num_words
Output:  ccrtAICC_sprom_rw_t   *spr           (pointer to struct)
        u_short *data_ptr
Return:  _ccrtAICC_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR           (successful)
        # CCRTAICC_LIB_NO_LOCAL_REGION    (error)
        # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
        # CCRTAICC_LIB_SERIAL_PROM_BUSY  (serial prom busy)
        # CCRTAICC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
        # CCRTAICC_LIB_SERIAL_PROM_NOT_PRESENT (serial prom not present)
*****/

```

2.2.151 ccrtAICC_SPROM_Read_Item() **

Currently, SPROM is not supported by this hardware.

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.

```

/*****
    _ccrtAICC_lib_error_number_t
    ccrtAICC_SPROM_Read_Item(void          *Handle,
                                   _ccrtAICC_sprom_access_t item,
                                   void      *item_ptr)

Description: Read Serial Prom for specified item

Input:      void          *Handle      (handle pointer)
            _ccrtAICC_sprom_access_t item (select item)
            # CCRTAICC_SPROM_HEADER

Output:     void          *item_ptr    (pointer to item struct)
            *ccrtAICC_sprom_header_t sprom_header
            u_int32_t      board_serial_number
            u_short        sprom_revision

Return:     _ccrtAICC_lib_error_number_t
            # CCRTAICC_LIB_NO_ERROR           (successful)
            # CCRTAICC_LIB_NO_LOCAL_REGION    (error)
            # CCRTAICC_LIB_INVALID_ARG       (invalid argument)
            # CCRTAICC_LIB_SERIAL_PROM_BUSY  (serial prom busy)
            # CCRTAICC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
            # CCRTAICC_LIB_SERIAL_PROM_NOT_PRESENT (serial prom not present)
*****/

```

2.2.152 ccrtAICC_SPROM_Write() **

Currently, SPROM is not supported by this hardware.

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 *ccrtAICC_SPROM_Write_Override()* to allow writing to the serial prom.

```

/*****
    _ccrtAICC_lib_error_number_t
    ccrtAICC_SPROM_Write(void          *Handle,
                           ccrtAICC_sprom_rw_t *spw)

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

```

```

Input:      void          *Handle      (handle pointer)
           ccrtaicc_sprom_rw_t *spw      (pointer to struct)
           u_short word_offset
           u_short num_words
           u_short *data_ptr

Output:     none

Return:     _ccrtaicc_lib_error_number_t
           # CCRTAICC_LIB_NO_ERROR      (successful)
           # CCRTAICC_LIB_NO_LOCAL_REGION (error)
           # CCRTAICC_LIB_INVALID_ARG   (invalid argument)
           # CCRTAICC_LIB_SERIAL_PROM_BUSY (serial prom busy)
           # CCRTAICC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
           # CCRTAICC_LIB_SERIAL_PROM_NOT_PRESENT (serial prom not present)
*****/

```

2.2.153 ccrtaICC_SPROM_Write_Item() **

Currently, SPROM is not supported by this hardware.

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. This call should normally not be used by the user.

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_SPROM_Write_Item(void          *Handle,
                           _ccrtaicc_sprom_access_t item,
                           void          *item_ptr)

```

Description: Write Serial Prom with specified item

```

Input:      void          *Handle      (handle pointer)
           _ccrtaicc_sprom_access_t item      (select item)
           # CCRTAICC_SPROM_HEADER
           void          *item_ptr      (pointer to item struct)
           *ccrtaicc_sprom_header_t sprom_header
           u_int32_t      board_serial_number
           u_short      sprom_revision

```

```

Output:     none

Return:     _ccrtaicc_lib_error_number_t
           # CCRTAICC_LIB_NO_ERROR      (successful)
           # CCRTAICC_LIB_NO_LOCAL_REGION (error)
           # CCRTAICC_LIB_INVALID_ARG   (invalid argument)
           # CCRTAICC_LIB_SERIAL_PROM_BUSY (serial prom busy)
           # CCRTAICC_LIB_SERIAL_PROM_FAILURE (serial prom failure)
           # CCRTAICC_LIB_SERIAL_PROM_NOT_PRESENT (serial prom not present)
*****/

```

2.2.154 ccrtaICC_SPROM_Write_Override() **

Currently, SPROM is not supported by this hardware.

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:

- ccrtAICC_Write_Serial_Prom()
- ccrtAICC_Write_Serial_Prom_Item()

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

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_SPROM_Write_Override (void *Handle,
                                int action)

  Description: Set Serial Prom Write Override

  Input:      void          *Handle          (handle pointer)
              int           action          (override action)
              # CCRTAICC_TRUE
              # CCRTAICC_FALSE

  Output:     none

  Return:     _ccrtaicc_lib_error_number_t
              # CCRTAICC_LIB_NO_ERROR          (successful)
              # CCRTAICC_LIB_BAD_HANDLE       (no/bad handler supplied)
              # CCRTAICC_LIB_NOT_OPEN        (device not open)
              # CCRTAICC_LIB_INVALID_ARG     (invalid argument)
              # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
              # CCRTAICC_LIB_SERIAL_PROM_NOT_PRESENT (serial prom not present)
*****/

```

2.2.155 ccrtAICC_Transfer_Data()

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

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

When scatter-gather DMA is selected, the *DmaEngineNo* argument is ignored and the *IoControl* argument must be set to *CCRTAICC_DMA_CONTROL_INCREMENT*.

There are certain limitations to modular scatter-gather feature:

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

```

/*****
  _ccrtaicc_lib_error_number_t
  ccrtAICC_Transfer_Data (void          *Handle,
                          volatile void *PciDmaMemory,
                          volatile void *AvalonMem,
                          uint          TransferSize,
                          _ccrtaicc_direction_t XferDirection,
*****/

```

```

        _ccrtaicc_library_rw_mode_t      LibMode,
        ccrtaicc_dma_engine_t           DMAEngineNo,
        ccrtaicc_bool                    UseInterrupts,
        int                               IoControl)

```

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

```

Input:  void                *Handle          (Handle pointer)
        volatile void      *PciDmaMemory    (pointer to virtual memory)
        volatile void      *AvalonMem       (pointer to virtual Avalon
        memory)
        uint                TransferSize     (size of transfer in bytes)
        _ccrtaicc_direction_t XferDirection (direction of transfer)
        # CCRTAICC_AVALON_2_PCIMEM
        # CCRTAICC_PCIMEM_2_AVALON
        _ccrtaicc_library_rw_mode_t LibMode (Lib transfer mode)
        # CCRTAICC_LIBRARY_PIO_MODE
        # CCRTAICC_LIBRARY_DMA_MODE
        # CCRTAICC_LIBRARY_MSGDMA_MODE
        ccrtaicc_dma_engine_t DMAEngineNo    (select DMA engine)
        # CCRTAICC_DMA0
        # CCRTAICC_DMA1
        # CCRTAICC_NONE
        ccrtaicc_bool        UseInterrupts    (enable interrupts)
        # CCRTAICC_TRUE
        # CCRTAICC_FALSE
        int                  IoControl        (DMA or PIO control flags)
        # CCRTAICC_DMA_CONTROL_RCON          (DMA: read constant)
        # CCRTAICC_DMA_CONTROL_WCON          (DMA: write constant)
        # CCRTAICC_DMA_CONTROL_INCREMENT     (DMA: increment)
        # CCRTAICC_PIO_CONTROL_RCON          (PIO: read constant)
        # CCRTAICC_PIO_CONTROL_WCON          (PIO: write constant)
        # CCRTAICC_PIO_CONTROL_INCREMENT     (PIO: increment)

```

Output: none

```

Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR              (no error)
        # CCRTAICC_LIB_BAD_HANDLE            (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN              (library not open)
        # CCRTAICC_LIB_INVALID_ARG          (invalid argument)
        # CCRTAICC_LIB_IOCTL_FAILED         (driver ioctl call failed)
        # CCRTAICC_LIB_MSG_DMA_READS_NOT_ALLOWED_FOR_SELECTED_ADDRESS
        (MSG DMA Reads not allowed
        for selected address)

```

*****/

2.2.156 ccrtaICC_Update_Clock_Generator_Divider()

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

```

/*****
        _ccrtaicc_lib_error_number_t
        ccrtaICC_Update_Clock_Generator_Divider (void                *Handle,
        _ccrtaicc_clock_generator_divider_t    WhichDivider)

```

Description: Update Clock Generator Divider

```

Input:  void                *Handle          (Handle pointer)
        _ccrtaicc_clock_generator_divider_t WhichDivider    (select divider)
        # CCRTAICC_CLOCK_GENERATOR_DIVIDER_M

```

```

# CCRTAICC_CLOCK_GENERATOR_DIVIDER_N0
# CCRTAICC_CLOCK_GENERATOR_DIVIDER_N1
# CCRTAICC_CLOCK_GENERATOR_DIVIDER_N2
# CCRTAICC_CLOCK_GENERATOR_DIVIDER_N3
# CCRTAICC_CLOCK_GENERATOR_DIVIDER_N_ALL
# CCRTAICC_CLOCK_GENERATOR_DIVIDER_P0
# CCRTAICC_CLOCK_GENERATOR_DIVIDER_P1
# CCRTAICC_CLOCK_GENERATOR_DIVIDER_P2
# CCRTAICC_CLOCK_GENERATOR_DIVIDER_PFB
# CCRTAICC_CLOCK_GENERATOR_DIVIDER_P_ALL
# CCRTAICC_CLOCK_GENERATOR_DIVIDER_PXAXB
Output: none
Return: _ccrtaicc_lib_error_number_t
# CCRTAICC_LIB_NO_ERROR (successful)
# CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
# CCRTAICC_LIB_NOT_OPEN (library not open)
# CCRTAICC_LIB_NO_LOCAL_REGION (local region error)
# CCRTAICC_LIB_INVALID_ARG (invalid argument)
# CCRTAICC_LIB_CLOCK_IS_NOT_ACTIVE (Clock is not active)
*****/

```

2.2.157 ccrtaICC_UserProcess_Command()

The user can control the execution of the created User Process with the help of this call.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtaICC_UserProcess_Command(void *Handle,
                             void *UFuncHandle,
                             ccrtaicc_uf_action_t Action)

Description: Command User process

Input: void *UFuncHandle (User Process Handle pointer)
       ccrtaicc_uf_action_t Action (command action)
       # CCRTAICC_UF_ACTION_STOP
       # CCRTAICC_UF_ACTION_RUN
       # CCRTAICC_UF_ACTION_TERMINATE

Output: none
Return: none
*****/

```

2.2.158 ccrtaICC_VoltsToData()

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

```

/*****
uint
ccrtaICC_VoltsToData (double volts,
                    ccrtaicc_volt_convert_t *conv)

Description: Convert Volts to data

Input: double volts (volts to convert)
       ccrtaicc_volt_convert_t *conv (pointer to conversion struct)
       double VoltageRange (maximum voltage range)
       _ccrtaicc_csr_dataformat_t Format (format)
       # CCRTAICC_OFFSET_BINARY
       # CCRTAICC_TWOS_COMPLEMENT
       ccrtaicc_bool BiPolar (bi-polar)
       # CCRTAICC_TRUE

```

```

        # CCRTAICC_FALSE
        int ResolutionBits (Number of resolution bits)
Output: none
Return: uint data (returned data)
*****/

```

2.2.159 ccrtAICC_Wait_For_Interrupt()

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

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_Wait_For_Interrupt (void *Handle,
                             ccrtaicc_driver_int_t *drv_int)

Description: Wait For Interrupt

Input: void *Handle (Handle pointer)
        ccrtaicc_driver_int_t *drv_int (pointer to drv_int struct)
        uint WakeupInterruptMask
        # CCRTAICC_DMA0_INTMASK
        # CCRTAICC_DMA1_INTMASK
        # CCRTAICC_MSGDMA_INTMASK
        # CCRTAICC_ADC_FIFO_INTMASK
        int timeout_seconds
Output: ccrtaicc_driver_int_t *drv_int (pointer to drv_int struct)
        long long unsigned count
        long long unsigned dma_count[CCRTAICC_DMA_MAX_ENGINES]
        long long unsigned MsgDma_count
        uint InterruptsOccurredMask
        uint WakeupInterruptMask
        int DmaControl (DMA control flags)
        # CCRTAICC_DMA_CONTROL_RCON (read constant)
        # CCRTAICC_DMA_CONTROL_WCON (write constant)
        # CCRTAICC_DMA_CONTROL_INCREMENT (increment)
Return: _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR (successful)
        # CCRTAICC_LIB_BAD_HANDLE (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN (device not open)
        # CCRTAICC_LIB_NO_LOCAL_REGION (local region not present)
        # CCRTAICC_LIB_INVALID_ARG (invalid argument)
*****/

```

2.2.160 ccrtAICC_Write()

This call is currently not supported by the driver.

```

/*****
_ccrtaicc_lib_error_number_t
ccrtAICC_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:  _ccrtaicc_lib_error_number_t
        # CCRTAICC_LIB_NO_ERROR      (successful)
        # CCRTAICC_LIB_BAD_HANDLE    (no/bad handler supplied)
        # CCRTAICC_LIB_NOT_OPEN      (device not open)
        # CCRTAICC_LIB_IO_ERROR      (write failed)
        # CCRTAICC_LIB_NOT_IMPLEMENTED (call not implemented)
*****/
```

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 ccrtaiicc_disp

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

```
Usage: ./ccrtaiicc_disp [-b BoardNo] [-d Delay] [-l LoopCnt] [-o Offset] [-s Size]
-b BoardNo (Board number -- default is 0)
-d Delay (Delay between screen refresh -- default is 0)
-l LoopCnt (Loop count -- default is 0)
-o Offset (Hex offset to read from -- default is 0x0)
-s Size (Number of bytes to read -- default is 0x400)
```

Example display:

```
./ccrtaiicc_disp
```

```
Board Number [-b]: 0
Delay [-d]: 0 milli-seconds
Loop Count [-l]: ***Forever***
Offset [-o]: 0x00000000
Size [-s]: 1024 (bytes)
```

```
ScanCount = 33783
```

```
          00          04          08          0C          10          14          18          1C
          =====
000000  93500101 12052018 00010000 00000000 00000000 00000000 00000000
000020  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000040  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000060  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000080  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000a0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000c0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000e0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000100  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000120  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000140  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000160  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000180  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001a0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001c0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001e0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000200  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000220  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000240  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000260  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000280  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0002a0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0002c0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0002e0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000300  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000320  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000340  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000360  00000000 00000000 00000000 00000000 00000000 00000000 00000000
000380  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0003a0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0003c0  00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
0003e0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

3.1.2 ccrtaiicc_dump

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

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

Example display:

```
./ccrtaiicc_dump
```

```
Device Name: /dev/ccrtaiicc0
```

```
LOCAL REGION: Physical Addr=0xbd500000 Size=131072 (0x00020000)  
CONFIG REGION: Physical Addr=0xbd520000 Size=32768 (0x00008000)
```

```
LOCAL: Register 0x7ffff7fd7000 Offset=0x0 Size=0x00020000  
CONFIG: Register 0x7ffff7fcf000 Offset=0x0 Size=0x00008000  
LIBPTR: Register 0x7ffff7fcd000 Offset=0x0 Size=0x00001008
```

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

```
LBR: @0x000000 --> 0x93500101  
LBR: @0x000004 --> 0x12052018  
LBR: @0x000008 --> 0x00010000  
LBR: @0x00000c --> 0x00000000  
LBR: @0x000010 --> 0x00000000  
LBR: @0x000014 --> 0x00000000  
LBR: @0x000018 --> 0x00000000  
LBR: @0x00001c --> 0x00000000  
LBR: @0x000020 --> 0x00000000  
LBR: @0x000024 --> 0x00000000  
LBR: @0x000028 --> 0x00000000  
LBR: @0x00002c --> 0x00000000  
LBR: @0x000030 --> 0x00000000
```

```
.  
. .
```

```
LBR: @0x01ffcc --> 0x00000000  
LBR: @0x01ffd0 --> 0x00000000  
LBR: @0x01ffd4 --> 0x00000000  
LBR: @0x01ffd8 --> 0x00000000  
LBR: @0x01ffdc --> 0x00000000  
LBR: @0x01ffe0 --> 0x00000000  
LBR: @0x01ffe4 --> 0x00000000  
LBR: @0x01ffe8 --> 0x00000000  
LBR: @0x01ffec --> 0x00000000  
LBR: @0x01fff0 --> 0x00000000  
LBR: @0x01fff4 --> 0x00000000  
LBR: @0x01fff8 --> 0x00000000  
LBR: @0x01fffc --> 0x00000000
```

```
===== LOCAL CONFIG REGISTERS =====
```

```
#### CONFIG REGS (PCIeLinkPartnerRegs) ####  
LCR: @0x0000 --> 0x00000000  
LCR: @0x0004 --> 0x00000000  
LCR: @0x0008 --> 0x00000000
```

```

LCR: @0x000c --> 0x00000000
LCR: @0x0010 --> 0x00000000
LCR: @0x0014 --> 0x00000000
LCR: @0x0018 --> 0x00000000
LCR: @0x001c --> 0x00000000
LCR: @0x0020 --> 0x00000000
LCR: @0x0024 --> 0x00000000
LCR: @0x0028 --> 0x00000000
LCR: @0x002c --> 0x00000000
LCR: @0x0030 --> 0x00000000
.
.
.
LCR: @0x0fc0 --> 0x00000000
LCR: @0x0fc4 --> 0x00000000
LCR: @0x0fc8 --> 0x00000000
LCR: @0x0fcc --> 0x00000000
LCR: @0x0fd0 --> 0x00000000
LCR: @0x0fd4 --> 0x00000000
LCR: @0x0fd8 --> 0x00000000
LCR: @0x0fdc --> 0x00000000
LCR: @0x0fe0 --> 0x00000000
LCR: @0x0fe4 --> 0x00000000
LCR: @0x0fe8 --> 0x00000000
LCR: @0x0fec --> 0x00000000
LCR: @0x0ff0 --> 0x00000000
LCR: @0x0ff4 --> 0x00000000
LCR: @0x0ff8 --> 0x00000000
LCR: @0x0ffc --> 0x00000000

#### CONFIG REGS (AvalonMM_2_PCIeAddrTrans) ####
LCR: @0x1000 --> 0x00000000
LCR: @0x1004 --> 0x00000000
LCR: @0x1008 --> 0x00000000
LCR: @0x100c --> 0x00000000
LCR: @0x1010 --> 0x00000000
LCR: @0x1014 --> 0x00000000
LCR: @0x1018 --> 0x00000000
LCR: @0x101c --> 0x00000000
LCR: @0x1020 --> 0x00000000
LCR: @0x1024 --> 0x00000000
LCR: @0x1028 --> 0x00000000
LCR: @0x102c --> 0x00000000
LCR: @0x1030 --> 0x00000000
.
.
.
LCR: @0x1fb0 --> 0x00000000
LCR: @0x1fb4 --> 0x00000000
LCR: @0x1fb8 --> 0x00000000
LCR: @0x1fbc --> 0x00000000
LCR: @0x1fc0 --> 0x00000000
LCR: @0x1fc4 --> 0x00000000
LCR: @0x1fc8 --> 0x00000000
LCR: @0x1fcc --> 0x00000000
LCR: @0x1fd0 --> 0x00000000
LCR: @0x1fd4 --> 0x00000000

```

```
LCR: @0x1fd8 --> 0x00000000
LCR: @0x1fdc --> 0x00000000
LCR: @0x1fe0 --> 0x00000000
LCR: @0x1fe4 --> 0x00000000
LCR: @0x1fe8 --> 0x00000000
LCR: @0x1fec --> 0x00000000
LCR: @0x1ff0 --> 0x00000000
LCR: @0x1ff4 --> 0x00000000
LCR: @0x1ff8 --> 0x00000000
LCR: @0x1ffc --> 0x00000000
```

CONFIG REGS (DMA Control Table)

```
LCR: @0x4000 --> 0x00000011
LCR: @0x4004 --> 0x0000700c
LCR: @0x4008 --> 0x008ac000
LCR: @0x400c --> 0x00000000
LCR: @0x4010 --> 0x00000000
LCR: @0x4014 --> 0x00000000
LCR: @0x4018 --> 0x00000000
LCR: @0x401c --> 0x00000000
LCR: @0x4020 --> 0x00000011
LCR: @0x4024 --> 0x0000fff0
LCR: @0x4028 --> 0x00827ff0
LCR: @0x402c --> 0x00000000
LCR: @0x4030 --> 0x00000000
LCR: @0x4034 --> 0x00000000
LCR: @0x4038 --> 0x00000000
LCR: @0x403c --> 0x00000000
```

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

```
PCR: @0x0000 --> 0x93501542
PCR: @0x0004 --> 0x00100406
PCR: @0x0008 --> 0x08800001
PCR: @0x000c --> 0x00000010
PCR: @0x0010 --> 0xbd520000
PCR: @0x0014 --> 0x00000000
PCR: @0x0018 --> 0xbd500000
PCR: @0x001c --> 0x00000000
PCR: @0x0020 --> 0x00000000
PCR: @0x0024 --> 0x00000000
PCR: @0x0028 --> 0x00000000
PCR: @0x002c --> 0x01001542
PCR: @0x0030 --> 0x00000000
PCR: @0x0034 --> 0x00000050
PCR: @0x0038 --> 0x00000000
PCR: @0x003c --> 0x0000010b
PCR: @0x0040 --> 0x00000000
PCR: @0x0044 --> 0x02006160
PCR: @0x0048 --> 0x00000000
PCR: @0x004c --> 0x00000000
PCR: @0x0050 --> 0x00857805
PCR: @0x0054 --> 0xfeeff00c
PCR: @0x0058 --> 0x00000000
PCR: @0x005c --> 0x00004165
PCR: @0x0060 --> 0x00000000
PCR: @0x0064 --> 0x00000000
PCR: @0x0068 --> 0x00007811
```

```

PCR: @0x006c --> 0x00000000
PCR: @0x0070 --> 0x00000000
PCR: @0x0074 --> 0x00000000
PCR: @0x0078 --> 0x00038001
PCR: @0x007c --> 0x00000000
PCR: @0x0080 --> 0x00020010
PCR: @0x0084 --> 0x00648001
PCR: @0x0088 --> 0x00002830
PCR: @0x008c --> 0x01406441
PCR: @0x0090 --> 0x10410040
PCR: @0x0094 --> 0x00000000
PCR: @0x0098 --> 0x00000000
PCR: @0x009c --> 0x00000000
PCR: @0x00a0 --> 0x00000000
PCR: @0x00a4 --> 0x0000001f
PCR: @0x00a8 --> 0x0000000d
PCR: @0x00ac --> 0x00000000
PCR: @0x00b0 --> 0x00010001
PCR: @0x00b4 --> 0x00000000
PCR: @0x00b8 --> 0x00000000
PCR: @0x00bc --> 0x00000000
PCR: @0x00c0 --> 0x00000000
PCR: @0x00c4 --> 0x00000000
PCR: @0x00c8 --> 0x00000000
PCR: @0x00cc --> 0x00000000
PCR: @0x00d0 --> 0x00000000
PCR: @0x00d4 --> 0x00000000
PCR: @0x00d8 --> 0x00000000
PCR: @0x00dc --> 0x00000000
PCR: @0x00e0 --> 0x00000000
PCR: @0x00e4 --> 0x00000000
PCR: @0x00e8 --> 0x00000000
PCR: @0x00ec --> 0x00000000
PCR: @0x00f0 --> 0x00000000
PCR: @0x00f4 --> 0x00000000
PCR: @0x00f8 --> 0x00000000
PCR: @0x00fc --> 0x00000000`

```

3.1.3 ccrtaiicc_rdreg

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

```

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

```

Example display:

```
./ccrtaiicc_rdreg -s64
```

```
Device Name: /dev/ccrtaiicc0
```

```

LOCAL REGION: Physical Addr=0xbd500000 Size=131072 (0x00020000)
CONFIG REGION: Physical Addr=0xbd520000 Size=32768 (0x00008000)

```

```

LOCAL: Register 0x7ffff7fd7000 Offset=0x0 Size=0x00020000
CONFIG: Register 0x7ffff7fcf000 Offset=0x0 Size=0x00008000

```

```

LIBPTR: Register 0x7ffff7fcd000 Offset=0x0 Size=0x00001008

#### LOCAL REGS #### (length=64)
+LCL+    0    93500101 12052018 00010000 00000000 *.P....*
+LCL+   0x10 00000000 00000000 00000000 00000000 *.....*
+LCL+   0x20 00000000 00000000 00000000 00000000 *.....*
+LCL+   0x30 00000000 00000000 00000000 00000000 *.....*
    18.167us ( 3.52 MB/s)

./ccrtaicc_rdreg -C -o4020 -s20

Device Name: /dev/ccrtaicc0

LOCAL REGION: Physical Addr=0xbd500000 Size=131072 (0x00020000)
CONFIG REGION: Physical Addr=0xbd520000 Size=32768 (0x00008000)

LOCAL: Register 0x7ffff7fd7000 Offset=0x0 Size=0x00020000
CONFIG: Register 0x7ffff7fcf000 Offset=0x0 Size=0x00008000
LIBPTR: Register 0x7ffff7fcd000 Offset=0x0 Size=0x00001008

#### CONFIG REGS #### (length=20)
+CFG+   0x4020 00000011 0000ffff 00827ff0 00000000 *.....*
+CFG+   0x4030 00000000 *.....*
    5.203us ( 3.84 MB/s)

```

3.1.4 ccrtaicc_reg

This call displays all the boards local and configuration registers.

```

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

```

Example display:

```

./ccrtaicc_reg

Device Name: /dev/ccrtaicc0

LOCAL REGION: Physical Addr=0xbd500000 Size=131072 (0x00020000)
CONFIG REGION: Physical Addr=0xbd520000 Size=32768 (0x00008000)

LOCAL: Register 0x7ffff7fd7000 Offset=0x0 Size=0x00020000
CONFIG: Register 0x7ffff7fcf000 Offset=0x0 Size=0x00008000
LIBPTR: Register 0x7ffff7fcd000 Offset=0x0 Size=0x00001008
LOCAL Register 0x7ffff7fd7000 size=0x00020000

#### LOCAL REGS #### (length=131072)
+LCL+    0    93500101 12052018 00010000 00000000 *.P....*
+LCL+   0x10 00000000 00000000 00000000 00000000 *.....*
+LCL+   0x20 00000000 00000000 00000000 00000000 *.....*
+LCL+   0x30 00000000 00000000 00000000 00000000 *.....*
+LCL+   0x40 00000000 00000000 00000000 00000000 *.....*
+LCL+   0x50 00000000 00000000 00000000 00000000 *.....*
+LCL+   0x60 00000000 00000000 00000000 00000000 *.....*
+LCL+   0x70 00000000 00000000 00000000 00000000 *.....*
+LCL+   0x80 00000000 00000000 00000000 00000000 *.....*
+LCL+   0x90 00000000 00000000 00000000 00000000 *.....*
+LCL+   0xa0 00000000 00000000 00000000 00000000 *.....*
+LCL+   0xb0 00000000 00000000 00000000 00000000 *.....*
+LCL+   0xc0 00000000 00000000 00000000 00000000 *.....*
+LCL+   0xd0 00000000 00000000 00000000 00000000 *.....*
+LCL+   0xe0 00000000 00000000 00000000 00000000 *.....*
+LCL+   0xf0 00000000 00000000 00000000 00000000 *.....*

```

```

+LCL+ 0x100 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x110 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x120 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x130 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x140 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x150 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x160 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x170 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x180 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x190 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1a0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1b0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1c0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1d0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1e0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1f0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x200 00000000 00000000 00000000 00000000 *.....*
.
.
.
+LCL+ 0x1fed0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1fee0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1fef0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ff00 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ff10 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ff20 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ff30 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ff40 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ff50 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ff60 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ff70 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ff80 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ff90 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ffa0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ffb0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ffc0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ffd0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1ffe0 00000000 00000000 00000000 00000000 *.....*
+LCL+ 0x1fff0 00000000 00000000 00000000 00000000 *.....*

```

CONFIG Register 0x7ffff7fcf000 size=0x00008000

```

#### CONFIG REGS (PCIeLinkPartnerRegs) #### (length=4096)
+CFG+ 0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x10 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x20 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x30 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x40 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x50 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x60 00000004 00000004 00000008 00000008 *.....*
+CFG+ 0x70 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x80 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x90 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xa0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xb0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xc0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xd0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xe0 00000004 00000004 00000008 00000008 *.....*
+CFG+ 0xf0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x100 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x110 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x120 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x130 00000000 00000000 00000000 00000000 *.....*

```



```

+CFG+ 0x140 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x150 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x160 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x170 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x180 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x190 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1a0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1b0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1c0 00000000 00000000 00000000 00000000 *.....*
.
.
+CFG+ 0xf00 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xf10 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xf20 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xf30 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xf40 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xf50 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xf60 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xf70 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xf80 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xf90 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xfa0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xfb0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xfc0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xfd0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xfe0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0xff0 00000000 00000000 00000000 00000000 *.....*

#### CONFIG REGS (AvalonMM_2_PCIeAddrTrans) #### (length=4096)
+CFG+ 0x1000 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1010 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1020 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1030 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1040 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1050 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1060 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1070 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1080 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1090 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x10a0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x10b0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x10c0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x10d0 00000000 00000000 00000000 00000000 *.....*
.
.
+CFG+ 0x1f50 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1f60 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1f70 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1f80 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1f90 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1fa0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1fb0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1fc0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1fd0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1fe0 00000000 00000000 00000000 00000000 *.....*
+CFG+ 0x1ff0 00000000 00000000 00000000 00000000 *.....*

#### CONFIG REGS (DMA Control Table) #### (length=64)
+CFG+ 0x4000 00000011 0000700c 008ac000 00000000 *.....p.....*
+CFG+ 0x4010 00000000 00000000 00000000 00000000 *.....*

```

```
+CFG+ 0x4020 00000011 0000ffff 00827ff0 00000000 *.....*
+CFG+ 0x4030 00000000 00000000 00000000 00000000 *.....*
```

===== LOCAL REGISTERS =====

```
BoardInfo =0x93500101 @0x00000000
FirmwareDate =0x12052018 @0x00000004
FirmwareRevision =0x00010000 @0x00000008
FirmwareTime =0x00000000 @0x0000000c
FirmwareFlavorCode =0x00000000 @0x00000010
NumberMsgDmaDescriptors =0x00000000 @0x00000018
BoardCSR =0x00000000 @0x00002000
InterruptStatus =0x00000000 @0x00002010
SPI_CommandStatus =0x03004000 @0x000020f0
SPI_FirmwareAddress =0x01ffff24 @0x000020f4
SPI_Ram[0] =0x80004000 @0x00002100
FPGA_ChipIdentification[0] =0x00f14102 @0x00002400
FPGA_ChipIdentification[1] =0x088a0904 @0x00002404
FPGA_ChipTemperature =0x00000000 @0x00002410
ClockGen_CSR =0x00000003 @0x00002500
ClockGen_access =0x000d00f4 @0x00002504
CalibrationCSR =0x00000000 @0x00002600
TestBusControl =0x00000000 @0x00002604
ADC_InputControl =0x00000000 @0x00002608
ExternalClockCSR =0x00610000 @0x00002700
ADC_Enable =0x00000001 @0x00003000
ADC_ControlStatus[CCRTAICC_ADC_0] =0x00000000 @0x00003010
ADC_ControlStatus[CCRTAICC_ADC_1] =0x00000000 @0x00003014
ADC_ControlStatus[CCRTAICC_ADC_2] =0x00000000 @0x00003018
ADC_ControlStatus[CCRTAICC_ADC_3] =0x00000000 @0x0000301c
ADC_FifoCSR =0x2001ffff @0x00003030
ADC_FifoThreshold =0x0001ffff @0x00003034
ADC_FifoChannelSelect[CCRTAICC_ADC_FIFO_CHANNEL_SELECT_0_31] =0xffffffff @0x00003038
ADC_FifoChannelSelect[CCRTAICC_ADC_FIFO_CHANNEL_SELECT_32_63] =0xffffffff @0x0000303c
ADC_Data[CCRTAICC_ADC_CHANNEL_0] =0x00020004 @0x00003200
ADC_Data[CCRTAICC_ADC_CHANNEL_1] =0x0001ffff @0x00003204
ADC_Data[CCRTAICC_ADC_CHANNEL_2] =0x0001ffff @0x00003208
ADC_Data[CCRTAICC_ADC_CHANNEL_3] =0x0001ffff @0x0000320c
ADC_Data[CCRTAICC_ADC_CHANNEL_4] =0x0001ffff @0x00003210
ADC_Data[CCRTAICC_ADC_CHANNEL_5] =0x0001ffff @0x00003214
ADC_Data[CCRTAICC_ADC_CHANNEL_6] =0x00020007 @0x00003218
ADC_Data[CCRTAICC_ADC_CHANNEL_7] =0x00020001 @0x0000321c
ADC_Data[CCRTAICC_ADC_CHANNEL_8] =0x00020000 @0x00003220
ADC_Data[CCRTAICC_ADC_CHANNEL_9] =0x00020000 @0x00003224
ADC_Data[CCRTAICC_ADC_CHANNEL_10] =0x00020000 @0x00003228
ADC_Data[CCRTAICC_ADC_CHANNEL_11] =0x00020000 @0x0000322c
ADC_Data[CCRTAICC_ADC_CHANNEL_12] =0x0001ffff @0x00003230
ADC_Data[CCRTAICC_ADC_CHANNEL_13] =0x00020001 @0x00003234
ADC_Data[CCRTAICC_ADC_CHANNEL_14] =0x0001ffff @0x00003238
ADC_Data[CCRTAICC_ADC_CHANNEL_15] =0x0001ffff @0x0000323c
ADC_Data[CCRTAICC_ADC_CHANNEL_16] =0x00020007 @0x00003240
ADC_Data[CCRTAICC_ADC_CHANNEL_17] =0x0001ffff @0x00003244
ADC_Data[CCRTAICC_ADC_CHANNEL_18] =0x00020005 @0x00003248
ADC_Data[CCRTAICC_ADC_CHANNEL_19] =0x0001ffff @0x0000324c
ADC_Data[CCRTAICC_ADC_CHANNEL_20] =0x0001ffff @0x00003250
ADC_Data[CCRTAICC_ADC_CHANNEL_21] =0x0001ffff @0x00003254
ADC_Data[CCRTAICC_ADC_CHANNEL_22] =0x00020002 @0x00003258
ADC_Data[CCRTAICC_ADC_CHANNEL_23] =0x0001ffff @0x0000325c
ADC_Data[CCRTAICC_ADC_CHANNEL_24] =0x00020001 @0x00003260
ADC_Data[CCRTAICC_ADC_CHANNEL_25] =0x0001ffff @0x00003264
ADC_Data[CCRTAICC_ADC_CHANNEL_26] =0x0001ffff @0x00003268
```

ADC_Data[CCRTAICC_ADC_CHANNEL_27]	=0x0001fffb	@0x0000326c
ADC_Data[CCRTAICC_ADC_CHANNEL_28]	=0x0001fffd	@0x00003270
ADC_Data[CCRTAICC_ADC_CHANNEL_29]	=0x00020003	@0x00003274
ADC_Data[CCRTAICC_ADC_CHANNEL_30]	=0x00020000	@0x00003278
ADC_Data[CCRTAICC_ADC_CHANNEL_31]	=0x0001fffa	@0x0000327c
ADC_Data[CCRTAICC_ADC_CHANNEL_32]	=0x00020003	@0x00003280
ADC_Data[CCRTAICC_ADC_CHANNEL_33]	=0x0001fffe	@0x00003284
ADC_Data[CCRTAICC_ADC_CHANNEL_34]	=0x0001fffe	@0x00003288
ADC_Data[CCRTAICC_ADC_CHANNEL_35]	=0x0001fffa	@0x0000328c
ADC_Data[CCRTAICC_ADC_CHANNEL_36]	=0x00020000	@0x00003290
ADC_Data[CCRTAICC_ADC_CHANNEL_37]	=0x0001ffff	@0x00003294
ADC_Data[CCRTAICC_ADC_CHANNEL_38]	=0x00020002	@0x00003298
ADC_Data[CCRTAICC_ADC_CHANNEL_39]	=0x00020000	@0x0000329c
ADC_Data[CCRTAICC_ADC_CHANNEL_40]	=0x00020000	@0x000032a0
ADC_Data[CCRTAICC_ADC_CHANNEL_41]	=0x00020001	@0x000032a4
ADC_Data[CCRTAICC_ADC_CHANNEL_42]	=0x00020000	@0x000032a8
ADC_Data[CCRTAICC_ADC_CHANNEL_43]	=0x00020004	@0x000032ac
ADC_Data[CCRTAICC_ADC_CHANNEL_44]	=0x00020003	@0x000032b0
ADC_Data[CCRTAICC_ADC_CHANNEL_45]	=0x00020005	@0x000032b4
ADC_Data[CCRTAICC_ADC_CHANNEL_46]	=0x00020001	@0x000032b8
ADC_Data[CCRTAICC_ADC_CHANNEL_47]	=0x00020003	@0x000032bc
ADC_Data[CCRTAICC_ADC_CHANNEL_48]	=0x00020001	@0x000032c0
ADC_Data[CCRTAICC_ADC_CHANNEL_49]	=0x00020000	@0x000032c4
ADC_Data[CCRTAICC_ADC_CHANNEL_50]	=0x0001ffff	@0x000032c8
ADC_Data[CCRTAICC_ADC_CHANNEL_51]	=0x00020001	@0x000032cc
ADC_Data[CCRTAICC_ADC_CHANNEL_52]	=0x00020005	@0x000032d0
ADC_Data[CCRTAICC_ADC_CHANNEL_53]	=0x00020001	@0x000032d4
ADC_Data[CCRTAICC_ADC_CHANNEL_54]	=0x00020003	@0x000032d8
ADC_Data[CCRTAICC_ADC_CHANNEL_55]	=0x00020000	@0x000032dc
ADC_Data[CCRTAICC_ADC_CHANNEL_56]	=0x0001fffc	@0x000032e0
ADC_Data[CCRTAICC_ADC_CHANNEL_57]	=0x00020003	@0x000032e4
ADC_Data[CCRTAICC_ADC_CHANNEL_58]	=0x00020004	@0x000032e8
ADC_Data[CCRTAICC_ADC_CHANNEL_59]	=0x0001fffe	@0x000032ec
ADC_Data[CCRTAICC_ADC_CHANNEL_60]	=0x00020009	@0x000032f0
ADC_Data[CCRTAICC_ADC_CHANNEL_61]	=0x00020001	@0x000032f4
ADC_Data[CCRTAICC_ADC_CHANNEL_62]	=0x0001fffc	@0x000032f8
ADC_Data[CCRTAICC_ADC_CHANNEL_63]	=0x00020000	@0x000032fc
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_0]	=0x8004959d	@0x00003500
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_1]	=0x7ffeeefe	@0x00003504
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_2]	=0x8003dfb0	@0x00003508
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_3]	=0x7ffbeeca	@0x0000350c
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_4]	=0x8003aeae	@0x00003510
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_5]	=0x800344ac	@0x00003514
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_6]	=0x800014f7	@0x00003518
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_7]	=0x8003cf21	@0x0000351c
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_8]	=0x7ffe4793	@0x00003520
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_9]	=0x7ffe175d	@0x00003524
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_10]	=0x7ffc3bc38	@0x00003528
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_11]	=0x7ffbda07	@0x0000352c
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_12]	=0x80047fda	@0x00003530
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_13]	=0x7ffee98d	@0x00003534
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_14]	=0x8009981b	@0x00003538
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_15]	=0x800af6f4	@0x0000353c
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_16]	=0x80011262	@0x00003540
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_17]	=0x80045ac7	@0x00003544
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_18]	=0x7ffe93b5	@0x00003548
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_19]	=0x8006d462	@0x0000354c
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_20]	=0x80038314	@0x00003550
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_21]	=0x8006361d	@0x00003554
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_22]	=0x80056e8d	@0x00003558
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_23]	=0x80081733	@0x0000355c
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_24]	=0x80039e55	@0x00003560
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_25]	=0x8001dd2e	@0x00003564

ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_26]	=0x8001d279	@0x00003568
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_27]	=0x8005a9c4	@0x0000356c
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_28]	=0x80012070	@0x00003570
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_29]	=0x7ffdbffc	@0x00003574
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_30]	=0x8008c961	@0x00003578
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_31]	=0x800789a5	@0x0000357c
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_32]	=0x8002fc4a	@0x00003580
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_33]	=0x8006028d	@0x00003584
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_34]	=0x8004af63	@0x00003588
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_35]	=0x80087407	@0x0000358c
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_36]	=0x800178d0	@0x00003590
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_37]	=0x80069cb9	@0x00003594
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_38]	=0x7fff77a0	@0x00003598
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_39]	=0x7fff8169	@0x0000359c
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_40]	=0x80031b3a	@0x000035a0
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_41]	=0x8001049e	@0x000035a4
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_42]	=0x8005dcd7	@0x000035a8
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_43]	=0x80032bf0	@0x000035ac
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_44]	=0x8008fc4c	@0x000035b0
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_45]	=0x8002dd6c	@0x000035b4
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_46]	=0x800547f7	@0x000035b8
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_47]	=0x8002ff43	@0x000035bc
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_48]	=0x8006bef4	@0x000035c0
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_49]	=0x8001ee93	@0x000035c4
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_50]	=0x8003d0f3	@0x000035c8
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_51]	=0x7ffe7bbe	@0x000035cc
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_52]	=0x8006c500	@0x000035d0
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_53]	=0x8006e672	@0x000035d4
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_54]	=0x800434e0	@0x000035d8
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_55]	=0x80014e4e	@0x000035dc
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_56]	=0x80070db7	@0x000035e0
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_57]	=0x8008bc6b	@0x000035e4
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_58]	=0x8007cef9	@0x000035e8
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_59]	=0x80098a6c	@0x000035ec
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_60]	=0x8002ae77	@0x000035f0
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_61]	=0x80048787	@0x000035f4
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_62]	=0x80031401	@0x000035f8
ADC_PositiveCalibration[CCRTAICC_ADC_CHANNEL_63]	=0x80091408	@0x000035fc
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_0]	=0x8005d6d6	@0x00003600
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_1]	=0x7ffe7ffa	@0x00003604
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_2]	=0x80042ef8	@0x00003608
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_3]	=0x7ffbfb50	@0x0000360c
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_4]	=0x8003ee19	@0x00003610
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_5]	=0x8002f701	@0x00003614
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_6]	=0x8001357a	@0x00003618
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_7]	=0x8004b664	@0x0000361c
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_8]	=0x7ffeeca7	@0x00003620
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_9]	=0x7ffe5519	@0x00003624
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_10]	=0x7ffd6b9a	@0x00003628
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_11]	=0x7ffc2d8c	@0x0000362c
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_12]	=0x8004eb9e	@0x00003630
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_13]	=0x7fff3d16	@0x00003634
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_14]	=0x800a5d9a	@0x00003638
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_15]	=0x800b7359	@0x0000363c
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_16]	=0x80016de6	@0x00003640
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_17]	=0x800371a5	@0x00003644
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_18]	=0x7fff4e3e	@0x00003648
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_19]	=0x80063a52	@0x0000364c
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_20]	=0x8003d183	@0x00003650
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_21]	=0x8005b749	@0x00003654
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_22]	=0x80066c72	@0x00003658
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_23]	=0x800832d3	@0x0000365c
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_24]	=0x80037d87	@0x00003660

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_25]	=0x8001ab7c	@0x00003664
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_26]	=0x800186a7	@0x00003668
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_27]	=0x800574de	@0x0000366c
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_28]	=0x8001431f	@0x00003670
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_29]	=0x7ffd9805	@0x00003674
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_30]	=0x800923ad	@0x00003678
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_31]	=0x80074d0d	@0x0000367c
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_32]	=0x800348a5	@0x00003680
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_33]	=0x8005b1c8	@0x00003684
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_34]	=0x8004486c	@0x00003688
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_35]	=0x8007e53e	@0x0000368c
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_36]	=0x800172a5	@0x00003690
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_37]	=0x80061e9d	@0x00003694
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_38]	=0x7fffdc94	@0x00003698
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_39]	=0x7ffee5b0	@0x0000369c
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_40]	=0x80027472	@0x000036a0
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_41]	=0x80009319	@0x000036a4
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_42]	=0x8005e276	@0x000036a8
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_43]	=0x800298cc	@0x000036ac
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_44]	=0x80097519	@0x000036b0
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_45]	=0x80027131	@0x000036b4
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_46]	=0x8005d3fd	@0x000036b8
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_47]	=0x8002de03	@0x000036bc
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_48]	=0x80062cc2	@0x000036c0
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_49]	=0x8000a24b	@0x000036c4
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_50]	=0x800401eb	@0x000036c8
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_51]	=0x7ffdeccf	@0x000036cc
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_52]	=0x80077663	@0x000036d0
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_53]	=0x8006d3ff	@0x000036d4
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_54]	=0x80041297	@0x000036d8
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_55]	=0x80010e93	@0x000036dc
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_56]	=0x800685c6	@0x000036e0
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_57]	=0x8007d1d9	@0x000036e4
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_58]	=0x80079571	@0x000036e8
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_59]	=0x8008e3c5	@0x000036ec
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_60]	=0x8002e763	@0x000036f0
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_61]	=0x80047caa	@0x000036f4
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_62]	=0x8002a288	@0x000036f8
ADC_NegativeCalibration[CCRTAICC_ADC_CHANNEL_63]	=0x8008ee50	@0x000036fc
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_0]	=0x0003ffff	@0x00003700
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_1]	=0x0003ffff	@0x00003704
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_2]	=0x0003ffff	@0x00003708
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_3]	=0x00000000	@0x0000370c
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_4]	=0x0003ffff	@0x00003710
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_5]	=0x0003ffff	@0x00003714
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_6]	=0x0003ffff	@0x00003718
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_7]	=0x0003ffff	@0x0000371c
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_8]	=0x0003ffff	@0x00003720
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_9]	=0x0003ffff	@0x00003724
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_10]	=0x0003ffff	@0x00003728
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_11]	=0x0003ffff	@0x0000372c
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_12]	=0x0003ffff	@0x00003730
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_13]	=0x0003ffff	@0x00003734
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_14]	=0x0003ffff	@0x00003738
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_15]	=0x0003ffff	@0x0000373c
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_16]	=0x0003ffff	@0x00003740
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_17]	=0x00000001	@0x00003744
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_18]	=0x0003ffff	@0x00003748
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_19]	=0x00000001	@0x0000374c
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_20]	=0x0003ffff	@0x00003750
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_21]	=0x00000001	@0x00003754
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_22]	=0x0003ffff	@0x00003758
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_23]	=0x0003ffff	@0x0000375c

```

ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_24] =0x0003ffff @0x00003760
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_25] =0x0003ffff @0x00003764
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_26] =0x0003ffff @0x00003768
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_27] =0x00000001 @0x0000376c
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_28] =0x0003ffff @0x00003770
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_29] =0x00000000 @0x00003774
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_30] =0x0003ffff @0x00003778
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_31] =0x0003ffff @0x0000377c
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_32] =0x0003ffff @0x00003780
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_33] =0x00000001 @0x00003784
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_34] =0x0003ffff @0x00003788
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_35] =0x00000000 @0x0000378c
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_36] =0x00000000 @0x00003790
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_37] =0x00000001 @0x00003794
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_38] =0x0003ffff @0x00003798
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_39] =0x00000001 @0x0000379c
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_40] =0x00000001 @0x000037a0
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_41] =0x00000001 @0x000037a4
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_42] =0x0003ffff @0x000037a8
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_43] =0x00000001 @0x000037ac
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_44] =0x0003ffff @0x000037b0
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_45] =0x00000001 @0x000037b4
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_46] =0x0003ffff @0x000037b8
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_47] =0x0003ffff @0x000037bc
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_48] =0x00000001 @0x000037c0
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_49] =0x00000002 @0x000037c4
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_50] =0x0003ffff @0x000037c8
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_51] =0x00000001 @0x000037cc
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_52] =0x0003ffff @0x000037d0
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_53] =0x00000000 @0x000037d4
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_54] =0x00000000 @0x000037d8
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_55] =0x00000001 @0x000037dc
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_56] =0x00000000 @0x000037e0
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_57] =0x00000002 @0x000037e4
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_58] =0x00000000 @0x000037e8
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_59] =0x00000002 @0x000037ec
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_60] =0x0003ffff @0x000037f0
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_61] =0x0003ffff @0x000037f4
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_62] =0x0003ffff @0x000037f8
ADC_OffsetCalibration[CCRTAICC_ADC_CHANNEL_63] =0x0003ffff @0x000037fc
ADC_FifoData =0x0501ffff @0x00003400
SDRAM_Enable =0x00000000 @0x00007000
SDRAM_CSR =0x00000000 @0x00007004
SDRAM_Address =0x00000000 @0x00007008
SDRAM_Data =0x00000000 @0x0000700c
DiagRam[0] =0xfac0000 @0x00008000
FpgawbRevision =0x00000000 @0x0001f000

```

===== CONFIG REGISTERS =====

```

PcieLinkPartners.a2p_interrupt_status =0x00000000 @0x00000040
PcieLinkPartners.a2p_interrupt_enable =0x00000000 @0x00000050

```

PCIe Link Partners (p2a_mailbox) #### (length=32)

```

+P2A+ 0x800 00000000 00000000 00000000 00000000 *.....*
+P2A+ 0x810 00000000 00000000 00000000 00000000 *.....*

```

PCIe Link Partners (a2p_mailbox) #### (length=32)

```

+A2P+ 0x900 00000000 00000000 00000000 00000000 *.....*
+A2P+ 0x910 00000000 00000000 00000000 00000000 *.....*

```

```

DMAEngine[CCRTAICC_DMA0].dma_status =0x00000011 @0x00004000
DMAEngine[CCRTAICC_DMA0].dma_readaddress =0x0000700c @0x00004004
DMAEngine[CCRTAICC_DMA0].dma_writeaddress =0x008ac000 @0x00004008
DMAEngine[CCRTAICC_DMA0].dma_length =0x00000000 @0x0000400c

```

```

DMAEngine[CCRTAICC_DMA0].dma_control           =0x00000000 @0x00004018

DMAEngine[CCRTAICC_DMA1].dma_status           =0x00000011 @0x00004020
DMAEngine[CCRTAICC_DMA1].dma_readaddress      =0x0000fff0 @0x00004024
DMAEngine[CCRTAICC_DMA1].dma_writeaddress     =0x00827ff0 @0x00004028
DMAEngine[CCRTAICC_DMA1].dma_length          =0x00000000 @0x0000402c
DMAEngine[CCRTAICC_DMA1].dma_control         =0x00000000 @0x00004038

MsgDmaDispatcherCsr.Status                   =0x00000000 @0x00004200
MsgDmaDispatcherCsr.Control                  =0x00000000 @0x00004204
MsgDmaDispatcherCsr.ReadFillLevel            =0x00000000 @0x00004208
MsgDmaDispatcherCsr.WriteFillLevel           =0x00000000 @0x0000420a
MsgDmaDispatcherCsr.ResponseFillLevel        =0x00000000 @0x0000420c
MsgDmaDispatcherCsr.ReadSequenceNumber       =0x00000000 @0x00004210
MsgDmaDispatcherCsr.WriteSequenceNumber      =0x00000000 @0x00004212

MsgDmaPrefetcherCsr.Control                  =0x00000000 @0x00004220
MsgDmaPrefetcherCsr.NextDescriptorPointerLow =0x00000000 @0x00004224
MsgDmaPrefetcherCsr.NextDescriptorPointerHigh =0x00000000 @0x00004228
MsgDmaPrefetcherCsr.DescriptorPollingFrequency =0x00000000 @0x0000422c
MsgDmaPrefetcherCsr.Status                   =0x00000000 @0x00004230

=== Descriptor at offset 0 ===
MsgDmaExtendedDescriptor[Id].ReadAddressLow   =0x00000000 @0x00004800
MsgDmaExtendedDescriptor[Id].WriteAddressLow  =0x00000000 @0x00004804
MsgDmaExtendedDescriptor[Id].Length           =0x00000000 @0x00004808
MsgDmaExtendedDescriptor[Id].NextDescriptorPointerLow =0x00000000 @0x0000480c
MsgDmaExtendedDescriptor[Id].ActualBytesTransferred =0x00000000 @0x00004810
MsgDmaExtendedDescriptor[Id].Status           =0x00000000 @0x00004814
MsgDmaExtendedDescriptor[Id].SequenceNumber   =0x00000000 @0x0000481c
MsgDmaExtendedDescriptor[Id].ReadBurstCount   =0x00000000 @0x0000481e
MsgDmaExtendedDescriptor[Id].WriteBurstCount  =0x00000000 @0x0000481f
MsgDmaExtendedDescriptor[Id].ReadStride        =0x00000000 @0x00004820
MsgDmaExtendedDescriptor[Id].WriteStride       =0x00000000 @0x00004822
MsgDmaExtendedDescriptor[Id].ReadAddressHigh  =0x00000000 @0x00004824
MsgDmaExtendedDescriptor[Id].WriteAddressHigh  =0x00000000 @0x00004828
MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh =0x00000000 @0x0000482c
MsgDmaExtendedDescriptor[Id].Control          =0x00000000 @0x0000483c

=== Descriptor at offset 1 ===
MsgDmaExtendedDescriptor[Id].ReadAddressLow   =0x00000000 @0x00004840
MsgDmaExtendedDescriptor[Id].WriteAddressLow  =0x00000000 @0x00004844
MsgDmaExtendedDescriptor[Id].Length           =0x00000000 @0x00004848
MsgDmaExtendedDescriptor[Id].NextDescriptorPointerLow =0x00000000 @0x0000484c
MsgDmaExtendedDescriptor[Id].ActualBytesTransferred =0x00000000 @0x00004850
MsgDmaExtendedDescriptor[Id].Status           =0x00000000 @0x00004854
MsgDmaExtendedDescriptor[Id].SequenceNumber   =0x00000000 @0x0000485c
MsgDmaExtendedDescriptor[Id].ReadBurstCount   =0x00000000 @0x0000485e
MsgDmaExtendedDescriptor[Id].WriteBurstCount  =0x00000000 @0x0000485f
MsgDmaExtendedDescriptor[Id].ReadStride        =0x00000000 @0x00004860
MsgDmaExtendedDescriptor[Id].WriteStride       =0x00000000 @0x00004862
MsgDmaExtendedDescriptor[Id].ReadAddressHigh  =0x00000000 @0x00004864
MsgDmaExtendedDescriptor[Id].WriteAddressHigh  =0x00000000 @0x00004868
MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh =0x00000000 @0x0000486c
MsgDmaExtendedDescriptor[Id].Control          =0x00000000 @0x0000487c

=== Descriptor at offset 2 ===
MsgDmaExtendedDescriptor[Id].ReadAddressLow   =0x00000000 @0x00004880
MsgDmaExtendedDescriptor[Id].WriteAddressLow  =0x00000000 @0x00004884
MsgDmaExtendedDescriptor[Id].Length           =0x00000000 @0x00004888

```

```

MsgDmaExtendedDescriptor [Id] .NextDescriptorPointerLow
                                =0x00000000 @0x0000488c
MsgDmaExtendedDescriptor [Id] .ActualBytesTransferred=0x00000000 @0x00004890
MsgDmaExtendedDescriptor [Id] .Status                =0x00000000 @0x00004894
MsgDmaExtendedDescriptor [Id] .SequenceNumber        =0x00000000 @0x0000489c
MsgDmaExtendedDescriptor [Id] .ReadBurstCount        =0x00000000 @0x0000489e
MsgDmaExtendedDescriptor [Id] .WriteBurstCount       =0x00000000 @0x0000489f
MsgDmaExtendedDescriptor [Id] .ReadStride            =0x00000000 @0x000048a0
MsgDmaExtendedDescriptor [Id] .WriteStride           =0x00000000 @0x000048a2
MsgDmaExtendedDescriptor [Id] .ReadAddressHigh       =0x00000000 @0x000048a4
MsgDmaExtendedDescriptor [Id] .WriteAddressHigh      =0x00000000 @0x000048a8
MsgDmaExtendedDescriptor [Id] .NextDescriptorPointerHigh
                                =0x00000000 @0x000048ac
MsgDmaExtendedDescriptor [Id] .Control                =0x00000000 @0x000048bc

=== Descriptor at offset 3 ===
MsgDmaExtendedDescriptor [Id] .ReadAddressLow        =0x00000000 @0x000048c0
MsgDmaExtendedDescriptor [Id] .WriteAddressLow       =0x00000000 @0x000048c4
MsgDmaExtendedDescriptor [Id] .Length               =0x00000000 @0x000048c8
MsgDmaExtendedDescriptor [Id] .NextDescriptorPointerLow
                                =0x00000000 @0x000048cc
MsgDmaExtendedDescriptor [Id] .ActualBytesTransferred=0x00000000 @0x000048d0
MsgDmaExtendedDescriptor [Id] .Status                =0x00000000 @0x000048d4
MsgDmaExtendedDescriptor [Id] .SequenceNumber        =0x00000000 @0x000048dc
MsgDmaExtendedDescriptor [Id] .ReadBurstCount        =0x00000000 @0x000048de
MsgDmaExtendedDescriptor [Id] .WriteBurstCount       =0x00000000 @0x000048df
MsgDmaExtendedDescriptor [Id] .ReadStride            =0x00000000 @0x000048e0
MsgDmaExtendedDescriptor [Id] .WriteStride           =0x00000000 @0x000048e2
MsgDmaExtendedDescriptor [Id] .ReadAddressHigh       =0x00000000 @0x000048e4
MsgDmaExtendedDescriptor [Id] .WriteAddressHigh      =0x00000000 @0x000048e8
MsgDmaExtendedDescriptor [Id] .NextDescriptorPointerHigh
                                =0x00000000 @0x000048ec
MsgDmaExtendedDescriptor [Id] .Control                =0x00000000 @0x000048fc
.
.
.
=== Descriptor at offset 29 ===
MsgDmaExtendedDescriptor [Id] .ReadAddressLow        =0x00000000 @0x00004f40
MsgDmaExtendedDescriptor [Id] .WriteAddressLow       =0x00000000 @0x00004f44
MsgDmaExtendedDescriptor [Id] .Length               =0x00000000 @0x00004f48
MsgDmaExtendedDescriptor [Id] .NextDescriptorPointerLow
                                =0x00000000 @0x00004f4c
MsgDmaExtendedDescriptor [Id] .ActualBytesTransferred=0x00000000 @0x00004f50
MsgDmaExtendedDescriptor [Id] .Status                =0x00000000 @0x00004f54
MsgDmaExtendedDescriptor [Id] .SequenceNumber        =0x00000000 @0x00004f5c
MsgDmaExtendedDescriptor [Id] .ReadBurstCount        =0x00000000 @0x00004f5e
MsgDmaExtendedDescriptor [Id] .WriteBurstCount       =0x00000000 @0x00004f5f
MsgDmaExtendedDescriptor [Id] .ReadStride            =0x00000000 @0x00004f60
MsgDmaExtendedDescriptor [Id] .WriteStride           =0x00000000 @0x00004f62
MsgDmaExtendedDescriptor [Id] .ReadAddressHigh       =0x00000000 @0x00004f64
MsgDmaExtendedDescriptor [Id] .WriteAddressHigh      =0x00000000 @0x00004f68
MsgDmaExtendedDescriptor [Id] .NextDescriptorPointerHigh
                                =0x00000000 @0x00004f6c
MsgDmaExtendedDescriptor [Id] .Control                =0x00000000 @0x00004f7c

=== Descriptor at offset 30 ===
MsgDmaExtendedDescriptor [Id] .ReadAddressLow        =0x00000000 @0x00004f80
MsgDmaExtendedDescriptor [Id] .WriteAddressLow       =0x00000000 @0x00004f84
MsgDmaExtendedDescriptor [Id] .Length               =0x00000000 @0x00004f88
MsgDmaExtendedDescriptor [Id] .NextDescriptorPointerLow
                                =0x00000000 @0x00004f8c
MsgDmaExtendedDescriptor [Id] .ActualBytesTransferred=0x00000000 @0x00004f90
MsgDmaExtendedDescriptor [Id] .Status                =0x00000000 @0x00004f94

```



```

MsgDmaExtendedDescriptor[Id].SequenceNumber      =0x00000000 @0x00004f9c
MsgDmaExtendedDescriptor[Id].ReadBurstCount      =0x00000000 @0x00004f9e
MsgDmaExtendedDescriptor[Id].WriteBurstCount     =0x00000000 @0x00004f9f
MsgDmaExtendedDescriptor[Id].ReadStride          =0x00000000 @0x00004fa0
MsgDmaExtendedDescriptor[Id].WriteStride         =0x00000000 @0x00004fa2
MsgDmaExtendedDescriptor[Id].ReadAddressHigh     =0x00000000 @0x00004fa4
MsgDmaExtendedDescriptor[Id].WriteAddressHigh    =0x00000000 @0x00004fa8
MsgDmaExtendedDescriptor[Id].NextDescriptorPointerHigh
                                                    =0x00000000 @0x00004fac
MsgDmaExtendedDescriptor[Id].Control             =0x00000000 @0x00004fbc

=== Terminating Descriptor at offset 31 ===
MsgDmaTerminatingDescriptor.ReadAddressLow       =0x00000000 @0x00004fc0
MsgDmaTerminatingDescriptor.WriteAddressLow     =0x00000000 @0x00004fc4
MsgDmaTerminatingDescriptor.Length              =0x00000000 @0x00004fc8
MsgDmaTerminatingDescriptor.NextDescriptorPointerLow
                                                    =0x00000000 @0x00004fcc
MsgDmaTerminatingDescriptor.ActualBytesTransferred
                                                    =0x00000000 @0x00004fd0
MsgDmaTerminatingDescriptor.Status               =0x00000000 @0x00004fd4
MsgDmaTerminatingDescriptor.SequenceNumber      =0x00000000 @0x00004fdc
MsgDmaTerminatingDescriptor.ReadBurstCount      =0x00000000 @0x00004fde
MsgDmaTerminatingDescriptor.WriteBurstCount     =0x00000000 @0x00004fdf
MsgDmaTerminatingDescriptor.ReadStride          =0x00000000 @0x00004fe0
MsgDmaTerminatingDescriptor.WriteStride         =0x00000000 @0x00004fe2
MsgDmaTerminatingDescriptor.ReadAddressHigh     =0x00000000 @0x00004fe4
MsgDmaTerminatingDescriptor.WriteAddressHigh    =0x00000000 @0x00004fe8
MsgDmaTerminatingDescriptor.NextDescriptorPointerHigh
                                                    =0x00000000 @0x00004fec
MsgDmaTerminatingDescriptor.Control             =0x00000000 @0x00004ffc

```

3.1.5 ccrtaiicc_regedit

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

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

Example display:

```

./ccrtaiicc_regedit

Device Name: /dev/ccrtaiicc0

LOCAL REGION: Physical Addr=0xbd500000 Size=131072 (0x00020000)
CONFIG REGION: Physical Addr=0xbd520000 Size=32768 (0x00008000)

LOCAL: Register 0x7ffff7fd7000 Offset=0x0 Size=0x00020000
CONFIG: Register 0x7ffff7fcf000 Offset=0x0 Size=0x00008000
LIBPTR: Register 0x7ffff7fcd000 Offset=0x0 Size=0x00001008

Initialize_Board: Firmware Rev. 0x10000 successful

Virtual Address: 0x7ffff7fd7000
1 = Create Physical Memory          2 = Destroy Physical memory
3 = Display Channel Data            4 = Display Driver Information
5 = Display Physical Memory Info    6 = Display Registers (CONFIG)
7 = Display Registers (LOCAL)      8 = Dump Physical Memory
9 = Reset Board                    10 = Write Register (LOCAL)
11 = Write Register (CONFIG)       12 = Write Physical Memory

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

```

3.1.6 ccrtaiicc_tst

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

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

Example display:

```
./ccrtaiicc_tst

Device Name: /dev/ccrtaiicc0

LOCAL REGION: Physical Addr=0xbd500000 Size=131072 (0x00020000)
CONFIG REGION: Physical Addr=0xbd520000 Size=32768 (0x00008000)

LOCAL: Register 0x7ffff7fd7000 Offset=0x0 Size=0x00020000
CONFIG: Register 0x7ffff7fcf000 Offset=0x0 Size=0x00008000
LIBPTR: Register 0x7ffff7fcd000 Offset=0x0 Size=0x00001008
Initialize_Board: Firmware Rev. 0x10000 successful

01 = add irq                02 = disable pci interrupts
03 = enable pci interrupts  04 = get device error
05 = get driver info        06 = get physical memory
07 = init board             08 = mmap select
09 = mmap(CONFIG registers) 09 = mmap(LOCAL registers)
11 = mmap(physical memory) 12 = munmap(physical memory)
13 = no command             14 = read operation
15 = remove irq             16 = reset board
17 = restore config registers 18 = write operation

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

3.1.7 ccrtaiicc_wreg

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

```
Usage: ./ccrtaiicc_wreg [-b Board] [-C] [-o Offset] [-s Size] [-v Value] [-x]
-b Board   : Board selection -- default board is 0
-C         : Select Config Registers instead of Local Registers
-o Offset  : Hex offset to write to -- default offset is 0x0
-s Size    : Number of bytes to write in decimal -- default size is 0x4
-v Value   : Hex value to write at offset -- default value is 0x0
-x         : Do not read back just written values -- default read back values
```

Example display:

```
./ccrtaiicc_wreg -v12345678 -o0x8000 -s400

Device Name: /dev/ccrtaiicc0

LOCAL REGION: Physical Addr=0xbd500000 Size=131072 (0x00020000)
CONFIG REGION: Physical Addr=0xbd520000 Size=32768 (0x00008000)

LOCAL: Register 0x7ffff7fd7000 Offset=0x0 Size=0x00020000
CONFIG: Register 0x7ffff7fcf000 Offset=0x0 Size=0x00008000
LIBPTR: Register 0x7ffff7fcd000 Offset=0x0 Size=0x00001008

Writing 0x12345678 to offset 0x8000 for 400 bytes

#### LOCAL REGS #### (length=400)
+LCL+  0x8000  12345678  12345678  12345678  12345678  *.4Vx.4Vx.4Vx.4Vx*
```

```

+LCL+ 0x8010 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8020 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8030 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8040 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8050 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8060 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8070 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8080 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8090 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80a0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80b0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80c0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80d0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80e0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x80f0 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8100 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8110 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8120 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8130 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8140 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8150 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8160 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8170 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+ 0x8180 12345678 12345678 12345678 12345678 *.4Vx.4Vx.4Vx.4Vx*

```

3.1.8 Flash/ccrtaicc_flash

This program is used to burn new firmware. This must only be done at the direction of Concurrent Real-Time support team, otherwise, they could render the board useless.

```

./ccrtaicc_flash -b[Board] -B -F[!] -i -L -q -Q -r[OutFile] -R -v -w[InFile] -X
-b [Board]          : Board number. Must be specified
-B                 : Reload Base Level Firmware if MultiFirmware support present
-F                 : Force Read Flash: Overwrite output file if exists
-F                 : Force Write Flash: Do not abort Flash burn for header label
                    mismatch
-F!                : Force Write Flash: Serious override required to continue
                    burning
-i                 : Query chip, on-board flash and InFile if specified
-L                 : Update License only. (default is to update entire firmware)
-q                 : Quite (non-interactive) mode
-Q                 : Quite (non-interactive) mode. Also dump FPGAWB message
-r                 : Read Flash and write to output file created by
                    ./ccrtaicc_flash
-r [OutFile]       : Read Flash and write to output file 'OutFile'
-R                 : Reload Firmware at sector address in Flash
-R [SectorNumber] : Reload Firmware at sector address 'SectorNumber'
-v                 : Enable verbose mode
-w [InFile]        : Read input FPGA file and Flash the board
-X                 : Use Full File. Do not truncate for firmware write

```

```

===== Notes =====
Board must be specified. Use '-b' option
Query option '-i' not allowed with '-B', '-R#', '-L', 'r' or '-X' options
Firmware reload '-B' or '-R' not allowed with '-i', '-L', '-r', '-w' or '-X' options
Firmware read flash '-r' not allowed with '-B', '-i', '-L', '-R', '-w' or '-X' options
Base Run Level '-B' or '-R#' option not allowed with '-i', '-L', 'r', '-w' or '-X' options
Must specify write flash option '-w' when License only option '-L' is specified
License only option '-L' not allowed with '-B', '-i', '-R', '-w' or '-X' options
Don't truncate file option '-X' cannot be selected with the license only update '-L' option
Don't truncate file option '-X' can only be used with the '-w' option
Inquiry '-i' can be used '-w' options
=====

```

e.g. ./ccrtaicc_flash -b0 (Query chip and on-board Flash)

```

./ccrtaicc_flash -b0 -i                (Query chip and on-board Flash)
./ccrtaicc_flash -b0 -i -w InFile      (Query chip, on-board Flash and InFile)
./ccrtaicc_flash -b0 -r OutFile        (On-board FPGA ==> OutFile)
./ccrtaicc_flash -b0 -w InFile         (InFile ==> On-board FPGA - use truncated file)
./ccrtaicc_flash -b0 -w InFile -v     (InFile ==> On-board FPGA - use truncated file -
                                       verbose)
./ccrtaicc_flash -b0 -w InFile -X     (InFile ==> On-board FPGA - use entire file)
./ccrtaicc_flash -b0 -w InFile -L     (InFile ==> On-board FPGA - only license updated
                                       - interactive)
./ccrtaicc_flash -b0 -w InFile -L -q  (InFile ==> On-board FPGA - only license updated
                                       - non-interactive)
./ccrtaicc_flash -b0 -R                (Reload Firmware - i.e. power-cycle the card)
                                       - Run Level
./ccrtaicc_flash -b0 -B                (Reload Firmware - i.e. power-cycle the card)
                                       - Base Level
./ccrtaicc_flash -b0 -R -B            (Reload Firmware - i.e. power-cycle the card)
                                       - Base Level
./ccrtaicc_flash -b0 -R 200          (Reload Firmware - i.e. power-cycle the card)
                                       - at sector 200

```

3.1.9 Flash/ccrtaicc_label

This utility is only supplied for those customers that are creating their own firmware and need to install in on a RedHawk system. In its simplest form, the customer will request a License from Concurrent Real-Time for the option to burn their custom firmware. The license file (*.lic) supplied by Concurrent Real-Time, along with the customer firmware (*.rpd) file will be supplied to this utility to create a burnable FPGA file (*.cust), that will be supplied to the *ccrtaicc_flash* utility to burn the firmware on the card.

```

./ccrtaicc_label -d[OutputDirectory] -c[ChipName] -F -i[InputFile]
                  -L[LicenseFile] -m[MemberCode] -o[OutputFile]
                  -S[RunLevelSectorAddress] -t[Tag]
-d [OutputDirectory] : Directory to use for Output File
-c [ChipName]        : Chip Name. One of:
                      EPCQ16 EPCQ32 EPCQ64 EPCQ128 EPCQ256 EPCQ512
                      (This option is mandatory if not specified in
                      license file)
-F                   : Force overwriting of output file if it exists
-i [InputFile]       : Raw input file. (.rpd extension)
-L [LicenseFile]     : License file (.lic extension) to restrict firmware
                      access (this option is mandatory)
                      If '-i' option is not specified, the license file is
                      dumped to stderr
-m [MemberCode]      : Specify Member Code (A1,A3,A5,A7,B1,B3,B5,B7)
                      (This option is mandatory if not specified in
                      license file)
-o [OutputFile]      : Use output file instead of the default file created
                      by the program
-S [RunLevelSectorAddress] : Run Level Sector Address. (This option is mandatory
                      if not specified in license file)
                      : S0=Base Level, S#=Run Level Number
-t [Tag]             : Insert this tag name in the default file created by
                      the program

```

==== Notes ====

- Options '-L' is required. If option '-i' is not specified, license file is dumped
- Options 'c', '-m' and '-S' are required if they have not already been defined in LicenseFile
- You cannot specify a Run Level Sector '-S' with Single Level Firmware '-1' option
- Run Level Sector address of zero '-S0' represents the Base Level Firmware in Multi-Firmware support
- If option '-o' is not specified, the created customer FPGA file name will be as follows:


```

<OutputDirectory>/<InputFile>_<Tag>_<Function>_<ChipName><MemberCode><RunLevel>.cust

```

```

e.g. ./ccrtaicc_label -iraw_file.rpd -L LicenseFile.lic (in its simplest form)
      (output file created is: 'raw_file <Function> <ChipName><MemberCode><RunLevel>.cust')
./ccrtaicc_label -L LicenseFile.lic (this will display licensing information)
./ccrtaicc_label -iraw_RUN_file.rpd -ooutput_file.cust -S100 -L LicenseFile.lic
./ccrtaicc_label -iraw_SINGLE_file.rpd -L LicenseFile.lic
./ccrtaicc_label -iraw_RUN_file.rpd -ooutput_file.cust -S200 -L LicenseFile.lic
./ccrtaicc_label -iraw_BASE_file.rpd -S0 -L LicenseFile.lic
      (Will cause firmware to be loaded at start offset Base Run Level)

./ccrtaicc_label -d[OutputDirectory] -c[ChipName] -F -i[InputFile] -K[FpgawbKey]
                 -L[LicenseFile] -m[MemberCode] -o[OutputFile] -S[RunLevelSectorAddress]
                 -t[Tag]
-d [OutputDirectory]      : Directory to use for Output File
-c [ChipName]             : Chip Name. One of:
                           EPCQ16 EPCQ32 EPCQ64 EPCQ128 EPCQ256 EPCQ512
                           (This option is mandatory if not specified in license file)
-F                        : Force overwriting of output file if it exists
-i [InputFile]            : Raw input file. (.rpd extension)
-K [FpgawbKey]           : Fpgawb Key is required if license contains FPGA workbench
                           restriction
-L [LicenseFile]          : License file (.lic extension) to restrict firmware access (this
                           option is mandatory)
                           If '-i' option is not specified, the license file is dumped to
                           stderr
-m [MemberCode]           : Specify Member Code (C7)
                           (This option is mandatory if not specified in license file)
-o [OutputFile]           : Use output file instead of the default file created by the
                           program
-S [RunLevelSectorAddress] : Run Level Sector Address. (This option is mandatory if not
                           specified in license file)
                           : S0=Base Level, S#=Run Level Number
-t [Tag]                  : Insert this tag name in the default file created by the program

==== Notes ====
- Options '-L' is required. If option '-i' is not specified, license file is dumped
- Options 'c', '-m' and '-S' are required if they have not already been defined in
  LicenseFile
- You cannot specify a Run Level Sector '-S' with Single Level Firmware '-1' option
- Run Level Sector address of zero '-S0' represents the Base Level Firmware in Multi-
  Firmware support
- If option '-o' is not specified, the created customer FPGA file name will be as follows:
  <OutputDirectory>/<InputFile> <Tag> <Function> <ChipName><MemberCode><RunLevel>.cust
- If the license file contains an FPGAWB restrict key, then the '-K' FpgawbKey is required

e.g. ./ccrtaicc_label -iraw_file.rpd -L LicenseFile.lic (in its simplest form)
      (output file created is: 'raw_file <Function> <ChipName><MemberCode><RunLevel>.cust')
./ccrtaicc_label -L LicenseFile.lic (this will display licensing information)
./ccrtaicc_label -iraw_RUN_file.rpd -ooutput_file.cust -S100 -L LicenseFile.lic
./ccrtaicc_label -iraw_SINGLE_file.rpd -L LicenseFile.lic
./ccrtaicc_label -iraw_RUN_file.rpd -ooutput_file.cust -S200 -L LicenseFile.lic
./ccrtaicc_label -iraw_BASE_file.rpd -S0 -L LicenseFile.lic
      (Will cause firmware to be loaded at start offset Base Run Level)

```

3.1.10 Flash/ccrtaicc_dump_license

This utility is allows the customer to dump the license information from a firmware (*.cust) file.

Format: ./ccrtaicc_dump_license <Firmware file>

This utility only dumps the license information from the *.cust file and not the *.lic license file

e.g. ./ccrtaicc_dump_license AICC_EPCQ256C7S150.cust

3.2 Application Program Interface (API) Access Example Tests

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

3.2.1 lib/ccrtaiicc_adc

This test performs validation of the Analog Input ADC card.

```
Usage: ./ccrtaiicc_adc [-A] [-a RollingAve] [-b BoardNo] [-c StartChan,EndChan]
                        [-C AdcUpdateClock] [-d Delay] [-D DMAEngine]
                        [-E ExpInpVolt] [-f DataFormat] [-F DebugFile] [-i]
                        [-j SpeedSelect] [-l LoopCnt] [-m XferMode] [-n NumChans]
                        [-N] [-p ClockTolerance] [-s InputSignal] [-t Compare]
                        [-T TestBus] [-V MaxBoardVolts] [-X ExtClock]
-A                      (Perform Auto Calibration first using reference voltage)
-a RollingAve          (Rolling average -- default "=== None ===")
-b BoardNo             (Board number -- default is 0)
-c StartChan,EndChan  (Select start and end channel numbers -- default 0,63
  -c 7,16              (select channels 7 through 16 for processing
  -c 32                (select channels 32 through 63 for processing
-C AdcUpdateClock     (select ADC update clock, a,b,c,d,e,A,B,C,D,E or 'n|N')
  -C a                 (Ch0..63=Normal Clock 0 set to MAX SPS 500000)
  -C b@20000.0/n       (Ch0..15=Normal Clock 1 at 20000 SPS, Ch16..63=No Clock)
  -C a,b,A,B           (Ch0..15=Normal Clock 0, Ch16..31 Normal Clock 1,
                       Ch32..47 Inverted Clock 0, Ch48..63 Inverted Clock 1)
  -C c,d@350000,e     (Ch0..15=Normal Clock 2 at MAX SPS 500000, Ch16..31
                       Normal Clock 3 at 350000 SPS, Ch 32-63 Normal External
                       Signal
-d Delay              (Delay between screen refresh -- default is 0 milli-
                      seconds)
-D DMA Engine         (DMA Engine number -- default = 0)
-E <ExpInpVolts>@<Tol> (Expected Input Volts@Tolerance -- default Tol=0.005000)
  +@<Tol>              (Positive Calibration Ref Volt@Tolerance)
  -@<Tol>              (Negative Calibration Ref Volt@Tolerance)
  s@<Tol>              (Requires '-s' input signal option to specify voltage
                       Volt@Tolerance)
                      (valid '-s' arguments are 'g','+', '-', 'f', 'n', 't')
-f DataFormat         (select data format, '2' or 'b')
  -f b,2              (Ch0..15=Offset binary, Ch16..63=Two's complement)
  -f 2,b/2,b         (Ch0..15 & Ch32..47=Two's complement, Ch16..31 &
                       Ch48..63=Offset binary)
  -f b                (Ch0..63=Offset binary)
-F DebugFile          (Debug file with menu display -- default "=== None ===")
  #DebugFile          (Debug file without display (only summary) -- default
                       "=== None ===")
  @DebugFile          (Debug file without display -- default "=== None ===")
  ~DebugFile          (For gnuplot, no header or summary -- default
                       "=== None ===")
  @, # or ~           (No debug file and no display -- default "=== None ===")
-i                    (Enable Interrupts -- default = Disable)
-j SpeedSelect        (select data speed, 'n' or 'h')
  -j n,h,n,h         (Ch0..15 & Ch32..47=normal speed, Ch16..31 &
                       Ch48..63=high speed)
  -j h/n             (Ch0..15=normal speed, Ch16..63=high speed)
  -j h                (Ch0..63=high speed)
-l LoopCnt            (Loop count -- default is 0)
-m XferMode           (Transfer Mode -- default = 'DMA Channel')
  -mdp               (Driver: (Channel Registers) PIO mode)
  -mdP               (Driver: (FIFO) PIO mode)
  -mlc               (Library: (Channel Registers) program I/O Fast Memory
                       Copy)
  -mld               (Library: (Channel Registers) DMA mode)
```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

-m1D          (Library: (FIFO) DMA mode)
-m1m          (Library: (Channel Registers) Modular scatter-gather DMA
              mode)
-m1M          (Library: (FIFO) Modular scatter-gather DMA mode)
-m1p          (Library: (Channel Registers) PIO)
-m1P          (Library: (FIFO) PIO mode)
-N            (Open device with O_NONBLOCK flag)
-p ClockTolerance (select clock tolerance in parts/trillion. default is
              system default 0.0070)
-s InputSignal (select input signal, 'e', 'g', '+', '-', 'f', '-n',
              't', 'o')
-s g          (All channels set to ground calibration)
-s e          (All channels set to external input)
-s +          (All channels set to positive 9.91V reference
              calibration)
-s -          (All channels set to negative 9.91V reference
              calibration)
-s f          (All channels set to positive 5V reference calibration)
-s n          (All channels set to negative 5V reference calibration)
-s t          (All channels set to positive 2V reference calibration)
-s o          (Calibration Bus Open)
-t Compare    (Compare two channels for +/- -- default is
              "=== None ===")
-t0,15        (Compare channel 0 and 15 for being in sync)
-t5/7         (Compare channel 5 and 7 for being in sync)
-t12,4@0.500 (Compare channel 4 and 12 for being in sync with 0.5V
              tolerance)
-T TestBus    (Test Bus Control 'b' or 'o'. Exit after programming
              this option)
-T b          (Calibration Bus Control)
-T o          (Open Bus Control)
-V MaxBoardVolts (Voltage range 'b5', 'u5', 'b10' or 'u10')
-V b5,b10     (Ch0..15=+/-5V, Ch16..63=+/-10V)
-V u5/u10/b5,b10 (Ch0..15=+5V, Ch16..31=+10V, Ch32..47=+/-5V,
              Ch48..63=+/-10V)
-V b10        (Ch0..63=+/-10V)
-X[0..3,e|E,n|N] (Board External Clock Output Selection)
              '0..3' - Clock Generator Output Number
              'e|E' - External clock input (redrive)
              'n|N' - No Clock

e.g. ./ccrtaicc_adc -A -Ca@470000.0/b@12345.0 -s+ (Autocal, ADC0=470000Hz, ADC1,
              ADC2, ADC3=12345Hz Positive
              9.91v Cal.)
      ./ccrtaicc_adc -A -Ca -s+ -E+ (Autocal, Max Clock for all 4
              ADC, Positive cal. input,
              validate result)
      ./ccrtaicc_adc -A -Ca -s- -t0,15 -a100 (Autocal, Max Clock for all 4
              ADC, Negative cal. input,
              compare ch0 and ch15,
              rolling ave=100)
      ./ccrtaicc_adc -Ca,b,n -s+ -c15,21 (Max Clock for ADC0 & ADC1,
              ADC2, ADC3 OFF, Positive
              cal. input, display channels
              15 through 21)
      ./ccrtaicc_adc -Ca -Vb10 -s- -Es (Max Clock for all 4 ADC, -
              9.91V input, validate against
              -9.91V)
      ./ccrtaicc_adc -Ca -Vb10 -st -Es (Max Clock for all 4 ADC, +2V
              input, validate against +2V)

```

Example display:

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

./ccrtaiicc_adc -A -Ca@470000,b@12345.0 -s+

local_ptr=0x7ffff7fd7000

Physical Memory Information:
UserPID =9849
PhysMemPtr =0x35dcd000
DriverVirtMemPtr=0xffff880035dcd000
MmappedUserMemPtr=0x7ffff7fcc000
PhysMemSize =0x00001000
PhysMemSizeFreed=0x00000000
EntryInTxTbl =0
NumOfEntriesUsed=1

Auto Calibration started...done. (1.690 seconds)

Board Number [-b]: 0
Channel Selection [-c]: Start=Ch0, End=Ch63 (Number of Active Channels=64)
: Active Channel Mask=0xffffffffffffffff
Update Clock Selected [-C]: Ch00..15 [0]Normal Clock 0 (470000.000 SPS)
: Ch16..31 [01]Normal Clock 1 (12345.000 SPS)
: Ch32..47 [01]Normal Clock 1 (12345.000 SPS)
: Ch48..63 [01]Normal Clock 1 (12345.000 SPS)
Delay [-d]: 0 milli-seconds
DMA Engine [-D]: 0
Expected Input Volts [-E]: === Not Specified ===
Data Format [-f]: Ch00..15 [0]Offset binary
: Ch16..31 [0]Offset binary
: Ch32..47 [0]Offset binary
: Ch48..63 [0]Offset binary
Interrupts [-i]: Disabled
Speed Select [-j]: Ch00..15 [0]Normal Speed
: Ch16..31 [0]Normal Speed
: Ch32..47 [0]Normal Speed
: Ch48..63 [0]Normal Speed
Transfer Mode [-m]: Library: (Channel Registers) DMA I/O
Clock Tolerance [-p]: 0.0070 (parts/trillion)
Input Signal [-s]: [1]Calibration Bus (0x01: Positive 9.91)
Voltage Range [-V]: Ch00..15 [1]+/-10 Volts
: Ch16..31 [1]+/-10 Volts
: Ch32..47 [1]+/-10 Volts
: Ch48..63 [1]+/-10 Volts
External Clock Output [-X]: 7 (No Clock)
Scan Count : 8193
Read Duration (microsecs) : TotalDelta: 18.331 (min= 18.117/max= 32.435/ave= 18.432)

Raw Data #####
[0] [1] [2] [3] [4] [5] [6] [7]
=====
[00 07] 03fb6f 03fb6b 03fb71 03fb71 03fb6d 03fb70 03fb6f 03fb71
[08 15] 03fb72 03fb6f 03fb73 03fb75 03fb73 03fb72 03fb74 03fb70
[16 23] 03fb6d 03fb6e 03fb6b 03fb6c 03fb6f 03fb6c 03fb6e 03fb6c
[24 31] 03fb6c 03fb6e 03fb70 03fb6b 03fb73 03fb6b 03fb6c 03fb71
[32 39] 03fb6b 03fb6e 03fb68 03fb6b 03fb68 03fb6d 03fb6b 03fb6b
[40 47] 03fb6f 03fb6f 03fb6d 03fb6d 03fb6b 03fb6e 03fb6f 03fb6c
[48 55] 03fb69 03fb6d 03fb6e 03fb70 03fb6c 03fb6d 03fb69 03fb6a
[56 63] 03fb6d 03fb6c 03fb6d 03fb6a 03fb71 03fb71 03fb6d 03fb70

Volts #####
[0] [1] [2] [3] [4] [5] [6] [7]
=====
[00 07] +9.9108 +9.9105 +9.9110 +9.9110 +9.9107 +9.9109 +9.9108 +9.9110
[08 15] +9.9110 +9.9108 +9.9111 +9.9113 +9.9111 +9.9110 +9.9112 +9.9109
[16 23] +9.9107 +9.9107 +9.9105 +9.9106 +9.9108 +9.9106 +9.9107 +9.9106
[24 31] +9.9106 +9.9107 +9.9109 +9.9105 +9.9111 +9.9105 +9.9106 +9.9110

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.


```

[32 39] +9.9105 +9.9107 +9.9103 +9.9105 +9.9103 +9.9107 +9.9105 +9.9105
[40 47] +9.9108 +9.9108 +9.9107 +9.9107 +9.9105 +9.9107 +9.9108 +9.9106
[48 55] +9.9104 +9.9107 +9.9107 +9.9109 +9.9106 +9.9107 +9.9104 +9.9104
[56 63] +9.9107 +9.9106 +9.9107 +9.9104 +9.9110 +9.9110 +9.9107 +9.9109

```

```

=====
Date: Tue Dec 18 11:41:10 2018
Expected Input Volts: == Not Specified ==
Scan Counter: 22096
WorstMinChanVoltsHWM: 9.896164 (Ch07)
WorstMaxChanVoltsHWM: 9.918823 (Ch17)
=====

```

Chan	<----- (volts) ----->			Tolerance Exceeded Count
	Min	Max	Ave	
00	9.897156	9.915390	9.909996	-
01	9.897079	9.915390	9.909962	-
02	9.897461	9.915009	9.909962	-
03	9.896698	9.914932	9.910021	-
04	9.896851	9.915619	9.909992	-
05	9.896774	9.915237	9.909930	-
06	9.897156	9.915771	9.910019	-
07	9.896164	9.915619	9.909951	-
08	9.896469	9.915237	9.909985	-
09	9.897308	9.915390	9.909964	-
10	9.897766	9.915466	9.910029	-
11	9.897690	9.915314	9.910003	-
12	9.897537	9.915543	9.910007	-
13	9.897156	9.915009	9.909934	-
14	9.897461	9.915237	9.909973	-
15	9.897308	9.915237	9.909985	-
16	9.902573	9.917450	9.910002	-
17	9.901047	9.918823	9.910035	-
18	9.900436	9.916916	9.909989	-
19	9.900818	9.918137	9.910025	-
20	9.903488	9.912796	9.909951	-
21	9.903717	9.913101	9.909988	-
22	9.903336	9.913177	9.909998	-
23	9.904099	9.913177	9.910041	-
24	9.904022	9.912872	9.910026	-
25	9.903717	9.912949	9.910007	-
26	9.904099	9.912872	9.909977	-
27	9.903717	9.912872	9.910007	-
28	9.904251	9.912796	9.910045	-
29	9.903564	9.912643	9.910012	-
30	9.903946	9.912796	9.910046	-
31	9.903870	9.912949	9.910021	-
32	9.904022	9.912720	9.909972	-
33	9.904099	9.912872	9.910048	-
34	9.904099	9.913177	9.910039	-
35	9.904022	9.912720	9.910006	-
36	9.904175	9.912720	9.909986	-
37	9.904099	9.912796	9.909982	-
38	9.904175	9.912796	9.909995	-
39	9.904175	9.912796	9.910041	-
40	9.903870	9.912796	9.910027	-
41	9.903946	9.912567	9.909989	-
42	9.903717	9.912643	9.910013	-
43	9.904099	9.912643	9.910020	-
44	9.904022	9.912491	9.909994	-
45	9.904251	9.912491	9.909981	-
46	9.903564	9.912643	9.910036	-
47	9.903793	9.912567	9.909994	-

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

48  9.904251    9.912796    9.910003    -
49  9.903870    9.912491    9.910000    -
50  9.903946    9.912949    9.909972    -
51  9.904175    9.912872    9.910006    -
52  9.904175    9.912567    9.909981    -
53  9.904022    9.912720    9.909983    -
54  9.904099    9.912796    9.909982    -
55  9.904175    9.912567    9.910021    -
56  9.904327    9.912720    9.910000    -
57  9.903870    9.912720    9.910019    -
58  9.904099    9.912949    9.910024    -
59  9.904022    9.912567    9.909988    -
60  9.904175    9.912643    9.910037    -
61  9.904175    9.912643    9.910029    -
62  9.904251    9.912567    9.910030    -
63  9.903946    9.912720    9.909974    -
=====

```

3.2.2 lib/ccrtaiicc_adc_calibrate

This test is useful for performing, saving and restoring ADC calibration. In order to calibrate, the clocks must be active and running for all the selected channels.

```

Usage: ./ccrtaiicc_adc_calibrate [-A] [-b board] [-i inCalFile] [-o outCalFile]
      -A                (perform Auto Calibration)
      -b <board>       (board #, default = 0)
      -i <In Cal File> (input calibration file [input->board_reg])
      -o <Out Cal File> (output calibration file [board_reg->output])

```

```

e.g. ./ccrtaiicc_adc_calibrate                (Dump calibration information to
                                                stdout)
      ./ccrtaiicc_adc_calibrate -A -o Calfile (Perform Auto calibration and dump
                                                information to 'Calfile')
      ./ccrtaiicc_adc_calibrate -i Calfile    (Update board calibration with
                                                supplied 'Calfile')

```

```

Usage: ./ccrtaiicc_adc_calibrate [-A StartCh,EndCh] [-b board] [-i inCalFile]
                                   [-o outCalFile] [-R]
      -A <StartCh,EndCh> (perform Auto Calibration between start & end chans.
                           default = 0,63)
      -b <board>         (board #, default = 0)
      -i <In Cal File>  (input calibration file [input->board_reg])
      -o <Out Cal File> (output calibration file [board_reg->output])
      -R                 (reset calibration data)

```

```

e.g. ./ccrtaiicc_adc_calibrate                (Dump calibration information to
                                                stdout)
      ./ccrtaiicc_adc_calibrate -A -o Calfile (Perform Auto calibration and dump
                                                information to 'Calfile')
      ./ccrtaiicc_adc_calibrate -i Calfile    (Update board calibration with
                                                supplied 'Calfile')
      ./ccrtaiicc_adc_calibrate -A 16,31     (Perform Auto calibration on second
                                                ADC i.e. ch16..31)
      ./ccrtaiicc_adc_calibrate -A ,12      (Perform Auto calibration on first
                                                ADC i.e. ch0..12)
      ./ccrtaiicc_adc_calibrate -R          (Reset calibration data)

```

Example display:

```

./ccrtaiicc_adc_calibrate -A

Device Name      : /dev/ccrtaiicc0
Board Serial No: 687377 (0x000a7d11)

```

Start Channel : 0
End Channel : 63
Auto Calibration started...done. (0.826 seconds)

==> Dump to 'stdout'
#Date : Tue Dec 18 11:53:26 2018

#Chan	Negative	Offset	Positive
#####	=====	=====	=====
ch00:	1.000046280678361654281616211	-0.00022888183593750000	1.000050663948059082031250000
ch01:	0.999962084926664829254150391	0.000000000000000000000	0.999961888883262872695922852
ch02:	0.999896354041993618011474609	-0.00007629394531250000	0.999897750560194253921508789
ch03:	1.000039299018681049346923828	0.00007629394531250000	1.000032507348805665969848633
ch04:	1.000215303152799606323242188	-0.00015258789062500000	1.000227037351578474044799805
ch05:	1.000232398509979248046875000	0.000000000000000000000	1.000233963597565889358520508
ch06:	1.000174359418451786041259766	-0.00015258789062500000	1.000177062116563320159912109
ch07:	0.999990142881870269775390625	-0.00007629394531250000	0.999986256472766399383544922
ch08:	1.000093540642410516738891602	-0.00015258789062500000	1.000101460143923759460449219
ch09:	0.999904607888311147689819336	-0.00007629394531250000	0.999918435700237751007080078
ch10:	1.000021356157958507537841797	-0.00015258789062500000	1.000024581328034400939941406
ch11:	0.999910922721028327941894531	-0.00007629394531250000	0.999915065709501504898071289
ch12:	1.000108417589217424392700195	-0.00015258789062500000	1.000114621128886938095092773
ch13:	1.000071702525019645690917969	-0.00007629394531250000	1.000082310289144515991210938
ch14:	1.000154437962919473648071289	-0.00015258789062500000	1.000149235129356384277343750
ch15:	1.000113883987069129943847656	-0.00015258789062500000	1.00012423936277627944962891
ch16:	1.000155716668814420700073242	-0.00007629394531250000	1.000164224766194820404052734
ch17:	1.000027132220566272735595703	0.00015258789062500000	1.000018282793462276458740234
ch18:	0.999931633006781339645385742	-0.00007629394531250000	0.999930288176983594894409180
ch19:	1.000115089118480682373046875	0.00015258789062500000	1.000108243897557258605957031
ch20:	1.000110489781945943832397461	-0.00015258789062500000	1.000121916644275188446044922
ch21:	1.000040223356336355209350586	0.00015258789062500000	1.000032895710319280624389648
ch22:	1.000062614679336547851562500	-0.00015258789062500000	1.000066666398197412490844727
ch23:	1.000183795113116502761840820	0.000000000000000000000	1.000178982503712177276611328
ch24:	0.999957418534904718399047852	0.000000000000000000000	0.999956104904413223266601562
ch25:	0.999945123679935932159423828	-0.00007629394531250000	0.999955588020384311676025391
ch26:	0.999991035554558038711547852	0.000000000000000000000	0.999997129198163747787475586
ch27:	1.000172080937772989273071289	0.00007629394531250000	1.000174157321453094482421875
ch28:	1.000162992626428604125976562	-0.00007629394531250000	1.000165878795087337493896484
ch29:	1.000091203488409519195556641	0.00007629394531250000	1.000087699387222528457641602
ch30:	1.000282614957541227340698242	-0.00007629394531250000	1.000285412650555372238159180
ch31:	1.000113802030682563781738281	-0.00007629394531250000	1.000128523912280797958374023
ch32:	1.000043609645217657089233398	-0.00007629394531250000	1.000059983066320419311523438
ch33:	0.999953246209770441055297852	0.00007629394531250000	0.999960092362016439437866211
ch34:	1.000127836130559444427490234	0.000000000000000000000	1.000133277848362922668457031
ch35:	0.999941903166472911834716797	0.00007629394531250000	0.999943493865430355072021484
ch36:	0.999991669319570064544677734	0.00007629394531250000	0.999989827629178762435913086
ch37:	1.000043198000639677047729492	0.00007629394531250000	1.000050891656428575515747070
ch38:	0.999993903562426567077636719	-0.00007629394531250000	1.000003706663846969604492188
ch39:	0.999979448039084672927856445	0.00007629394531250000	0.999988916795700788497924805
ch40:	1.000092174392193555831909180	0.00007629394531250000	1.000099983066320419311523438
ch41:	1.000059309415519237518310547	0.00007629394531250000	1.000068699475377798080444336
ch42:	1.000167491380125284194946289	-0.00007629394531250000	1.000180571340024471282958984
ch43:	1.000253048259764909744262695	0.00007629394531250000	1.000258826185017824172973633
ch44:	1.000121331308037042617797852	-0.00007629394531250000	1.000130682252347469329833984
ch45:	1.000186069402843713760375977	0.00007629394531250000	1.000193846412003040313720703
ch46:	1.000097106676548719406127930	-0.00015258789062500000	1.000103554688394069671630859
ch47:	1.000041996128857135772705078	-0.00007629394531250000	1.000057924538850784301757812
ch48:	0.999975174665451049804687500	0.00007629394531250000	0.999979007523506879806518555
ch49:	1.000073832459747791290283203	0.00015258789062500000	1.000083983875811100006103516
ch50:	1.000060985330492258071899414	-0.00007629394531250000	1.000069220084697008132934570
ch51:	1.000034462660551071166992188	0.00007629394531250000	1.000042255502194166183471680
ch52:	1.000020204577594995498657227	-0.00015258789062500000	1.000030491501092910766601562
ch53:	1.000030837021768093109130859	0.000000000000000000000	1.000043619889765977859497070
ch54:	0.999996385071426630020141602	0.000000000000000000000	1.000005447771400213241577148
ch55:	0.999992591794580221176147461	0.00007629394531250000	0.999998908489942550659179688
ch56:	1.000176913104951381683349609	0.00007629394531250000	1.00017974947852373123168945
ch57:	1.000093434471637010574340820	0.00015258789062500000	1.000098186079412698745727539
ch58:	1.000042880419641733169555664	-0.00007629394531250000	1.000060809310525655746459961
ch59:	1.000205467455089092254638672	0.00015258789062500000	1.000202169176191091537475586
ch60:	1.000177376437932252883911133	-0.00007629394531250000	1.000182859599590301513671875

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

ch61: 1.000130820553749799728393555 -0.00007629394531250000 1.000148957595229148864746094
ch62: 1.000131698790937662124633789 -0.00007629394531250000 1.000147680286318063735961914
ch63: 1.000189523678272962570190430 -0.00007629394531250000 1.000199154019355773925781250

```

3.2.3 lib/ccrtaiicc_adc_fifo

This test performs validation of the Analog Input ADC FIFO operation of the card.

```

Usage: ./ccrtaiicc_adc_fifo [-A] [-b BoardNo] [-c StartChan,EndChan]
                                [-C AdcUpdateClock] [-d Delay] [-D DMAEngine]
                                [-E ExpInpVolt] [-f DataFormat] [-F DebugFile] [-i]
                                [-j SpeedSelect] [-l LoopCnt] [-m XferMode] [-N]
                                [-p ClockTolerance] [-P FromChan,ToChan]
                                [-s InputSignal] [-S NumberOfSamples] [-T TestBus]
                                [-V MaxBoardVolts] [-w RawDataFile] [-X ExtClock]

-A                               (Perform Auto Calibration first using reference voltage)
-b BoardNo                       (Board number -- default is 0)
-c StartChan,EndChan             (Select start and end channel numbers -- default 0,63)
    -c 7,16                       (select channels 7 through 16 for processing)
    -c 32                          (select channels 32 through 63 for processing)
-C AdcUpdateClock                (select ADC update clock, a,b,c,d,e,A,B,C,D,E or 'n|N')
    -C a                            (Ch0..63=Normal Clock 0 set to MAX SPS 500000)
    -C b@20000.0/n                 (Ch0..15=Normal Clock 1 at 20000 SPS, Ch16..63=No Clock)
    -C a,b,A,B                     (Ch0..15=Normal Clock 0, Ch16..31 Normal Clock 1,
    -C c,d@350000,e                (Ch0..15=Normal Clock 2 at MAX SPS 500000, Ch16..31
    Normal Clock 3 at 350000 SPS, Ch 32-63 Normal External
    Signal)
-d Delay                          (Delay between screen refresh -- default is 0 milli-
    seconds)
-D DMA Engine                    (DMA Engine number -- default = 0)
-E <ExpInpVolts>@<Tol>           (Expected Input Volts@Tolerance -- default Tol=0.005000)
    +@<Tol>                         (Positive Calibration Ref Volt@Tolerance)
    -@<Tol>                         (Negative Calibration Ref Volt@Tolerance)
    s@<Tol>                         (Requires '-s' input signal option to specify voltage
    Volt@Tolerance)
    (valid '-s' arguments are 'g','+', '-','f','n','t')
-f DataFormat                    (select data format, '2' or 'b')
    -f b,2                          (Ch0..7=Offset binary, Ch8..15=Two's complement)
    -f 2/b                          (Ch0..7=Two's complement, Ch8..15=Offset binary)
    -f b                             (Ch0..15=Offset binary)
-F DebugFile                     (Debug file with rate display -- default "=== None ===")
    @DebugFile                      (Debug file without rate display -- default
    "=== None ===")
    ~DebugFile                     (For gnuplot, no header or summary -- default
    "=== None ===")
    @ or ~                          (No debug file and no rate display -- default
    "=== None ===")
-i                               (Enable Interrupts -- default = Disable)
-j SpeedSelect                   (select data speed, 'n' or 'h')
    -j n,h,n,h                      (Ch0..15 & Ch32..47=normal speed, Ch16..31 &
    Ch48..63=high speed)
    -j h/n                          (Ch0..15=normal speed, Ch16..63=high speed)
    -j h                             (Ch0..63=high speed)
-l LoopCnt                       (Loop count -- default is 0)
-m XferMode                      (Transfer Mode -- default = Library DMA)
    -mdP                            (Driver: (FIFO) PIO mode)
    -mlD                            (Library: (FIFO) DMA mode)
    -mlM                            (Library: (FIFO) Modular scatter-gather DMA mode)
    -mlP                            (Library: (FIFO) PIO mode)
-N                               (Open device with O_NONBLOCK flag for driver operations)
-p ClockTolerance                (select clock tolerance in parts/trillion. default is
    system default 0.0070)
-P FromChan,ToChan              (Pair FromChan to ToChan when Debug File specified --

```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

                                default is paired to itself)
-s InputSignal                  (select input signal, 'e', 'g', '+', '-', 'f', '-n',
                                't')
-s g                            (All channels set to ground calibration)
-s e                            (All channels set to external input)
-s +                            (All channels set to positive 9.91V reference
                                calibration)
-s -                            (All channels set to negative 9.91V reference
                                calibration)
-s f                            (All channels set to positive 5V reference calibration)
-s n                            (All channels set to negative 5V reference calibration)
-s t                            (All channels set to positive 2V reference calibration)
-S NumberOfSamples             (Number of Samples -- default is 49152)
-T TestBus                      (Test Bus Control 'b' or 'o'. Exit after programming
                                this option)
-T b                            (Calibration Bus Control)
-T o                            (Open Bus Control)
-V MaxBoardVolts               (Voltage range 'b5', 'u5', 'b10' or 'u10')
-V b5,b10                      (Ch0..15=+/-5V, Ch16..63=+/-10V)
-V u5/u10/b5,b10              (Ch0..15=+5V, Ch16..31=+10V, Ch32..47=+/-5V,
                                Ch48..63=+/-10V)
-V b10                          (Ch0..63=+/-10V)
-w RawDataFile                 (Only raw sample data written to file)
-X[0..3,e|E,n|N]              (Board External Clock Output Selection)
                                '0..3' - Clock Generator Output Number
                                'e|E' - External clock input (redrive)
                                'n|N' - No Clock

```

```

e.g. ./ccrtaicc_adc_fifo -Ca@100000,b@12345 -s+ (ADC0 is 100000Hz, ADC1, ADC2
                                                and ADC3 is 12345Hz, all
                                                Positive 9.91v Cal.)
./ccrtaicc_adc_fifo -Ca,B -P16,15 -c15,16 -F~Outfile -l100 -S50000
                                                (ADC0=500000Hz, ADC1=Inverted
                                                500000Hz, Channel 16 merged
                                                into channel 15, Samples in
                                                Outfile for gnuplot)
./ccrtaicc_adc_fifo -A -Ca@50000 -sf -Es (Autocal, All channels 50000Hz,
                                                all Positive 5V Cal, Expected
                                                5V)

```

Example display:

```
./ccrtaicc_adc_fifo -Ca@100000,b@12345 -s+ -I500
```

```

local_ptr=0x7ffff7fd7000
Number of Samples =49152
Transfer Mode      =Library DMA Mode
Physical Memory Information:
  UserPID          =11205
  PhysMemPtr       =0x5900000
  DriverVirtMemPtr=0xffff880005900000
  MmappedUserMemPtr=0x7ffff7f40000
  PhysMemSize      =0x00080000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl    =0
  NumOfEntriesUsed=1
Channels: Total=64, First=0, Last=63, Adc0Chans=16, Adc1Chans=16
Adc2Chans=16, Adc3Chans=16
500: usec=11097.15 (min=11095.75/max=11338.13/ave=11098.27) 17.72 MBytes/Sec
- EmptyCnt=104752 (80%)
=====
Date: Tue Dec 18 12:01:47 2018
Expected Input Volts: === Not Specified ===
Scan Counter: ADC0=1120881 ADC1=138373 ADC2=138373 ADC3=138373

```

Approx. Sample/Second: ADC0=100000 ADC1=12345 ADC2=12345 ADC3=12345
 NumberOfChans: ADC0=16 ADC1=16 ADC2=16 ADC3=16
 Channel Pairing: *** No pairing of any channels ***
 WorstMinChanVoltsHWM: 9.893799 (Ch10)
 WorstMaxChanVoltsHWM: 9.918137 (Ch16)

```

=====
<----- (volts) ----->
Chan  Min          Max          Ave          DetectedCnt  TolerExeededCnt
=====
00    9.894104       9.917755       9.909340     1120881      -
01    9.894638       9.916077       9.909494     1120881      -
02    9.894562       9.916153       9.909365     1120881      -
03    9.894714       9.916153       9.909381     1120881      -
04    9.894180       9.916000       9.909418     1120881      -
05    9.894409       9.916153       9.909427     1120881      -
06    9.894028       9.916382       9.909353     1120881      -
07    9.894028       9.916229       9.909367     1120881      -
08    9.893951       9.916382       9.909438     1120881      -
09    9.893875       9.916077       9.909417     1120881      -
10    9.893799       9.916306       9.909445     1120881      -
11    9.894104       9.916077       9.909371     1120881      -
12    9.893875       9.915695       9.909331     1120881      -
13    9.894409       9.915924       9.909367     1120881      -
14    9.894028       9.916000       9.909462     1120881      -
15    9.894714       9.916306       9.909493     1120881      -
16    9.900208       9.918137       9.909503     138373       -
17    9.900589       9.917450       9.909189     138373       -
18    9.900818       9.916458       9.909458     138373       -
19    9.899826       9.917374       9.909563     138373       -
20    9.903336       9.912491       9.909420     138373       -
21    9.903488       9.912415       9.909445     138373       -
22    9.903641       9.912491       9.909524     138373       -
23    9.903488       9.912567       9.909492     138373       -
24    9.903488       9.912796       9.909467     138373       -
25    9.903564       9.912415       9.909445     138373       -
26    9.903488       9.912338       9.909455     138373       -
27    9.903717       9.912338       9.909478     138373       -
28    9.903793       9.912567       9.909474     138373       -
29    9.903641       9.912262       9.909515     138373       -
30    9.903564       9.912491       9.909588     138373       -
31    9.903717       9.912415       9.909560     138373       -
32    9.904022       9.912491       9.909567     138373       -
33    9.903870       9.912491       9.909564     138373       -
34    9.903183       9.912643       9.909612     138373       -
35    9.903641       9.912262       9.909574     138373       -
36    9.903793       9.912872       9.909592     138373       -
37    9.903946       9.912415       9.909593     138373       -
38    9.903488       9.912720       9.909595     138373       -
39    9.903564       9.912186       9.909568     138373       -
40    9.903793       9.912491       9.909604     138373       -
41    9.903946       9.912567       9.909564     138373       -
42    9.903793       9.912567       9.909644     138373       -
43    9.903870       9.912415       9.909654     138373       -
44    9.903946       9.912491       9.909677     138373       -
45    9.904175       9.912720       9.909755     138373       -
46    9.903564       9.912415       9.909610     138373       -
47    9.904022       9.912491       9.909646     138373       -
48    9.903946       9.912567       9.909775     138373       -
49    9.904022       9.912491       9.909737     138373       -
50    9.903870       9.912872       9.909743     138373       -
51    9.903870       9.912491       9.909795     138373       -
52    9.904251       9.912415       9.909664     138373       -
53    9.904175       9.912491       9.909677     138373       -
=====

```

54	9.903946	9.912643	9.909834	138373	-
55	9.904175	9.912796	9.909796	138373	-
56	9.904099	9.912720	9.909860	138373	-
57	9.904099	9.912567	9.909823	138373	-
58	9.904022	9.912567	9.909796	138373	-
59	9.904022	9.912491	9.909837	138373	-
60	9.904022	9.912720	9.909734	138373	-
61	9.904099	9.912491	9.909811	138373	-
62	9.904099	9.912643	9.909863	138373	-
63	9.904327	9.912643	9.909812	138373	-

=====

3.2.4 lib/ccrtaiicc_adc_sps

This is a useful tool to display the sample rate of various channels.

```
Usage: ./ccrtaiicc_adc_sps [-b Board] [-c StartChan,StopChan] [-C AdcUpdateClock]
      [-E ExpSPS@Tol] [-j SpeedSelect] [-l LoopCnt]
      [-m XferMode] [-S NumSamples] [-T SglBrdClkLpbkTst]
      [-X ExtClock]

-b Board                (Board number -- default is 0)
-c StartChan,EndChan   (Select start and end channel numbers -- default 0,63
  -c 7,16                (select channels 7 through 16 for processing
  -c 32                  (select channels 32 through 63 for processing)
-C AdcUpdateClock      (select ADC update clock, a,b,c,d,e,A,B,C,D,E or 'n|N')
  -C a                  (Ch0..63=Normal Clock 0 set to MAX SPS 500000)
  -C b@20000.0/n        (Ch0..15=Normal Clock 1 at 20000 SPS, Ch16..63=No Clock)
  -C a,b,A,B            (Ch0..15=Normal Clock 0, Ch16..31 Normal Clock 1,
                        Ch32..47 Inverted Clock 0, Ch48..63 Inverted Clock 1)
  -C c,d@350000,e      (Ch0..15=Normal Clock 2 at MAX SPS 500000, Ch16..31
                        Normal Clock 3 at 350000 SPS, Ch 32-63 Normal External
                        Signal)
-E ExpSPS@Tol          (specify expected samples/second and tolerance for each
                        ADC)
  -E C                  (All ADC's to use clock samples/second and default
                        tolerance 0.010%)
  -E c@0.02,30000      (ADC 0 uses clock samples/second and tolerance 0.02%,
                        remaining use 30,000 SPS and default tolerance 0.010%)
  -E C@0.02,C          (All ADC's to use clock samples/second and default
                        tolerance except for ADC 0 tolerance of 0.02%)
  -E 10000,c           (ADC 0 to use 10000 SPS, rest of ADCs to use clock
                        samples/second. Default tolerance for all ADCs)
-F DebugFile           (Debug file with menu display -- default "=== None ===")
  @DebugFile           (Debug file without menu display (only summary and rate
                        display) -- default "=== None ===")
  @                     (No debug file and no menu display (only summary and
                        rate display) -- default "=== None ===")
-j SpeedSelect          (select data speed, 'n' or 'h')
  -j n,h,n,h           (Ch0..15 & Ch32..47=normal speed, Ch16..31 &
                        Ch48..63=high speed)
  -j h/n               (Ch0..15=normal speed, Ch16..63=high speed)
  -j h                 (Ch0..63=high speed)
-l LoopCnt             (Loop Count -- default is 10000000)
-l 0                   (Loop forever)
-m XferMode            (Transfer Mode -- default = 'DMA Channel' for old
                        firmware, 'MsgDma' for new firmware)
  -md0                 (Library: DMA 0 mode)
  -md1                 (Library: DMA 1 mode)
  -mm                  (Library: Modular Scatter-Gather DMA mode)
-S NumSamples          (Number of Samples/Read -- default is 4096 for DMA and
                        2000 for MsgDma)
-T                     (Single Board Clock Loopback Test - External Clock
                        In/Out requires loopback cable)
```

```
-X[0..3,e|E,n|N]          (Board External Clock Output Selection)
                          '0..3' - Clock Generator Output Number
                          'e|E' - External clock input (redrive)
                          'n|N' - No Clock
```

e.g. ./ccrtaicc_adc_sps -Ca@123456,n,c@78912,n (ADC0 is 123456Hz, ADC1 is off, ADC2 is 78912Hz and ADC3 is off)

Example display:

```
./ccrtaicc_adc_sps -Ca@123456,n,c@78912,n
```

```
local_ptr=0x7ffff7fd7000
Physical Memory Information:
  UserPID          =21205
  PhysMemPtr       =0x615a0000
  DriverVirtMemPtr=0xfffff8800615a0000
  MmappedUserMemPtr=0x7ffff7fcb000
  PhysMemSize      =0x00002000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl     =0
  NumOfEntriesUsed=1

Board Number      [-b]: 0
Channel Selection [-c]: Start=Ch0, End=Ch63 (Number of Active Channels=64)
                  : Active Channel Mask=0xffffffffffffffff
Update Clock Selected [-C]: Ch00..15 [00]Normal Clock 0 (123456.000 SPS)
                  : Ch16..31 [15]### CLOCK IS NOT CONNECTED ###
                  : Ch32..47 [02]Normal Clock 2 (78912.000 SPS)
                  : Ch48..63 [15]### CLOCK IS NOT CONNECTED ###
Speed Select      [-j]: Ch00..15 [0]Normal Speed
                  : Ch16..31 [0]Normal Speed
                  : Ch32..47 [0]Normal Speed
                  : Ch48..63 [0]Normal Speed
Loop Count        [-l]: 10000000
Transfer Mode     [-m]: Library: (FIFO) MODULAR SCATTER-GATHER DMA I/O
Number of Samples [-S]: 2000 (8000 bytes)
Clock Loopback Test [-T]: No
External Clock Output [-X]: 7 (No Clock)
Input Signal      : [0]External Signal
Voltage Range     : Ch00..15 [1]+/-10 Volts
                  : Ch16..31 [1]+/-10 Volts
                  : Ch32..47 [1]+/-10 Volts
                  : Ch48..63 [1]+/-10 Volts
Data Format        : Ch00..15 [0]Offset binary
                  : Ch16..31 [0]Offset binary
                  : Ch32..47 [0]Offset binary
                  : Ch48..63 [0]Offset binary

Read: Size 8000, Count 24 (FIFO wait: 602.7us, Read time/rate: 25.3us/316.5MBPS)
===== Samples/Second =====
      [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]
=====
[00 07] 123459 123459 123459 123459 123459 123459 123459 123459
[08 15] 123459 123459 123459 123459 123459 123459 123459 123459
[16 23] 000000 000000 000000 000000 000000 000000 000000 000000
[24 31] 000000 000000 000000 000000 000000 000000 000000 000000
[32 39] 078915 078915 078915 078915 078915 078915 078915 078915
[40 47] 078915 078915 078915 078915 078915 078915 078915 078915
[48 55] 000000 000000 000000 000000 000000 000000 000000 000000
[56 63] 000000 000000 000000 000000 000000 000000 000000 000000

==== No overflow occurred (HWM Samples In fifo 2368) ====

=====
      <-- (Samples/Second) -->
Chan  Min      Max      Ave
=====
      0      123450 123461 123456
```



```
1 123450 123461 123456
2 123450 123461 123456
3 123450 123461 123456
4 123450 123461 123456
5 123450 123461 123456
6 123450 123461 123456
7 123450 123461 123456
8 123450 123461 123456
9 123450 123461 123456
10 123450 123461 123456
11 123450 123461 123456
12 123450 123461 123456
13 123450 123461 123456
14 123450 123461 123456
15 123450 123461 123456
16 000000 000000 000000
17 000000 000000 000000
18 000000 000000 000000
19 000000 000000 000000
20 000000 000000 000000
21 000000 000000 000000
22 000000 000000 000000
23 000000 000000 000000
24 000000 000000 000000
25 000000 000000 000000
26 000000 000000 000000
27 000000 000000 000000
28 000000 000000 000000
29 000000 000000 000000
30 000000 000000 000000
31 000000 000000 000000
32 078909 078915 078912
33 078909 078915 078912
34 078909 078915 078912
35 078909 078915 078912
36 078909 078915 078912
37 078909 078915 078912
38 078909 078915 078912
39 078909 078915 078912
40 078909 078915 078912
41 078909 078915 078912
42 078909 078915 078912
43 078909 078915 078912
44 078909 078915 078912
45 078909 078915 078912
46 078909 078915 078912
47 078909 078915 078912
48 000000 000000 000000
49 000000 000000 000000
50 000000 000000 000000
51 000000 000000 000000
52 000000 000000 000000
53 000000 000000 000000
54 000000 000000 000000
55 000000 000000 000000
56 000000 000000 000000
57 000000 000000 000000
58 000000 000000 000000
59 000000 000000 000000
60 000000 000000 000000
61 000000 000000 000000
62 000000 000000 000000
63 000000 000000 000000
```

```
./ccrtaiicc_adc_sps -Ca@123456 -T // An external clock loopback cable needs to be connected
```

```
local_ptr=0x7ffff7fd7000
Physical Memory Information:
UserPID =21353
PhysMemPtr =0x3fd90000
DriverVirtMemPtr=0xffff88003fd90000
```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

MmappedUserMemPtr=0x7ffff7fcb000
PhysMemSize      =0x00002000
PhysMemSizeFreed=0x00000000
EntryInTxTbl    =0
NumOfEntriesUsed=1

```

```

Board Number      [-b]: 0
Channel Selection [-c]: Start=Ch0, End=Ch63 (Number of Active Channels=64)
                   : Active Channel Mask=0xfffffffffffffff
Update Clock Selected [-C]: Ch00..15 [08]Normal External Signal
                   : Ch16..31 [08]Normal External Signal
                   : Ch32..47 [08]Normal External Signal
                   : Ch48..63 [08]Normal External Signal
Speed Select      [-j]: Ch00..15 [0]Normal Speed
                   : Ch16..31 [0]Normal Speed
                   : Ch32..47 [0]Normal Speed
                   : Ch48..63 [0]Normal Speed
Loop Count        [-l]: 10000000
Transfer Mode     [-m]: Library: (FIFO) MODULAR SCATTER-GATHER DMA I/O
Number of Samples [-S]: 2000 (8000 bytes)
Clock Loopback Test [-T]: Yes (Loopback cable required)
External Clock Output [-X]: 0 (Clock Generator Output 0)
Input Signal      : [0]External Signal
Voltage Range     : Ch00..15 [1]+/-10 Volts
                   : Ch16..31 [1]+/-10 Volts
                   : Ch32..47 [1]+/-10 Volts
                   : Ch48..63 [1]+/-10 Volts
Data Format       : Ch00..15 [0]Offset binary
                   : Ch16..31 [0]Offset binary
                   : Ch32..47 [0]Offset binary
                   : Ch48..63 [0]Offset binary

```

Read: Size 8000, Count 51 (FIFO wait: 184.2us, Read time/rate: 25.3us/316.1MBPS)

```

===== Samples/Second =====
          [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]
          =====
[00 07] 123459 123459 123459 123459 123459 123459 123459 123459
[08 15] 123459 123459 123459 123459 123459 123459 123459 123459
[16 23] 123460 123460 123460 123460 123460 123460 123460 123460
[24 31] 123460 123460 123460 123460 123460 123460 123460 123460
[32 39] 123460 123460 123460 123460 123460 123460 123460 123460
[40 47] 123460 123460 123460 123460 123460 123460 123460 123460
[48 55] 123460 123460 123460 123460 123460 123460 123460 123460
[56 63] 123460 123460 123460 123460 123460 123460 123460 123460

```

==== No overflow occurred (HWM Samples In fifo 3072) ====

```

=====
<-- (Samples/Second) -->
Chan  Min      Max      Ave
=====
  0    123450  123463  123456
  1    123450  123463  123456
  2    123450  123463  123456
  3    123450  123463  123456
  4    123450  123463  123456
  5    123450  123463  123456
  6    123450  123463  123456
  7    123450  123463  123456
  8    123450  123463  123456
  9    123450  123463  123456
 10    123450  123463  123456
 11    123450  123463  123456
 12    123450  123463  123456
 13    123450  123463  123456
 14    123450  123463  123456
 15    123450  123463  123456
 16    123449  123464  123456
 17    123449  123464  123456
 18    123449  123464  123456
 19    123449  123464  123456

```

```

20 123449 123464 123456
21 123449 123464 123456
22 123449 123464 123456
23 123449 123464 123456
24 123449 123464 123456
25 123449 123464 123456
26 123449 123464 123456
27 123449 123464 123456
28 123449 123464 123456
29 123449 123464 123456
30 123449 123464 123456
31 123449 123464 123456
32 123449 123464 123456
33 123449 123464 123456
34 123449 123464 123456
35 123449 123464 123456
36 123449 123464 123456
37 123449 123464 123456
38 123449 123464 123456
39 123449 123464 123456
40 123449 123464 123456
41 123449 123464 123456
42 123449 123464 123456
43 123449 123464 123456
44 123449 123464 123456
45 123449 123464 123456
46 123449 123464 123456
47 123449 123464 123456
48 123449 123464 123456
49 123449 123464 123456
50 123449 123464 123456
51 123449 123464 123456
52 123449 123464 123456
53 123449 123464 123456
54 123449 123464 123456
55 123449 123464 123456
56 123449 123464 123456
57 123449 123464 123456
58 123449 123464 123456
59 123449 123464 123456
60 123449 123464 123456
61 123449 123464 123456
62 123449 123464 123456
63 123449 123464 123456

```

3.2.5 lib/ccrtaicc_check_bus

This is a simple test to check whether there is interference from other cards that may be sharing the same bus. It simply computes the time it takes to perform hardware reads and computes the jitter. It must be run as *root*.

```

Usage: ./ccrtaicc_check_bus [-b Board] [-c CPU] [-l LoopCnt] [-t Tolerance]
-b Board      (Board number -- default is 0)
-c CPU        (CPU number -- default is 1)
-l LoopCnt    (Loop Count -- default is 10000000)
-l 0          (Loop forever)
-t Tolernace  (Tolerance -- default is 2.00 micro-seconds)

```

Example display:

```
sudo ./ccrtaicc_check_bus
```

```

local_ptr=0x7ffff7fd7000
10000000: usec/read: Cur=1.167 (Min=1.092 Max=1.768 Ave=1.143276)
          [Bus Jitter (usec): 0.676 ==> LOW]

```

3.2.6 lib/ccrtaiicc_clock

This is a useful tool to display information of the various clocks and also program them.

```
Usage: ./ccrtaiicc_clock [-b BoardNo] [-C UpdateClock] [-d Delay] [-l LoopCnt]
[-R]
-b BoardNo          (Board number -- default is 0)
-C <Clock>@<Frequency> (set update clock '0..6' with frequency )
-d Delay            (Delay between screen refresh -- default is 10 milli-
seconds)
-l LoopCnt          (Loop count -- default is 0)
-R                  (Reset/Clear all clocks)

e.g. ./ccrtaiicc_clock -C 1@300000
      (Set Clock 1 to 300000 SPS - do not change any other
      running clocks)
      ./ccrtaiicc_clock -R -C0@100000 -C4@12345
      (Reset all clocks and then set Clock 0 to 100000 SPS and
      Clock 4 to 12345 SPS)
```

Example display:

```
./ccrtaiicc_clock -R -C0@100000 -C4@12345

Board Number [-b]: 0
Delay [-d]: 10 milli-seconds
Loop Count [-l]: ***Forever***
Scan Count: 693

_____ Clock Revision _____
Silicon Revision: A1
Base Part Number: 5341
Device Speed Grade: A
Device Revision: A

_____ Clock CSR _____
Clock Interface: Idle
Clock Output: Enabled
Clock State: Active

_____ Input Clock Status _____
Calibration: Not In-Progress
SMBUS Timeout: Not Timed Out
PLL Lock: Locked
Input Signal: Present
Input_0 Clock: Present
Input_1 Clock: *** Not Present ***
Input_2 Clock: *** Not Present ***
Input_FB Clock: Present
XAXB Input Clock: *** Not Present ***

_____ Output Clock Setting _____
User output clock frequency 0: 100000.000 Samples/Second/Channel
User output clock frequency 1: *** Not Set ***
User output clock frequency 2: *** Not Set ***
User output clock frequency 3: *** Not Set ***
User output clock frequency 4: 12345.000 Samples/Second/Channel
User output clock frequency 5: *** Not Set ***
User output clock frequency 6: *** Not Set ***
SD-RAM output clock frequency 7: 10000000.000 Samples/Second/Channel
External output clock frequency 8: 10000000.000 Samples/Second/Channel
Feed-Back output clock frequency 9: 10000000.000 Samples/Second/Channel
```

3.2.7 lib/ccrtaiicc_disp

Useful program to display the local board registers. This program uses the *curses* library. This test is similar to the previous non-library test.

```
Usage: ./ccrtaiicc_disp [-b Board] [-d Delay] [-D DMAEngineNo] [-H] [-i]
      [-l LoopCnt] [-m XferMode] [-o Offset] [-P Pause]
      [-s XferSize] [-S DispSize]
-b Board      (Board number -- default board is 0)
-d Delay      (Delay between screen refresh -- default is 0)
-D DMAEngineNo (DMA Engine number -- default = 0)
-H           (Enable Hyper-Drive Mode -- default "=== Disabled ===")
-i           (Enable Interrupts -- default = Disable)
-l LoopCnt   (Loop Count - default = 0)
-m XferMode  (Transfer Mode -- default = DMA)
  -md       (Avalon Memory: DMA mode)
  -mm       (Avalon Memory: Modular Scatter-Gather DMA mode)
  -mp       (Avalon Memory: Programmed I/O mode)
-o Offset    (Hex offset to read from -- default is 0x0)
-P Pause     (Microseconds to sleep in User Function loop -- default is 0)
-s XferSize  (Number of bytes to transfer -- default is 0x1000)
-S DispSize  (Number of bytes to display -- default is 0x200)
```

Example display:

```
./ccrtaiicc_disp

local_ptr=0x7ffff7fd7000
Physical Memory Information:
  UserPID      =16618
  PhysMemPtr   =0x7c2be000
  DriverVirtMemPtr=0xffff88007c2be000
  MmappedUserMemPtr=0x7ffff7fcc000
  PhysMemSize  =0x00001000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl =0
  NumOfEntriesUsed=1
-----
Board Number      [-b]: 0
Delay             [-d]: 0 milli-seconds
DMA Engine        [-D]: 0
Hyper-Drive       [-H]: Disabled
Interrupts        [-i]: Disabled
Loop Count        [-l]: ***Forever***
Transfer Mode     [-m]: Basic DMA I/O (Avalon Memory)
Offset            [-o]: 0x00000000
Transfer Size     [-s]: 0x00001000 (4096) bytes ( 17.308 MBytes/Second)
Display Size      [-S]: 0x00000200 (512) bytes

ScanCount         : 13228
Read Duration (microsecs) : 236.660 (min= 234.615/max= 247.757/ave= 236.132)

      00      04      08      0C      10      14      18      1C
=====
000000  93500101 12052018 00010000 00000000 00000000 00000000 00000000 00000000
000020  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000040  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000060  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000080  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000a0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000c0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000e0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000100  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

000120  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000140  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000160  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000180  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001a0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001c0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001e0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

```

3.2.8 lib/ccrtaicc_dma

This test transfers data from physical memory to the Local register area and back. There are three modes of operation. One is regular DMA, the second is Modular Scatter-Gather DMA and the third is programmed I/O. Depending on the number of DMA engines supported by the card, the user can select one of them to perform the DMA. Additionally, if the card supports Modular Scatter-Gather DMA, then they can also select that. Area select is one of three areas the user can specify. They represent the area in physical memory and local register where the transfer is to occur. The test automatically switches to a different area corresponding to the regular DMA engine supplied. If multiple copies of this application is run on the same card using the same DMA engine, then the user needs to manually select a different area '-A' so the data mismatch does not occur due to using the same area region.

```

Usage: ./ccrtaicc_dma [-A Area2Select] [-b Board] [-D DMAEngineNo] [-i]
        [-l LoopCnt] [-m XferMode] [-s Size] [-v VerboseNo]
-A Area2Select (Area to select -- default = -1)
-b Board      (Board number -- default = 0)
-D DMAEngineNo (DMA Engine number -- default = 0)
-i           (Enable Interrupts -- default = Disable)
-l LoopCnt   (Loop Count - default = 1000)
-m XferMode  (Transfer Mode -- default = DMA)
  -md       (DMA mode)
  -mm      (MsgDma mode - '-D' option is ignored)
  -mp      (Programmed I/O mode)
-s Size      (Transfer Size in bytes (multiple of byte width) -
             default = 12288)
-V VerboseNo (verbose -- default = 0)

```

```

e.g. ./ccrtaicc_dma -A1      (perform dma using DMA0 on area 1 )
     ./ccrtaicc_dma -i -D1  (perform dma using DMA1 with interrupts on area 0)
     ./ccrtaicc_dma -mm     (perform dma using MsgDMA on area 0)

```

Example display:

```
./ccrtaicc_dma
```

```

Device Name: /dev/ccrtaicc0
local_ptr=0x7ffff7fd7000
      Physical Memory Information:
      UserPID           =16945
      PhysMemPtr        =0x5a00000
      DriverVirtMemPtr=0xffff880005a00000
      MmappedUserMemPtr=0x7ffff70e8000
      PhysMemSize       =0x00200000
      PhysMemSizeFreed=0x00000000
      EntryInTxTbl     =0
      NumOfEntriesUsed=2
### Avalon Address[A0]: 0x00001000 - 0x00004000
###   DMA Address[A0]: 0x00100400 - 0x00103400
###   Transfer Size: 12288 (0x00003000) bytes (DMA without Interrupts:
                        DMA Engine 0) ###
1000: A2P: Total: 697.244us ( 17.62 MB/s): first=0xface0000 last=0xface0bff
      (micro-seconds)          (MBytes/second)
      Min      Max      Ave      Min      Max      Ave

```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

-----
P2A:  427.70  433.56  429.61      28.34  28.73  28.60
A2P:  696.25  722.41  697.19      17.01  17.65  17.63

```

3.2.9 lib/ccrtaiicc_example

This test provides a simple example of programming ADC.

```

Usage: ./ccrtaiicc_example [-b Board]
       -b Board      (Board number -- default is 0)

```

Example display:

```

./ccrtaiicc_example

local_ptr=0x7ffff7fd7000
Physical Memory Information:
  UserPID      =17871
  PhysMemPtr   =0x7c291000
  DriverVirtMemPtr=0xffff88007c291000
  MmappedUserMemPtr=0x7ffff7fcc000
  PhysMemSize  =0x00001000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl =0
  NumOfEntriesUsed=1
### Configuring ADC ###
- Activate ADC
- Configure ADC
- Set Input Control to Calibration Bus
- Set Calibration to Positive Reference Voltage
### Programming Clocks ###
### Calibrate All ADC Channels ###
### Reading ADC Channels using ccrtAICC_Transfer_Data() ###

==== ADC Channels - Using ccrtAICC_Transfer_Data() ==== (length=64)
+DMP+      0  0001fb6b  0001fb6f  0001fb6e  0001fb6b  *...k...o...n...k*
+DMP+     0x10  0001fb68  0001fb6c  0001fb6d  0001fb69  *...h...l...m...i*
+DMP+     0x20  0001fb6d  0001fb6e  0001fb66  0001fb6a  *...m...n...f...j*
+DMP+     0x30  0001fb6a  0001fb6a  0001fb64  0001fb64  *...j...j...d...d*

### Reading ADC Channels using ccrtAICC_DMA_Configure()/ccrtAICC_DMA_Fire() ###
- Convert Physical DMA Memory Address to Avalon Equivalent Address
- Configure DMA
- Fire DMA

==== ADC Channels - Using ccrtAICC_DMA_Fire() ==== (length=64)
+DMP+      0  0001fb6c  0001fb66  0001fb66  0001fb6a  *...l...f...f...j*
+DMP+     0x10  0001fb69  0001fb67  0001fb6c  0001fb68  *...i...g...l...h*
+DMP+     0x20  0001fb65  0001fb6b  0001fb6b  0001fb6a  *...e...k...k...j*
+DMP+     0x30  0001fb70  0001fb6b  0001fb69  0001fb6a  *...p...k...i...j*
- Seizing MSGDMA
### Single (one descriptor) Modular Scatter-Gather DMA ###
- Configure Single MSGDMA (PCIe ==> Avalon)
- Fire Single MSGDMA: Xfer 0x8000 bytes: Pcie ==> Avalon (@0x8000)
- Validating data
- Configure Single MSGDMA (Avalon ==> PCIe)
- Fire Single MSGDMA: Xfer 0x8000 bytes: Avalon (@0x8000) ==> PCIe
- Validating data
- Releasing MSGDMA
### Multi (four descriptor) Modular Scatter-Gather DMA ###
- Allocating memory and seeding with pattern)
- Seizing MSGDMA
- Configure multi MSGDMA (PCIe ==> Avalon ==> PCIe ==> Avalon ==> PCIe)

```

- Setup Multi MSGDMA
- Fire Multi MSGDMA
- Validating data
- Releasing MSGDMA

3.2.10 lib/ccrtaicc_expires

This test is useful in displaying board expires information.

```
Usage: ./ccrtaicc_expires -[b Board] -[s]
       -b <board>          (board #, default = 0)
       -s                  (short display, default = verbose)
```

Example display:

./ccrtaicc_expires (for card that has no restrictions)

```
Device Name: /dev/ccrtaicc0
Board Serial No: 687377 (0x000a7d11)
```

```
#####
###                               ###
###          UNRESTRICTED FIRMWARE      ###
###                               ###
#####
```

./ccrtaicc_expires (for restricted card that has NO expiration date)

```
Device Name: /dev/ccrtaicc0
Board Serial No: 98765 (0x000181cd)
```

```
#####
###                               ###
###          RESTRICTED FIRMWARE        ###
###                               ###
#####
```

```
=====
=== No Expiration Date ===
=====
```

./ccrtaicc_expires (for restricted card that has expiration date)

```
Device Name: /dev/ccrtaicc0
Board Serial No: 98765 (0x000181cd)
```

```
#####
###                               ###
###          RESTRICTED FIRMWARE        ###
###                               ###
#####
```

```
=====
Local Expiration Date: 03/11/2018 13:21:52
GMT Expiration Date: 03/11/2018 17:21:52
Duration to Expire: Days=122, Hours=2, Minutes=49, Seconds=20
=====
```

./ccrtaicc_expires -s (for card that has no restrictions)

Unrestricted


```
./ccrtaicc_expires -s (for restricted card that has NO expiration date)
```

```
Restricted: No expiration date
```

```
./ccrtaicc_expires -s (for restricted card that has expiration date)
```

```
Restricted: Expire in 10550462 seconds
```

3.2.11 lib/ccrtaicc_identify

This test is useful in identifying a particular card by displaying its LED.

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

Example display:

```
./ccrtaicc_identify
```

```
Device Name      : /dev/ccrtaicc0
Board ID         : 9350
Board Type       : 01
Board Function   : 01
Board Serial No: 687377 (0x000a7d11)
```

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

3.2.12 lib/ccrtaicc_info

This test is useful in getting information for all the *ccrtaicc* devices in the system.

```
Usage: ./ccrtaicc_info -[b Board] -[l] -[v]
        -b <board>          (board #, default = 0)
        -l                  (long display, default = short)
        -v                  (long display and verbose, default = no verbose)
        -l -v               (long display and verbose, default = no verbose)
```

Example display:

```
./ccrtaicc_info
```

```
# IRQ MSI Bu:Sl:Fv VnID:Sub BdID:Ty:Fu:Sub FMaj.Min(mm:dd:yy hh:mm:ss) MC FmFlvCod FwbRev
Temp:C/F SerialNo RLS# Func
0 59 Y 03:00:00 1542:1542 9350.01.01:0100 1.0(12/05/18 00:00:00) C7 00000000 00000000
0/ 32.0 687377 150 AICC
```

```
./ccrtaicc_info -l
```

```
##### Board 0 #####
Version: 23.2.0
Build: Wed June 12 11:33:33 EST 2019
Module: ccrtaicc
Board Index: 0 (PCIe-CCRT_FPGA_AICC)
Bus: 0x03
Slot: 0x00
Func: 0x00
Vendor ID: 0x1542
```

```

Sub-Vendor ID: 0x1542
Board Info: 0x93500101 (id=9350, type=0x01, func=0x01 (AICC))
Member Code: 1 (C7)
Sub-Device ID: 0x0100
Firmware Date/Time: 0x04032019 0x00000000 (04/03/2019 00:00:00)
Firmware Revision: 0x00020000 (2.0)
Epgawb Revision: 0x00000000 (0000.00-00) (Not Supported)
Firmware Flavor Code: 0x00000000 (0) (****)
Board Serial Number: 0x000a7d11 (687377)
FPGA Chip Temperature: 0x00 (0 degree C, 32.0 degree F)
Run Level Sector Number: 0x96 (150)
Multi-Firmware Support: 0x1 (Yes)
MSI Support: Enabled
Scatter-Gather DMA Support: No
Double-Word Support: Yes
IRQ Level: 59
10V Calibration Reference: 9.91 Volts
5V Calibration Reference: 4.95 Volts

```

./ccrtaicc_info -l -v

```

##### Board 0 #####
Version: 23.2.0
Build: Wed June 12 11:33:33 EST 2019
Module: ccrtaicc
Board Index: 0 (PCIe-CCRT_FPGA_AICC)
Bus: 0x03
Slot: 0x00
Func: 0x00
Vendor ID: 0x1542
Sub-Vendor ID: 0x1542
Board Info: 0x93500101 (id=9350, type=0x01, func=0x01 (AICC))
Member Code: 1 (C7)
Sub-Device ID: 0x0100
Firmware Date/Time: 0x04032019 0x00000000 (04/03/2019 00:00:00)
Firmware Revision: 0x00020000 (2.0)
Epgawb Revision: 0x00000000 (0000.00-00) (Not Supported)
Firmware Flavor Code: 0x00000000 (0) (****)
Board Serial Number: 0x000a7d11 (687377)
FPGA Chip Temperature: 0x00 (0 degree C, 32.0 degree F)
Run Level Sector Number: 0x96 (150)
Multi-Firmware Support: 0x1 (Yes)
MSI Support: Enabled
Scatter-Gather DMA Support: No
Double-Word Support: Yes
IRQ Level: 59
10V Calibration Reference: 9.91 Volts
5V Calibration Reference: 4.95 Volts

```

```

---ADC Information---
Maximum Voltage Range: 10 Volts
Number of ADCs: 4
Number of ADC Channels: 64
Number of ADC Resolution: 18 Bits
All ADC Channels Mask: 0xfffffffffffffff
Maximum ADC Fifo Threshold: 0x00020000

```

```

---DMA Information---
Driver DMA Size: 524288
Num of Trans Tbl Entries: 8
Avalon Page Bits: 20
Avalon Page Size: 1048576
TX Interface Base: 8388608
DMA Maximum Engines: 2
DMA Maximum Burst Size: 1024
DMA Maximum Transactions: 32
DMA Maximum Size: 1048576 (DMA 0)
DMA Width in Bytes: 4 (DMA 0)
DMA Fire Command: 140 (DMA 0)
DMA Maximum Size: 32752 (DMA 1)
DMA Width in Bytes: 16 (DMA 1)

```

All information contained in this document is confidential and proprietary to Concurrent Real-Time. No part of this document may be reproduced, transmitted, in any form, without the prior written permission of Concurrent Real-Time. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document.

```

DMA Fire Command: 2184      (DMA 1)

---Analog/DMA Interrupt Information---
Interrupt Count: 2000
  DMA 0 Count: 0
  DMA 1 Count: 2000
MSG DMA Count: 0
Interrupts Occurred Mask: 0x00000002
Wakeup Interrupt Mask: 0x00000000
Timeout Seconds: 0
DMA Control: 0x00000000

---Memory Regions Information---
Region 0: Addr=0xbd520000 Size=32768 (0x8000)
Region 2: Addr=0xbd500000 Size=131072 (0x20000)

```

3.2.13 lib/ccrtaicc_msgdma

This test performs a modular scatter-gather DMA test on boards that support it. Additionally, it displays performance information for each mode of operation.

```

Usage: ./ccrtaicc_msgdma [-a AddrOff,ToAddrOff] [-b Board] [-d NumDesc]
      [-f Input,Output] [-i] [-l LoopCnt] [-m Mode]
      [-s TotalXferSize] [-v] [-X]

-a <AddrOff,ToAddrOff> (First Avalon Address Offset, default DiagRam offset)
                        (Second 'ToAddrOff' only for Avalon2Avalon mode)
-b <Board>              (board #, default = 0)
-d <NumDesc>           (Number of Descriptors, default = 1)
-f <Input>,<#Output>   (Use input file as input data. default None)
                        (Use Output file to write 'to' data. default None)
                        (Prepend with '#' to remove comments and address)
-i                     (Use interrupts, default is poll)
-l <LoopCnt>          (Loop Count, default = 1000)
-m <Mode>              (Mode of Operation, default = all)
  'a2p'                (Avalon memory address to Pci memory address)
  'p2a'                (Pci memory address to Avalon memory address)
  'p2p'                (Pci memory address to Pci memory address)
  'a2a'                (Avalon memory address to Avalon memory address)
  'all'                (All above modes with only memory addresses)

  'A2p'                (Avalon FIFO address to Pci memory address)
  'p2A'                (Pci memory address to Avalon FIFO address)
  'A2A'                (Avalon FIFO address to Avalon FIFO address)
-s <TotalXferSize>    (Total Transfer Size, default size of DiagRam)
                        (Maximum transfer size is 0x3FFFF)
-v                     (Verbose operation. default is quiet)
-X                     (Skip Data Validation, default is to validate)

```

Notes:

- 1) For modes 'p2a' or 'a2p' only the first address 'AddrOff' is used in option '-a'
- 2) For modes 'a2a' the first address 'AddrOff' is "FROM" and second address 'ToAddrOff' is "TO"
- 3) If Input file is specified in the '-f' option, its contents is used to seed input
- 4) If '-X' option is specified, no pattern is written to input, unless '-f Input' option is specified
- 5) Multiple '-m' options can be specified on a single command line
- 6) When address '-a' option is not specified, DiagRam offset is used for Analog input/output
- 7) Normal running process if no arguments specified is as follows:
 - a) Incrementing pattern written to the input using programmed I/O and readback validated
 - b) Output written with 'baadbeef' pattern using programmed I/O
 - c) Scatter-Gather DMA performed from Input to Output
 - d) Data is read back from both Input and Output using programmed I/O and compared
- 8) An upper case 'A' in the -m option represents an Avalon FIFO address, while

- a lower case 'a' in the -m option represents a regular Avalon memory address
- 9) If a regular memory Avalon address is specified as an Avalon FIFO address and vice-versa results will be unpredictable
- 10) When either input or output Avalon address is pointing to a FIFO, then data validation is skipped
- 11) If a size is specified for a memory or FIFO address that is greater than it can handle, the result will be unpredictable. You will need to reset the firmware to restore proper operation

```
e.g. ./ccrtaicc_msgdma -mall (Run all transfer modes with validation)
     ./ccrtaicc_msgdma -a0x8000 -s0x100 (Run all modes with Avalon Address
     0x8000 and size 0x100)
     ./ccrtaicc_msgdma -a0xA000 -s0x200 -ma2a (Run a2a with Avalon Address 0xA000 and
     size 0x200)
     ./ccrtaicc_msgdma -mp2a -l1 -dl -fHexFile_16K -a0x10004 -X
     (Transfer Input file to Avalon memory at 0x10004)
     ./ccrtaicc_msgdma -ma2p -l1 -dl -f,OutFile -s0x4000 -a0x10004 -X
     (Transfer Avalon memory at 0x10004 to output file
     'OutFile')
     ./ccrtaicc_msgdma -mA2p -l10000 -s0x20000 -dl6 -a0x18010
     (Transfer Avalon FIFO at 0x18010 to PCI memory with 16
     Descriptors where each descriptor has a transfer size
     of 0x2000 bytes. No validation will be performed)
```

Example display:

```
./ccrtaicc_msgdma
1000: P2P Total: Size 0x8000, Fire= 110.56us/ 296.40MB/s (mi/ma/av: 266.96/
296.49/ 296.11 MB/s, 110.52/ 122.75/ 110.66 us) LastWord=0x007cffff
1000: A2A Total: Size 0x4000, Fire= 93.01us/ 176.16MB/s (mi/ma/av: 165.27/
192.47/ 176.21 MB/s, 85.12/ 99.13/ 92.98 us) LastWord=0x003e7fff
1000: P2A Total: Size 0x8000, Fire= 110.41us/ 296.78MB/s (mi/ma/av: 280.92/
300.87/ 299.39 MB/s, 108.91/ 116.65/ 109.45 us) LastWord=0x007cffff
1000: A2P Total: Size 0x8000, Fire= 96.11us/ 340.93MB/s (mi/ma/av: 320.07/
371.48/ 341.26 MB/s, 88.21/ 102.38/ 96.02 us) LastWord=0x007cffff
```

3.2.14 lib/ccrtaicc_msgdma_info

This test provides useful modular scatter-gather DMA information for cards that support it.

```
Usage: ./ccrtaicc_msgdma_info [-b Board] [-l]
       -b <Board> (board #, default = 0)
```

Example display:

```
./ccrtaicc_msgdma_info
#### This card does not support modular scatter-gather DMA ####
===== Dispatcher =====
Status = 0x0000000a
Control = 0x0000000c
ReadFillLevel = 0x00000000
WriteFillLevel = 0x00000000
ResponseFillLevel = 0x00000000
ReadSequenceNumber = 0x00000001
WriteSequenceNumber = 0x00000001

===== Prefetcher =====
Status = 0x00000000
Control = 0x00000000
NextDescriptorPointer = 0x0000000000004800 (### Descriptor ID 1 ###)
DescriptorPollingFrequency = 0x00000000

===== Descriptors =====
ID Addr ReadAddr WritAddr Length Stat Control Rbct Wbct Rstr Wstr ActBytXfr NextPtr
```

```

=====
 1 (4800): 00008000 00850000 008000 0000 80000000 00 00 0000 0000 00000000 00004fc0
                                                    (Terminator)
 2 (4840): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
 3 (4880): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
 4 (48c0): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
 5 (4900): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
 6 (4940): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
 7 (4980): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
 8 (49c0): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
 9 (4a00): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
10 (4a40): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
11 (4a80): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
12 (4ac0): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
13 (4b00): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
14 (4b40): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
15 (4b80): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
16 (4bc0): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
17 (4c00): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
18 (4c40): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
19 (4c80): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
20 (4cc0): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
21 (4d00): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
22 (4d40): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
23 (4d80): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
24 (4dc0): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
25 (4e00): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
26 (4e40): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
27 (4e80): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
28 (4ec0): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
29 (4f00): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
30 (4f40): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
31 (4f80): 00000000 00000000 000000 0000 00000000 00 00 0000 0000 00000000 00000000
=====

```

3.2.15 lib/ccrtaicc_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: ./ccrtaicc_smp_affinity [-b Board] [-c CpuMask]
      -b Board          (Board number -- default is 0)
      -c CpuMask       (CPU mask in HEX -- default is none)

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

```

Example display:

```

./ccrtaicc_smp_affinity
(IRQ57) fc user f8 actual

```

```

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

```

3.2.16 lib/ccrtaicc_transfer

This test performs various DMA and Programmed I/O transfers between the board components and the PCI memory.

```

Usage: ./ccrtaicc_transfer [-b Board] [-c CaseNum] [-i] [-l LoopCnt]
      [-s XferSize]
      -b Board          (Board number -- default is 0)
      -c CaseNum       (Select Case Numbers -- default = ALL CASES)

```

```

-c 4,1,7-9 select case 1,4,7,8,9)
-c 8-      select case 8 to end)
-c -3     select case 1,2,3)
-i        (Enable Interrupts -- default = Disable)
-l LoopCnt (Loop Count -- default is 100)
-s XferSize (Avalon Ram Xfer Size in bytes -- default is 32768)

```

Example display:

```

./ccrtaiicc_transfer

local_ptr=0x7ffff7fd7000
Size of Avalon RAM = 32768 (0x00008000)
Physical Memory Information:
  UserPID      =28194
  PhysMemPtr   =0x4340000
  DriverVirtMemPtr=0xffff880004340000
  MmappedUserMemPtr=0x7ffff7fc5000
  PhysMemSize  =0x00008000
  PhysMemSizeFreed=0x00000000
  EntryInTxTbl =0
  NumOfEntriesUsed=1

1: Memory -> Avalon RAM (DMA0) (Size=0x8000):          100 (1404.64 us, 23.33 MBytes/Sec)
2: Memory -> Avalon RAM (DMA1) (Size=0x8000):          100 (1404.64 us, 23.33 MBytes/Sec)
3: Memory -> Avalon RAM (MSGDMA) (Size=0x8000):        100 (111.45 us, 294.01 MBytes/Sec)
4: Memory -> Avalon RAM (PIO) (Size=0x8000):           100 (124.85 us, 262.45 MBytes/Sec)
5: Avalon RAM -> Memory (DMA0) (Size=0x8000):          100 (529.98 us, 61.83 MBytes/Sec)
6: Avalon RAM -> Memory (DMA1) (Size=0x8000):          100 (530.00 us, 61.83 MBytes/Sec)
7: Avalon RAM -> Memory (MSGDMA) (Size=0x8000):        100 (90.12 us, 363.62 MBytes/Sec)
8: Avalon RAM -> Memory (PIO) (Size=0x8000):           100 (12380.83 us, 2.65 MBytes/Sec)
9: Memory -> Avalon ADC Calibration (DMA0) (Size=0x100): 10000 (17.14 us, 14.93 MBytes/Sec)
10: Memory -> Avalon ADC Calibration (PIO) (Size=0x100): 10000 (0.55 us, 462.64 MBytes/Sec)
11: Avalon ADC Calibration -> Memory (DMA0) (Size=0x100): 10000 (12.91 us, 19.82 MBytes/Sec)
12: Avalon ADC Calibration -> Memory (PIO) (Size=0x100): 10000 (131.22 us, 1.95 MBytes/Sec)
**** Test Passed ****

```

3.2.17 lib/ccrtaiicc_tst_lib

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

```

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

```

Example display:

```

./ccrtaiicc_tst_lib

Device Name: /dev/ccrtaiicc0
01 = Abort DMA
03 = Clear Driver Error
05 = Display BOARD Registers
07 = Dump Physical Memory List
09 = Get Board CSR
11 = Get Driver Error
13 = Get Cable Fault CSR
15 = Get Library Error
17 = Get Mapped Driver/Library Pointer
19 = Get Physical Memory
21 = Get Value
23 = MMap Physical Memory
25 = Reload Firmware
27 = Set Board CSR
29 = Set Test Bus Control
31 = ### ADC MENU ###

02 = Clear Cable Fault
04 = Clear Library Error
06 = Display CONFIG Registers
08 = Get All Boards Driver Information
10 = Get Board Information
12 = Get External Clock CSR
14 = Get Driver Information
16 = Get Mapped Config Pointer
18 = Get Mapped Local Pointer
20 = Get Test Bus Control
22 = Initialize Board
24 = Munmap Physical Memory
26 = Reset Board
28 = Set External Clock CSR
30 = Set Value
32 = ### CALIBRATION MENU ###

```

33 = ### CLOCK GENERATOR MENU ###

34 = ### INTERRUPT MENU ###

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

Main Selection ('h'=display menu, 'q'=quit)-> 31

Command: ADC_menu()

01 = ADC Activate	02 = ADC Disable
03 = ADC Reset (disable/activate)	04 = ADC Driver Read Operation
05 = ADC Get CSR	06 = ADC Get Driver Read Mode
07 = ADC Get FIFO Channel Select	08 = ADC Get FIFO Information
09 = ADC Get FIFO Status	10 = ADC Get FIFO Threshold
11 = ADC Get Input Control	12 = ADC Read Channels
13 = ADC Reset FIFO	14 = ADC Set CSR
15 = ADC Set Driver Read Mode	16 = ADC Set FIFO Channel Select
17 = ADC Set FIFO Threshold	18 = ADC Set Input Control

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

Main Selection ('h'=display menu, 'q'=quit)-> 32

Command: calibration_menu()

01 = ADC: Get Calibrated Values	02 = ADC: Perform Auto Calibration
03 = ADC: Perform External Negative Calib.	04 = ADC: Perform External Offset Calib.
05 = ADC: Perform External Positive Calib.	06 = ADC: Perform Negative Calibration
07 = ADC: Perform Offset Calibration	08 = ADC: Perform Positive Calibration
09 = ADC: Read Channels Calibration	10 = ADC: Reset Calibration
11 = ADC: Write Channels Calibration	12 = Get Calibration CSR
13 = Set Calibration CSR	

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

Main Selection ('h'=display menu, 'q'=quit)-> 33

Command: clock_generator_menu()

01 = Clock Disable Outputs	02 = Clock Enable Outputs
03 = Clock Get Generator CSR	04 = Clock Get Generator Dividers
05 = Clock Get Generator Information	06 = Clock Get Generator Input Clock Enable
07 = Clock Get Generator Input Clock Select	08 = Clock Get Generator Input Clock Status
09 = Clock Get Generator Output Config	10 = Clock Get Generator Output Format
11 = Clock Get Generator Output Mode	12 = Clock Get Generator Output Mux
13 = Clock Get Generator P-Divider Enable	14 = Clock Get Generator Revision
15 = Clock Get Generator Value	16 = Clock Get Generator Voltage Select
17 = Clock Get Generator Zero Delay	18 = Clock Set Generator CSR
19 = Clock Set Generator Dividers	20 = Clock Set Generator Input Clock Enable
21 = Clock Set Generator Input Clock Select	22 = Clock Set Generator Output Config
23 = Clock Set Generator Output Format	24 = Clock Set Generator Output Mode
25 = Clock Set Generator Output Mux	26 = Clock Set Generator P-Divider Enable
27 = Clock Set Generator Value	28 = Clock Set Generator Voltage Select
29 = Clock Set Generator Zero Delay	30 = Compute All Output Clocks
31 = Program All Output Clocks	32 = Read Clock Registers
33 = Reset Clock (Hardware)	34 = Soft Reset
35 = Update Clock Generator Divider	36 = Write Clock Registers

Clock Generator Selection ('h'=display menu, 'q'=quit)->

Main Selection ('h'=display menu, 'q'=quit)-> 34

Command: interrupt_menu()

01 = Add Irq	02 = Disable Pci Interrupts
03 = Enable Pci Interrupts	04 = Get Interrupt Status
05 = Get Interrupt Timeout	06 = Remove Irq
07 = Set Interrupt Status	08 = Set Interrupt Timeout

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

This page intentionally left blank