Software Interface
CCURUEGO (WC-UEGO)

PCIe 4-Channel Universal Exhaust Gas Oxygen Sensor Simulator Card (UEGO)

Driver	ccuruego (WC-UEGO)	
OS	RedHawk (CentOS or Ubuntu based)	
Vendor	Concurrent Real-Time	
Hardware	PCIe 4-Channel Universal Exhaust Gas Oxygen Sensor Simulator Card (UEGO)	
Author	Darius Dubash	
Date	October 3rd, 2023	Rev 2023.1

This page intentionally left blank

Table of Contents

1. INTRODUCTION 6

1.1 Related Documents 6

2. SOFTWARE SUPPORT 6

2.1 Direct Driver Access 6
2.1.1 open(2) system call 6
2.1.2 ioctl(2) system call 7
2.1.3 mmap(2) system call 9
2.1.4 read(2) system call 10
2.1.5 write(2) system call 10
2.2 Application Program Interface (API) Access 11
2.2.1 ccurUEGO_Abort_DMA() 13
2.2.2 ccurUEGO_Add_Irq() 13
2.2.3 ccurUEGO_CJ135_Command() 13
2.2.4 ccurUEGO_CJ135_Control_Activate() 14
2.2.5 ccurUEGO_CJ135_Get_External_Control() 15
2.2.6 ccurUEGO_CJ135_Get_Interrupt_Counter() 15
2.2.7 ccurUEGO_CJ135_Get_Register_Bank_Info() 15
2.2.8 ccurUEGO_CJ135_Get_Status() 16
2.2.9 ccurUEGO_CJ135_Interrupt_Activate() 17
2.2.10 ccurUEGO_CJ135_Read_Address() 17
2.2.11 ccurUEGO_CJ135_Read_RAM_Address() 18
2.2.12 ccurUEGO_CJ135_Set_External_Control() 19
2.2.13 ccurUEGO_CJ135_Write_Address() 19
2.2.14 ccurUEGO_Clear_Driver_Error() 20
2.2.15 ccurUEGO_Clear_Lib_Error() 20
2.2.16 ccurUEGO_Close() 20
2.2.17 ccurUEGO_DAC_Control_Activate() 20
2.2.18 ccurUEGO_DAC_Read_Channels() 21
2.2.19 ccurUEGO_DAC_Write_Channels() 22
2.2.20 ccurUEGO_DataToVolts() 22
2.2.21 ccurUEGO_Digital_Potentiometer_Activate() 22
2.2.22 ccurUEGO_Digital_Potentiometer_Get_Resistance() 23
2.2.23 ccurUEGO_Digital_Potentiometer_Set_Resistance() 23
2.2.24 ccurUEGO_Disable_Pci_Interrupts() 24
2.2.25 ccurUEGO_Enable_Pci_Interrupts() 24
2.2.26 ccurUEGO_Fast_Memcpy() 25
2.2.27 ccurUEGO_Fast_Memcpy_Unlocked() 25
2.2.28 ccurUEGO_Fraction_To_Hex() 25
2.2.29 ccurUEGO_Get_Board_CSR() 26
2.2.30 ccurUEGO_Get_Board_Info() 26
2.2.31 ccurUEGO_Get_CalibrationBus_Control() 27
2.2.32 ccurUEGO_Get_Driver_Error() 27
2.2.33 ccurUEGO_Get_Driver_Info() 28
2.2.34 ccurUEGO_Get_Driver_Read_Mode() 29
2.2.35 ccurUEGO_Get_Driver_Write_Mode() 29
2.2.36 ccurUEGO_Get_Interrupt_Control() 30
2.2.37 ccurUEGO_Get_Interrupt_Status() 30
2.2.38 ccurUEGO_Get_Interrupt_Timeout_Seconds() 31
2.2.39 ccurUEGO_Get_Lib_Error_Description() 31
2.2.40 ccurUEGO_Get_Lib_Error() 31

2.2.41 ccurUEGO_Get_Mapped_Config_Ptr() 33
2.2.42 ccurUEGO_Get_Mapped_Driver_Library_Ptr() 33
2.2.43 ccurUEGO_Get_Mapped_Local_Ptr() 34
2.2.44 ccurUEGO_Get_Open_File_Descriptor() 34
2.2.45 ccurUEGO_Get_Physical_Memory() 35
2.2.46 ccurUEGO_Get_Value() 35
2.2.47 ccurUEGO_Hex_To_Fraction() 36
2.2.48 ccurUEGO_Identify_Board() 36
2.2.49 ccurUEGO_Initialize_Board() 37
2.2.50 ccurUEGO_IO_Control_Activate() 37
2.2.51 ccurUEGO_IO_Get_Control() 38
2.2.52 ccurUEGO_IO_Set_Control() 38
2.2.53 ccurUEGO_MMap_Physical_Memory() 39
2.2.54 ccurUEGO_Munmap_Physical_Memory() 40
2.2.55 ccurUEGO_NanoDelay() 40
2.2.56 ccurUEGO_Open() 40
2.2.57 ccurUEGO_PWM_CalcDutyCycle() 41
2.2.58 ccurUEGO_PWM_CalcFreqinHz() 41
2.2.59 ccurUEGO_PWM_CalcPeriodinUsec() 42
2.2.60 ccurUEGO_PWM_Control_Activate() 42
2.2.61 ccurUEGO_PWM_Get_Count() 42
2.2.62 ccurUEGO_PWM_Get_Diagnostic_Count() 43
2.2.63 ccurUEGO_PWM_Get_Diagnostic_Frequency_Duty() 43
2.2.64 ccurUEGO_PWM_Get_Diagnostic_Status() 44
2.2.65 ccurUEGO_PWM_Set_Diagnostic_Count() 44
2.2.66 ccurUEGO_PWM_Set_Diagnostic_Frequency_Duty() 44
2.2.67 ccurUEGO_Read() 45
2.2.68 ccurUEGO_Read_Serial_Prom() 46
2.2.69 ccurUEGO_Read_Serial_Prom_Item() 46
2.2.70 ccurUEGO_Remove_Irq() 46
2.2.71 ccurUEGO_Reset_Board() 47
2.2.72 ccurUEGO_Select_Driver_Read_Mode() 47
2.2.73 ccurUEGO_Select_Driver_Write_Mode() 48
2.2.74 ccurUEGO_Serial_Prom_Write_Override() 48
2.2.75 ccurUEGO_Set_Board_CSR() 49
2.2.76 ccurUEGO_Set_CalibrationBus_Control() 49
2.2.77 ccurUEGO_Set_Interrupt_Control() 50
2.2.78 ccurUEGO_Set_Interrupt_Status() 50
2.2.79 ccurUEGO_Set_Interrupt_Timeout_Seconds() 51
2.2.80 ccurUEGO_Set_Value() 51
2.2.81 ccurUEGO_VoltsToData() 52
2.2.82 ccurUEGO_VoltsToDataChanCal() 52
2.2.83 ccurUEGO_Wait_For_Interrupt() 53
2.2.84 ccurUEGO_Write() 53
2.2.85 ccurUEGO_Write_Serial_Prom() 54
2.2.86 ccurUEGO_Write_Serial_Prom_Item() 54

3. TEST PROGRAMS 56

3.1 Direct Driver Access Example Tests 56
3.1.1 ccuruego_dump 56
3.1.2 ccuruego_rdreg 59
3.1.3 ccuruego_reg 59
3.1.4 ccuruego_regedit 63
3.1.5 ccuruego_tst 63
3.1.6 ccuruego_wreg 63
3.1.7 Flash/ccuruego_flash 64

3.1.8 Flash/ccuruego_fwreload 64
3.1.9 Eeprom/ccuruego_eeprom 65
3.2 Application Program Interface (API) Access Example Tests 65
3.2.1 lib/ccuruego_disp 65
3.2.1.1 lib/ccuruego_display – Injecting Faults 72
3.2.2 lib/ccuruego_identify 72
3.2.3 lib/ccuruego_info 73
3.2.4 lib/ccuruego_pwm 75
3.2.5 lib/ccuruego_tst_lib 77
3.2.6 lib/Sprom/ccuruego_sprom 78

This page intentionally left blank

1. Introduction

This document provides the software interface to the ***ccuruego*** driver which communicates with the Concurrent Real-Time PCI Express 4-Channel Universal Exhaust Gas Oxygen Sensor Simulator Card (UEGO).

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

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

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

1.1 Related Documents

- PCIe 4-Channel Universal Exhaust Gas Oxygen Sensor Simulator Driver Installation on RedHawk Release Notes 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/ccuruego0", 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/ccuruego<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.

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_CCURUEGO_ABORT_DMA
IOCTL_CCURUEGO_ADD_IRQ
IOCTL_CCURUEGO_DISABLE_PCI_INTERRUPTS
IOCTL_CCURUEGO_ENABLE_PCI_INTERRUPTS
IOCTL_CCURUEGO_GET_DRIVER_ERROR
IOCTL_CCURUEGO_GET_DRIVER_INFO
IOCTL_CCURUEGO_GET_PHYSICAL_MEMORY
IOCTL_CCURUEGO_GET_READ_MODE
IOCTL_CCURUEGO_GET_WRITE_MODE
IOCTL_CCURUEGO_INIT_BOARD
IOCTL_CCURUEGO_INTERRUPT_TIMEOUT_SECONDS
IOCTL_CCURUEGO_MAIN_CONTROL_REGISTERS
IOCTL_CCURUEGO_MMAP_SELECT
IOCTL_CCURUEGO_NO_COMMAND
IOCTL_CCURUEGO_PCI_BRIDGE_REGISTERS
IOCTL_CCURUEGO_PCI_CONFIG_REGISTERS
IOCTL_CCURUEGO_READ_EEPROM
IOCTL_CCURUEGO_REMOVE_IRQ
IOCTL_CCURUEGO_RESET_BOARD
IOCTL_CCURUEGO_SELECT_READ_MODE
IOCTL_CCURUEGO_SELECT_WRITE_MODE
IOCTL_CCURUEGO_WAIT_FOR_INTERRUPT
IOCTL_CCURUEGO_WRITE_EEPROM
```

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

IOCTL_CCURUEGO_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_CCURUEGO_REMOVE_IRQ* call earlier to remove the interrupt handler.

IOCTL_CCURUEGO_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_CCURUEGO_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_CCURUEGO_GET_DRIVER_ERROR: The argument supplied to this *ioctl* is a pointer to the *ccuruego_user_error_t* structure. Information on the structure is located in the *ccuruego_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 *CCURUEGO_SUCCESS.*

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

IOCTL_CCURUEGO_GET_PHYSICAL_MEMORY: The argument supplied to this *ioctl* is a pointer to the *ccuruego_user_phys_mem_t* structure. Information on the structure is located in the *ccuruego_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_CCURUEGO_GET_READ_MODE: The argument supplied to this *ioctl* is a pointer an *unsigned long int*. The value returned will be one of the read modes as defined by the *enum _ccuruego_driver_rw_mode_t* located in the *ccuruego_user.h* include file. Though this is an analog output card, the user can read last values of the channel registers that were written to. If user is writing data to the board using the on-board FIFO, then the channel registers would reflect the most recent FIFO data that was output by the board. FIFO operation is not supported by the read mode as the FIFO is a write only register.

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

IOCTL_CCURUEGO_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_CCURUEGO_RESET_BOARD* call.

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

IOCTL_CCURUEGO_MMAP_SELECT: The argument to this *ioctl* is a pointer to the *ccuruego_mmap_select_t* structure. Information on the structure is located in the *ccuruego_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 *(CCURUEGO_SELECT_LOCAL_MMAP)*, the configuration register space *(CCURUEGO_SELECT_CONFIG_MMAP)* the physical memory *(CCURUEGO_SELECT_PHYS_MEM_MMAP)* that is created by the *mmap(2)* system call and the driver/library mapping *(CCURUEGO_SELECT_DRIVER_LIBRARY_MMAP)*.

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

IOCTL_CCURUEGO_PCI_CONFIG_REGISTERS: The argument supplied to this *ioctl* is a pointer to the *ccuruego_pci_config_reg_addr_mapping_*t structure whose definition is located in the *ccuruego_user.h* include file.

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

IOCTL_CCURUEGO_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_CCURUEGO_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_CCURUEGO_INIT_BOARD* call.

IOCTL_CCURUEGO_SELECT_READ_MODE: The argument supplied to this *ioctl* is a pointer an *unsigned long int*. The value set will be one of the read modes as defined by the *enum _ccuruego_driver_rw_mode_t* located in the *ccuruego_user.h* include file. FIFO operation is not supported by the read mode as the FIFO is a write only register.

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

IOCTL_CCURUEGO_WAIT_FOR_INTERRUPT: The argument to this *ioctl* is a pointer to the *ccuruego_driver_int_t* structure. Information on the structure is located in the *ccuruego_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_CCURUEGO_WRITE_EEPROM: The argument to this *ioctl* is a pointer to the *ccuruego_eeprom_t* structure. Information on the structure is located in the *ccuruego_user.h* include file. This call is specifically used by the supplied *eeprom* application and should not be used by the user.

2.1.3 mmap(2) system call

This system call provides the ability to map either the local board registers, the configuration board registers, 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_CCURUEGO_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;
ccuruego_local_ctrl_data_t  *local_ptr;
ccuruego_mmap_select_t  mmap_select;
unsigned long mmap_local_size;

mmap_select.select = CCURUEGO_SELECT_LOCAL_MMAP;
mmap_select.offset=0;
mmap_select.size=0;
ioctl(fp, IOCTL_CCURUEGO_MMAP_SELECT,(void *)&mmap_select);
mmap_local_size = mmap_select.size;
```

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

local_ptr = (ccuruego_local_ctrl_data_t *)munmap_local_ptr;
local_ptr = (ccuruego_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 programmed I/O and DMA reads of offset DAC channel registers.The option selected is determined by the *ccurUEGO_Select_Driver_Read_Mode()* call.

2.1.5 write(2) system call

This system call currently supports programmed I/O and DMA writes of offset DAC channel registers.The option selected is determined by the *ccurUEGO_Select_Driver_Write_Mode()* call.

2.2 Application Program Interface (API) Access

The API is the recommended method of communicating with the board for most users.

There are a lot of APIs that have multiple arguments to set various parameters. If the user only wishes to change certain parameters for the call, they need to get the current settings via a query API, change only those parameters that need to be modified and then invoke a setting API to update these parameters *(i.e. read/modify/write)*. This is a two API call operation.

A nice feature has been implemented in these APIs to simplify the user programming by having a common parameter CCURUEGO_DO_NOT_CHANGE which is a #define, that can be used for a lot of these calls. Arguments with this parameter will therefore cause the API to perform the read/modify/write operation instead of the user performing the same function with two API calls. The drawback to this approach is that some compilers will complain about the use of this parameter and therefore the user will require appropriate casting to get rid of warnings/errors.

The following are a list of calls that are available.

```
ccurUEGO_Abort_DMA()
ccurUEGO_Add_Irq()
ccurUEGO_CJ135_Command()
ccurUEGO_CJ135_Control_Activate()
ccurUEGO_CJ135_Get_External_Control()
ccurUEGO_CJ135_Get_Interrupt_Counter()
ccurUEGO_CJ135_Get_Register_Bank_Info()
ccurUEGO_CJ135_Get_Status()
ccurUEGO_CJ135_Interrupt_Activate()
ccurUEGO_CJ135_Read_Address()
ccurUEGO_CJ135_Read_RAM_Address()
ccurUEGO_CJ135_Set_External_Control()
ccurUEGO_CJ135_Write_Address()
ccurUEGO_Clear_Driver_Error()
ccurUEGO_Clear_Lib_Error()
ccurUEGO_Close()
ccurUEGO_DAC_Control_Activate()
ccurUEGO_DAC_Read_Channels()
ccurUEGO_DAC_Write_Channels()
ccurUEGO_DataToVolts()
ccurUEGO_Digital_Potentiometer_Activate()
ccurUEGO_Digital_Potentiometer_Get_Resistance()
ccurUEGO_Digital_Potentiometer_Set_Resistance()
ccurUEGO_Disable_Pci_Interrupts()
ccurUEGO_Enable_Pci_Interrupts()
ccurUEGO_Fast_Memcpy()
ccurUEGO_Fast_Memcpy_Unlocked()
ccurUEGO_Fraction_To_Hex()
ccurUEGO_Get_Board_CSR()
ccurUEGO_Get_Board_Info()
ccurUEGO_Get_CalibrationBus_Control()
ccurUEGO_Get_Driver_Error()
ccurUEGO_Get_Driver_Info()
ccurUEGO_Get_Driver_Read_Mode()
ccurUEGO_Get_Driver_Write_Mode()
ccurUEGO_Get_Interrupt_Control()
ccurUEGO_Get_Interrupt_Status()
ccurUEGO_Get_Interrupt_Timeout_Seconds()
ccurUEGO_Get_Lib_Error_Description()
```

```
ccurUEGO_Get_Lib_Error()
ccurUEGO_Get_Mapped_Config_Ptr()
ccurUEGO_Get_Mapped_Driver_Library_Ptr()
ccurUEGO_Get_Mapped_Local_Ptr()
ccurUEGO_Get_Open_File_Descriptor()
ccurUEGO_Get_Physical_Memory()
ccurUEGO_Get_Value()
ccurUEGO_Hex_To_Fraction()
ccurUEGO_Identify_Board()
ccurUEGO_Initialize_Board()
ccurUEGO_IO_Control_Activate()
ccurUEGO_IO_Get_Control()
ccurUEGO_IO_Set_Control()
ccurUEGO_MMap_Physical_Memory()
ccurUEGO_Munmap_Physical_Memory()
ccurUEGO_NanoDelay()
ccurUEGO_Open()
ccurUEGO_PWM_CalcDutyCycle()
ccurUEGO_PWM_CalcFreqinHz()
ccurUEGO_PWM_CalcPeriodinUsec()
ccurUEGO_PWM_Control_Activate()
ccurUEGO_PWM_Get_Count()
ccurUEGO_PWM_Get_Diagnostic_Count()
ccurUEGO_PWM_Get_Diagnostic_Status()
ccurUEGO_PWM_Set_Diagnostic_Count()
ccurUEGO_PWM_Set_Diagnostic_Frequency_Duty()
ccurUEGO_PWM_Set_Diagnostic_Signal()
ccurUEGO_Read()
ccurUEGO_Read_Serial_Prom()
ccurUEGO_Read_Serial_Prom_Item()
ccurUEGO_Remove_Irq()
ccurUEGO_Reset_Board()
ccurUEGO_Select_Driver_Read_Mode()
ccurUEGO_Select_Driver_Write_Mode()
ccurUEGO_Serial_Prom_Write_Override()
ccurUEGO_Set_Board_CSR()
ccurUEGO_Set_CalibrationBus_Control()
ccurUEGO_Set_Interrupt_Control()
ccurUEGO_Set_Interrupt_Status()
ccurUEGO_Set_Interrupt_Timeout_Seconds()
ccurUEGO_Set_Value()
ccurUEGO_VoltsToData()
ccurUEGO_VoltsToDataChanCal()
ccurUEGO_Wait_For_Interrupt()
ccurUEGO_Write()
ccurUEGO_Write_Serial_Prom()
ccurUEGO_Write_Serial_Prom_Item()
```

2.2.1 ccurUEGO_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.

```
/************************************************************************
   _ccuruego_lib_error_number_t ccurUEGO_Abort_DMA(void *Handle)

   Description: Abort any DMA in progress

   Input:   void *Handle                        (Handle pointer)
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR          (successful)
               # CCURUEGO_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN          (device not open)
               # CCURUEGO_LIB_NO_LOCAL_REGION   (local region not present)
               # CCURUEGO_LIB_IOCTL_FAILED      (driver ioctl call failed)
************************************************************************/
```

2.2.2 ccurUEGO_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 ccurUEGO_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 ccurUEGO_Remove_Irq(), then this call adds it back.

   Input:   void *Handle                        (Handle pointer)
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR          (successful)
               # CCURUEGO_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN          (library not open)
               # CCURUEGO_LIB_IOCTL_FAILED      (driver ioctl call failed)
************************************************************************/
```

2.2.3 ccurUEGO_CJ135_Command()

This call is used to control the CJ135 in order to test the UEGO. Normally, the CJ135 is used inside an engine control unit (ECU) for a Gasoline or a Diesel Engine. It has been successfully operated with sensors from Bosch, NTK, and other suppliers.

```
/************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_CJ135_Command(void                                    *Handle,
                          _ccuruego_cj135_command_data_status_t Command,
                          void                                    *Data)

   Description: Issue CJ135 Command

   Input:   void                                  *Handle   (handle pointer)
            _ccuruego_cj135_command_data_status_t Command   (CJ135 command)
                # CCURUEGO_CJ135_CMD_READ_DEVICE_ID
                # CCURUEGO_CJ135_CMD_READ_DEVICE_REVISION
                # CCURUEGO_CJ135_CMD_READ_HW_REVISION
                # CCURUEGO_CJ135_CMD_SOFTWARE_RESET
                # CCURUEGO_CJ135_CMD_READ_CHIP_STATUS
```

```
                # CCURUEGO_CJ135_CMD_READ_DIAGNOSTIC_0
                # CCURUEGO_CJ135_CMD_READ_DSP_STATUS
                # CCURUEGO_CJ135_CMD_READ_DIAGNOSTIC_1
                # CCURUEGO_CJ135_CMD_READ_RAM_MULTIPLE
   Output:  void                              *Data     (pointer to Data)
   Return:  _ccuruego_lib_error_number_t
                # CCURUEGO_LIB_NO_ERROR               (successful)
                # CCURUEGO_LIB_NO_LOCAL_REGION        (error)
                # CCURUEGO_LIB_CJ135_BUSY             (CJ135 prom busy)
                # CCURUEGO_LIB_NOT_OPEN               (device not open)
                # CCURUEGO_LIB_INVALID_ARG            (invalid argument)
                # CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE
                                                      (CJ135 control not active)
*************************************************************************/
```

The following commands require a pointer to a byte Data where the information is returned.

```
      # CCURUEGO_CJ135_CMD_READ_DEVICE_ID
      # CCURUEGO_CJ135_CMD_READ_DEVICE_REVISION
      # CCURUEGO_CJ135_CMD_READ_HW_REVISION
      # CCURUEGO_CJ135_CMD_READ_CHIP_STATUS
      # CCURUEGO_CJ135_CMD_READ_DIAGNOSTIC_0
      # CCURUEGO_CJ135_CMD_READ_DSP_STATUS
      # CCURUEGO_CJ135_CMD_READ_DIAGNOSTIC_1
```

The following command does not require and Data argument as nothing is returned.

```
      # CCURUEGO_CJ135_CMD_SOFTWARE_RESET
```

The following command requires a Data argument that is a pointer to *CCURUEGO_CJ135_RAM_BANK_SIZE* shorts.

```
      # CCURUEGO_CJ135_CMD_READ_RAM_MULTIPLE
```

When the CJ135 engine is running, it is collecting data for the mode of operation and saving it in the CJ135 RAM Bank approximately every 10 milli-seconds. It has a size 64-short words of size *CCURUEGO_CJ135_RAM_BANK_SIZE*. This command returns to the user the entire contents of the RAM Bank ensuring that the data within the bank is contiguous.

2.2.4 ccurUEGO_CJ135_Control_Activate()

This is the first call that needs to be issued to the CJ135 to activate the component. Until the CJ135 has been activated, it will not respond to any commands. The user can also use this call to get the current state of the CJ135 without accually changing it by supplying *CCURUEGO_CJ135_CONTROL_ENABLE_DO_NOT_CHANGE* to the *activate* argument and specifying a pointer to return the *current_state* argument.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_CJ135_Control_Activate (void                          *Handle,
                                    _ccuruego_cj135_control_enable_t activate,
                                    _ccuruego_cj135_control_enable_t *current_state)

   Description: Activate/DeActivate CJ135 Control module

   Input:   void                                  *Handle    (Handle pointer)
            _ccuruego_cj135_control_enable_t      activate   (activate/deactivate)
               # CCURUEGO_CJ135_CONTROL_DISABLE
               # CCURUEGO_CJ135_CONTROL_ENABLE
               # CCURUEGO_CJ135_CONTROL_ENABLE_DO_NOT_CHANGE
   Output:  _ccuruego_cj135_control_enable_t      *current_state (active/deactive)
               # CCURUEGO_CJ135_CONTROL_DISABLE
               # CCURUEGO_CJ135_CONTROL_ENABLE
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR          (successful)
```

```
                # CCURUEGO_LIB_BAD_HANDLE        (no/bad handler supplied)
                # CCURUEGO_LIB_NOT_OPEN          (device not open)
                # CCURUEGO_LIB_INVALID_ARG       (invalid argument)
                # CCURUEGO_LIB_NO_LOCAL_REGION   (local region not present)
*************************************************************************/
```

2.2.5 ccurUEGO_CJ135_Get_External_Control()

This call returns the external controls of the CJ135.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_CJ135_Get_External_Control(void                       *Handle,
                              _ccuruego_cj135_external_control_t *CJ135_ExtCtrl)

   Description: Get CJ135 External Control

   Input:   void                                 *Handle         (handle pointer)
   Output:  _ccuruego_cj135_external_control_t *CJ135_ExtCtrl  (CJ135 external
                                                               control)
               _ccuruego_cj135_external_signal_t  external_signal
                   # CCURUEGO_CJ135_EXTERNAL_SIGNAL_DISABLE
                   # CCURUEGO_CJ135_EXTERNAL_SIGNAL_ENABLE
               _ccuruego_cj135_external_fault_ground_t fault_ground
                   # CCURUEGO_CJ135_EXTERNAL_FAULT_GROUND_CONNECT_DISCONNECT
                   # CCURUEGO_CJ135_EXTERNAL_FAULT_GROUND_CONNECT_CONNECT
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR              (successful)
               # CCURUEGO_LIB_NO_LOCAL_REGION       (error)
               # CCURUEGO_LIB_NOT_OPEN              (device not open)
               # CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE
                                                    (CJ135 control not active)
*************************************************************************/
```

2.2.6 ccurUEGO_CJ135_Get_Interrupt_Counter()

This call returns the CJ135 interrupt counter.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_CJ135_Get_Interrupt_Counter (void     *Handle,
                                         u_short  *CJ135_IntCounter)

   Description: Get CJ135 Interrupt Counter

   Input:   void                   *Handle           (Handle pointer)
   Output:  u_short                *CJ135_IntCounter (CJ135 interrupt counter)
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR              (successful)
               # CCURUEGO_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN              (device not open)
               # CCURUEGO_LIB_INVALID_ARG           (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE
                                                    (CJ135 control not active)
*************************************************************************/
```

2.2.7 ccurUEGO_CJ135_Get_Register_Bank_Info()

This call returns various CJ135 bank registers.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
```

```
    ccurUEGO_CJ135_Get_Register_Bank_Info(void                  *Handle,
                              _ccuruego_cj135_register_bank_t *CJ135_RegsterBank)

    Description: Get CJ135 Register Bank Info

    Input:   void                              *Handle              (handle pointer)
    Output:  _ccuruego_cj135_register_bank_t *CJ135_RegisterBank  (CJ135 Register
                                                                   Bank)
                 u_short                                 Mode
                 u_short                                 UnSet
                 u_short                                 KP
                 u_short                                 KI
                 u_short                                 KD
                 u_short                                 Up0Lean
                 u_short                                 Up0Rich
                 u_short                                 IpSet
                 u_short                                 IpBlack
                 u_short                                 KRF
                 u_short                                 IpOff
                 u_short                                 SteerPat
                 u_short                                 ConfigDsp
                 u_short                                 Trim
                 u_short                                 RefPat
                 u_short                                 Free
                 u_short                                 ActualMode
    Return:  _ccuruego_lib_error_number_t
                 # CCURUEGO_LIB_NO_ERROR             (successful)
                 # CCURUEGO_LIB_NO_LOCAL_REGION      (error)
                 # CCURUEGO_LIB_NOT_OPEN             (device not open)
                 # CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE
                                                     (CJ135 control not active)
*****************************************************************************/
```

2.2.8 ccurUEGO_CJ135_Get_Status()

This call returns various CJ135 status information.

```
/*****************************************************************************
    _ccuruego_lib_error_number_t
    ccurUEGO_CJ135_Get_Status(void                     *Handle,
                              _ccuruego_cj135_status_t *CJ135_Status)

    Description: Get CJ135 Status

    Input:   void                                 *Handle         (handle pointer)
    Output:  _ccuruego_cj135_status_t             *CJ135_Status   (CJ135 Status)
                 u_short                          DeviceID
                 u_short                          DeviceRevision
                 u_short                          HardwareRevision
                 u_short                          ChipStatus
                 u_short                          Diag0
                 u_short                          DspStatus
                 u_short                          Diag1
                 u_short                          InterruptCounter
                 _ccuruego_cj135_interrupt_enable_t  InterruptState
                     # CCURUEGO_CJ135_INTERRUPT_DISABLE
                     # CCURUEGO_CJ135_INTERRUPT_ENABLE
    Return:  _ccuruego_lib_error_number_t
                 # CCURUEGO_LIB_NO_ERROR               (successful)
                 # CCURUEGO_LIB_NO_LOCAL_REGION        (error)
                 # CCURUEGO_LIB_NOT_OPEN               (device not open)
                 # CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE
                                                       (CJ135 control not active)
```

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

2.2.9 ccurUEGO_CJ135_Interrupt_Activate()

This is the first call that needs to be issued to the CJ135 Interrupt to activate the component. Until the CJ135 Interrupt has been activated, it will not respond to any commands. The user can also use this call to get the current state of the CJ135 interrupt without accually changing it by supplying *CCURUEGO_CJ135_INTERRUPT_ENABLE_DO_NOT_CHANGE* to the *activate* argument and specifying a pointer to return the *current_state* argument.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_CJ135_Interrupt_Activate (void                          *Handle,
                              _ccuruego_cj135_interrupt_enable_t activate,
                              _ccuruego_cj135_interrupt_enable_t *current_state)

   Description: Activate/DeActivate CJ135 Interrupt module

   Input:   void                                  *Handle     (Handle pointer)
            _ccuruego_cj135_interrupt_enable_t  activate    (activate/deactivate)
               # CCURUEGO_CJ135_INTERRUPT_DISABLE
               # CCURUEGO_CJ135_INTERRUPT_ENABLE
               # CCURUEGO_CJ135_INTERRUPT_ENABLE_DO_NOT_CHANGE
   Output:  _ccuruego_cj135_interrupt_enable_t  *current_state  (active/deactive)
               # CCURUEGO_CJ135_INTERRUPT_DISABLE
               # CCURUEGO_CJ135_INTERRUPT_ENABLE
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR              (successful)
               # CCURUEGO_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN              (device not open)
               # CCURUEGO_LIB_INVALID_ARG           (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE
                                                    (CJ135 control not active)
*****************************************************************************/
```

2.2.10 ccurUEGO_CJ135_Read_Address()

This call provides the user the ability to read some of the CJ135 internal registers.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_CJ135_Read_Address(void                        *Handle,
                              _ccuruego_cj135_address_t CJ135_Address,
                              u_short                   *Data)

   Description: Read CJ135 Address Command

   Input:   void                                  *Handle       (handle pointer)
            _ccuruego_cj135_address_t             CJ135_Address (CJ135 address)
                # CCURUEGO_CJ135_ADDR_MODE
                # CCURUEGO_CJ135_ADDR_UNSET
                # CCURUEGO_CJ135_ADDR_KP
                # CCURUEGO_CJ135_ADDR_KI
                # CCURUEGO_CJ135_ADDR_KD
                # CCURUEGO_CJ135_ADDR_UP0LEAN
                # CCURUEGO_CJ135_ADDR_UP0RICH
                # CCURUEGO_CJ135_ADDR_IP_SET
                # CCURUEGO_CJ135_ADDR_IP_BLACK
                # CCURUEGO_CJ135_ADDR_KRF
                # CCURUEGO_CJ135_ADDR_IPOFF
                # CCURUEGO_CJ135_ADDR_STEERPAT
                # CCURUEGO_CJ135_ADDR_CONFIGDSP
```

```
                # CCURUEGO_CJ135_ADDR_TRIM
                # CCURUEGO_CJ135_ADDR_REFPAT
                # CCURUEGO_CJ135_ADDR_FREE
                # CCURUEGO_CJ135_ADDR_ACTUAL_MODE
   Output:  u_short                          *Data     (pointer to Data)
   Return:  _ccuruego_lib_error_number_t
                # CCURUEGO_LIB_NO_ERROR               (successful)
                # CCURUEGO_LIB_NO_LOCAL_REGION        (error)
                # CCURUEGO_LIB_CJ135_BUSY             (CJ135 prom busy)
                # CCURUEGO_LIB_NOT_OPEN               (device not open)
                # CCURUEGO_LIB_INVALID_ARG            (invalid argument)
                # CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE
                                                      (CJ135 control not active)
*************************************************************************/
```

2.2.11 ccurUEGO_CJ135_Read_RAM_Address()

When the CJ135 engine is running, it is collecting data for the particular mode of operation and saving it in the CJ135 RAM Bank which is 64-words in size *(CCURUEGO_CJ135_RAM_BANK_SIZE).* This call gives the user the ability to supply an offset address into the RAM Bank and return its contents one short word at a time. Alternatively, the user can read the entire RAM bank by using the the *ccurUEGO_CJ135_Command()*call with the *CCURUEGO_CJ135_CMD_READ_RAM_MULTIPLE* command.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_CJ135_Read_Address(void                      *Handle,
                               _ccuruego_cj135_address_t CJ135_Address,
                               u_short                   *Data)

   Description: Read CJ135 Address Command

   Input:   void                               *Handle       (handle pointer)
            _ccuruego_cj135_address_t          CJ135_Address (CJ135 address)
                # CCURUEGO_CJ135_ADDR_MODE
                # CCURUEGO_CJ135_ADDR_UNSET
                # CCURUEGO_CJ135_ADDR_KP
                # CCURUEGO_CJ135_ADDR_KI
                # CCURUEGO_CJ135_ADDR_KD
                # CCURUEGO_CJ135_ADDR_UP0LEAN
                # CCURUEGO_CJ135_ADDR_UP0RICH
                # CCURUEGO_CJ135_ADDR_IP_SET
                # CCURUEGO_CJ135_ADDR_IP_BLACK
                # CCURUEGO_CJ135_ADDR_KRF
                # CCURUEGO_CJ135_ADDR_IPOFF
                # CCURUEGO_CJ135_ADDR_STEERPAT
                # CCURUEGO_CJ135_ADDR_CONFIGDSP
                # CCURUEGO_CJ135_ADDR_TRIM
                # CCURUEGO_CJ135_ADDR_REFPAT
                # CCURUEGO_CJ135_ADDR_FREE
                # CCURUEGO_CJ135_ADDR_ACTUAL_MODE
   Output:  u_short                           *Data     (pointer to Data)
   Return:  _ccuruego_lib_error_number_t
                # CCURUEGO_LIB_NO_ERROR               (successful)
                # CCURUEGO_LIB_NO_LOCAL_REGION        (error)
                # CCURUEGO_LIB_CJ135_BUSY             (CJ135 prom busy)
                # CCURUEGO_LIB_NOT_OPEN               (device not open)
                # CCURUEGO_LIB_INVALID_ARG            (invalid argument)
                # CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE
                                                      (CJ135 control not active)
*************************************************************************/
```

2.2.12 ccurUEGO_CJ135_Set_External_Control()

This call allows the user to set the external controls for the CJ135.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_CJ135_Set_External_Control(void                        *Handle,
                              _ccuruego_cj135_external_control_t *CJ135_ExtCtrl)

   Description: Set CJ135 External Control

   Input:   void                                  *Handle         (handle pointer)
   Output:  _ccuruego_cj135_external_control_t  *CJ135_ExtCtrl  (CJ135 external
                                                                 control)
                _ccuruego_cj135_external_signal_t       external_signal
                    # CCURUEGO_CJ135_EXTERNAL_SIGNAL_DISABLE
                    # CCURUEGO_CJ135_EXTERNAL_SIGNAL_ENABLE
                _ccuruego_cj135_external_fault_ground_t fault_ground
                    # CCURUEGO_CJ135_EXTERNAL_FAULT_GROUND_CONNECT_DISCONNECT
                    # CCURUEGO_CJ135_EXTERNAL_FAULT_GROUND_CONNECT_CONNECT
   Return:  _ccuruego_lib_error_number_t
                # CCURUEGO_LIB_NO_ERROR               (successful)
                # CCURUEGO_LIB_NO_LOCAL_REGION        (error)
                # CCURUEGO_LIB_NOT_OPEN               (device not open)
                # CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE
                                                      (CJ135 control not active)
*************************************************************************/
```

2.2.13 ccurUEGO_CJ135_Write_Address()

This call provides the user the ability to write to some of the CJ135 internal registers.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_CJ135_Write_Address(void                       *Handle,
                               _ccuruego_cj135_address_t Address,
                               u_short                   *Data)

   Description: Write CJ135 Address Command

   Input:   void                                  *Handle   (handle pointer)
            _ccuruego_cj135_address_t             Address   (CJ135 address)
                # CCURUEGO_CJ135_ADDR_MODE
                # CCURUEGO_CJ135_ADDR_UNSET
                # CCURUEGO_CJ135_ADDR_KP
                # CCURUEGO_CJ135_ADDR_KI
                # CCURUEGO_CJ135_ADDR_KD
                # CCURUEGO_CJ135_ADDR_UP0LEAN
                # CCURUEGO_CJ135_ADDR_UP0RICH
                # CCURUEGO_CJ135_ADDR_IP_SET
                # CCURUEGO_CJ135_ADDR_IP_BLACK
                # CCURUEGO_CJ135_ADDR_KRF
                # CCURUEGO_CJ135_ADDR_IPOFF
                # CCURUEGO_CJ135_ADDR_STEERPAT
                # CCURUEGO_CJ135_ADDR_CONFIGDSP
                # CCURUEGO_CJ135_ADDR_TRIM
                # CCURUEGO_CJ135_ADDR_REFPAT
                # CCURUEGO_CJ135_ADDR_FREE
                # CCURUEGO_CJ135_ADDR_ACTUAL_MODE
   Output:  u_short                        *Data     (pointer to Data)
   Return:  _ccuruego_lib_error_number_t
                # CCURUEGO_LIB_NO_ERROR             (successful)
                # CCURUEGO_LIB_NO_LOCAL_REGION      (error)
```

```
                # CCURUEGO_LIB_CJ135_BUSY           (CJ135 prom busy)
                # CCURUEGO_LIB_NOT_OPEN             (device not open)
                # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                # CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE
                                                    (CJ135 control not active)
*****************************************************************************/
```

2.2.14 ccurUEGO_Clear_Driver_Error()

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

```
/*****************************************************************************
   _ccuruego_lib_error_number_t ccurUEGO_Clear_Driver_Error(void *Handle)

   Description: Clear any previously generated driver related error.

   Input:   void *Handle                        (Handle pointer)
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR          (successful)
               # CCURUEGO_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN          (device not open)
               # CCURUEGO_LIB_IOCTL_FAILED      (driver ioctl call failed)
*****************************************************************************/
```

2.2.15 ccurUEGO_Clear_Lib_Error()

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

```
/*****************************************************************************
   _ccuruego_lib_error_number_t ccurUEGO_Clear_Lib_Error(void *Handle)

   Description: Clear any previously generated library related error.

   Input:   void *Handle                        (Handle pointer)
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR          (successful)
               # CCURUEGO_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN          (device not open)
*****************************************************************************
```

2.2.16 ccurUEGO_Close()

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

```
/*****************************************************************************
   _ccuruego_lib_error_number_t ccurUEGO_Close(void *Handle)

   Description: Close a previously opened device.

   Input:   void *Handle                    (Handle pointer)
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR      (successful)
               # CCURUEGO_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN      (device not open)
 *****************************************************************************/
```

2.2.17 ccurUEGO_DAC_Control_Activate()

This is the first call that needs to be issued to DAC to activate the component. Until the DAC has been activated, it will not respond to any commands. The user can also use this call to get the current state of the

DAC without accually changing it by supplying *CCURUEGO_OUPUT_DAC_CONTROL_ENABLE_DO_NOT_CHANGE* to the *activate* argument and specifying a pointer to return the *current_state* argument.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_DAC_Control_Activate (void                                  *Handle,
                           _ccuruego_output_dac_control_enable_t activate,
                           _ccuruego_output_dac_control_enable_t *current_state)

   Description: Activate/DeActivate DAC Control module

   Input:   void                                    *Handle   (Handle pointer)
            _ccuruego_output_dac_control_enable_t activate  (activate/deactivate)
               # CCURUEGO_OUPUT_DAC_CONTROL_DISABLE
               # CCURUEGO_OUPUT_DAC_CONTROL_ENABLE
               # CCURUEGO_OUPUT_DAC_CONTROL_ENABLE_DO_NOT_CHANGE
   Output:  _ccuruego_output_dac_control_enable_t *current_state(active/deactive)
               # CCURUEGO_OUPUT_DAC_CONTROL_DISABLE
               # CCURUEGO_OUPUT_DAC_CONTROL_ENABLE
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR               (successful)
               # CCURUEGO_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN               (device not open)
               # CCURUEGO_LIB_INVALID_ARG            (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION        (local region not present)
*****************************************************************************/
```

2.2.18 ccurUEGO_DAC_Read_Channels()

This call returns the raw and floating-point information for each of the DAC channels.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_DAC_Read_Channels(void                            *Handle,
                           _ccuruego_dac_channel_mask_t     ChanMask,
                           _ccuruego_dac_offset_reference_t Which,
                           ccuruego_dac_volts_t             *dac_volts)

   Description: Read DAC Channels

   Input:   void                              *Handle       (Handle pointer)
            _ccuruego_dac_channel_mask_t    ChanMask      (specify channel mask)
               # CCURUEGO_DAC_CHANNEL_MASK_0
               # CCURUEGO_DAC_CHANNEL_MASK_1
               # CCURUEGO_DAC_CHANNEL_MASK_2
               # CCURUEGO_DAC_CHANNEL_MASK_3
               # CCURUEGO_ALL_DAC_CHANNELS_MASK
            _ccuruego_dac_offset_reference_t Which        (specify which output)
               # CCURUEGO_DAC_OFFSET
               # CCURUEGO_DAC_REFERENCE
            ccuruego_dac_volts_t           *dac_volts (pointer to DAC volts)
                  uint    Raw[CCURUEGO_MAX_DAC_CHANNELS]
                  double  Float[CCURUEGO_MAX_DAC_CHANNELS]
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                 (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION          (local region not present)
               # CCURUEGO_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                 (library not open)
               # CCURUEGO_LIB_INVALID_ARG              (invalid argument)
               # CCURUEGO_LIB_DAC_CONTROL_IS_NOT_ACTIVE(DAC is not active)
*****************************************************************************/
```

2.2.19 ccurUEGO_DAC_Write_Channels()

This call sets the user supplied floating-point values for each of the DAC channels. On successful completion the call returns also returns the raw DAC values to the user.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   _ccurUEGO_DAC_Write_Channels(void                                *Handle,
                               _ccuruego_dac_channel_mask_t     ChanMask,
                               _ccuruego_dac_offset_reference_t Which,
                               ccuruego_dac_volts_t             *dac_volts)

   Description: Write DAC Channels

   Input:   void                              *Handle       (Handle pointer)
            _ccuruego_dac_channel_mask_t    ChanMask      (specify channel mask)
               # CCURUEGO_DAC_CHANNEL_MASK_0
               # CCURUEGO_DAC_CHANNEL_MASK_1
               # CCURUEGO_DAC_CHANNEL_MASK_2
               # CCURUEGO_DAC_CHANNEL_MASK_3
               # CCURUEGO_ALL_DAC_CHANNELS_MASK
            _ccuruego_dac_offset_reference_t Which        (specify which output)
               # CCURUEGO_DAC_OFFSET
               # CCURUEGO_DAC_REFERENCE
            ccuruego_dac_volts_t           *dac_volts (pointer to DAC volts)
                  double  Float[CCURUEGO_MAX_DAC_CHANNELS]
   Output:  ccuruego_dac_volts_t           *dac_volts (pointer to DAC volts)
                  uint    Raw[CCURUEGO_MAX_DAC_CHANNELS]
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR             (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION      (local region not present)
*************************************************************************/
```

2.2.20 ccurUEGO_DataToVolts()

This routine takes a raw analog input data value and converts it to a floating point.

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

   Description: Convert Data to volts

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

2.2.21 ccurUEGO_Digital_Potentiometer_Activate()

This is the first call that needs to be issued to Digital Potentiometer to activate the component. Until the Digital Potentiometer has been activated, it will not respond to any commands. The user can also use this call to get the current state of the Digital Potentiometer without accually changing it by supplying *CCURUEGO_DIGITAL_POTENTIOMETER_CONTROL_ENABLE_DO_NOT_CHANGE* to the *activate* argument and specifying a pointer to return the *current_state* argument.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Digital_Potentiometer_Activate (void                   *Handle,
               _ccuruego_digital_potentiometer_control_enable_t activate,
               _ccuruego_digital_potentiometer_control_enable_t *current_state)


   Description: Activate/DeActivate Digital Potentiomenter module
```

```
   Input:   void                              *Handle         (Handle pointer)
            _ccuruego_digital_potentiometer_control_enable_t
                                              activate        (activate/deactivate)
               # CCURUEGO_DIGITAL_POTENTIOMETER_CONTROL_DISABLE
               # CCURUEGO_DIGITAL_POTENTIOMETER_CONTROL_ENABLE
               # CCURUEGO_DIGITAL_POTENTIOMETER_CONTROL_ENABLE_DO_NOT_CHANGE
   Output:  _ccuruego_digital_potentiometer_control_enable_t
                                              *current_state  (active/deactive)
               # CCURUEGO_DIGITAL_POTENTIOMETER_CONTROL_DISABLE
               # CCURUEGO_DIGITAL_POTENTIOMETER_CONTROL_ENABLE
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                (successful)
               # CCURUEGO_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                (device not open)
               # CCURUEGO_LIB_INVALID_ARG             (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION         (local region not present)
*****************************************************************************/
```

2.2.22 ccurUEGO_Digital_Potentiometer_Get_Resistance()

This call returns both the raw digital value and the corresponding ohm resistance to the user for each of the selected channels.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Digital_Potentiometer_Get_Resistance (void             *Handle,
                       _ccuruego_dac_channel_mask_t                ChanMask,
I                      ccuruego_digital_potentiometer_resistance_t *Resistance)

   Description: Get Potentiometer Resistance

   Input:   void                              *Handle       (Handle pointer)
            _ccuruego_dac_channel_mask_t      ChanMask      (specify channel mask)
               # CCURUEGO_DAC_CHANNEL_MASK_0
               # CCURUEGO_DAC_CHANNEL_MASK_1
               # CCURUEGO_DAC_CHANNEL_MASK_2
               # CCURUEGO_DAC_CHANNEL_MASK_3
               # CCURUEGO_ALL_DAC_CHANNELS_MASK
   Output:  ccuruego_digital_potentiometer_resistance_t
                           Resistance[CCURUEGO_MAX_DAC_CHANNELS]
                                                 (pointer to resistance struct)
                u_int value
                u_int ohms
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURUEGO_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                (library not open)
               # CCURUEGO_LIB_INVALID_ARG             (invalid argument)
               # CCURUEGO_LIB_IO_CONTROL_IS_NOT_ACTIVE (IO is not active)
*****************************************************************************/
```

2.2.23 ccurUEGO_Digital_Potentiometer_Set_Resistance()

This call sets the user supplied resistance for each of the selected channels. The resistance is granular in nature and unless the user specifies a valid resistance, the call will fail. Alternatively, the user can specify the CCURUEGO_DIGITAL_POTENTIOMETER_AUTOCORRECT_OHM_TAG flag by or'ing with the supplied resistance. In this case, the routine will auto-correct the user supplied resistance to the closest valid one and return that value in the *ohms* argument along with the raw potentiometer value.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Digital_Potentiometer_Set_Resistance (void             *Handle,
```

```
                        _ccuruego_dac_channel_mask_t                ChanMask,
                        ccuruego_digital_potentiometer_resistance_t *Resistance)

   Description: Set Digital Potentiometer Resistance

   Input:   void                              *Handle       (Handle pointer)
            _ccuruego_dac_channel_mask_t      ChanMask      (specify channel mask)
               # CCURUEGO_DAC_CHANNEL_MASK_0
               # CCURUEGO_DAC_CHANNEL_MASK_1
               # CCURUEGO_DAC_CHANNEL_MASK_2
               # CCURUEGO_DAC_CHANNEL_MASK_3
               # CCURUEGO_ALL_DAC_CHANNELS_MASK
            ccuruego_digital_potentiometer_resistance_t
                                       Resistance[CCURUEGO_MAX_DAC_CHANNELS]
                                              (pointer to resistance struct)
                    u_int ohms                (ohm resistance)
   Output:  ccuruego_digital_potentiometer_resistance_t
                                       Resistance[CCURUEGO_MAX_DAC_CHANNELS]
                                              (pointer to resistance struct)
               u_int value                    (raw digital value)
               u_int ohms                     (corresponding ohm resistance)
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR        (successful)
               # CCURUEGO_LIB_BAD_HANDLE      (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN        (device not open)
               # CCURUEGO_LIB_INVALID_ARG     (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION (local region not present)
               # CCURUEGO_LIB_DIGITAL_POTENTIOMETER_CONTROL_IS_NOT_ACTIVE
                                          (Digital Potentiometer not active)
*****************************************************************************/
```

2.2.24 ccurUEGO_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.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Disable_Pci_Interrupts (void *Handle)


   Description: Disable interrupts being generated by the board.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR       (successful)
                    # CCURUEGO_LIB_BAD_HANDLE     (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN       (device not open)
                    # CCURUEGO_LIB_IOCTL_FAILED   (driver ioctl call failed)
*****************************************************************************/
```

2.2.25 ccurUEGO_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.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Enable_Pci_Interrupts (void *Handle, uint interrupt_mask)

   Description: Enable interrupts being generated by the board.
```

```
   Input:   void         *Handle            (Handle pointer)
            uint         interrupt_mask     (interrupt mask)
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR      (successful)
               # CCURUEGO_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN      (device not open)
               # CCURUEGO_LIB_IOCTL_FAILED  (driver ioctl call failed)
 ****************************************************************************/
```

2.2.26 ccurUEGO_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.

```
/****************************************************************************
   ccurUEGO_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)
   Oupput:  volatile void *Destination      (pointer to destination buffer)
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR      (successful)
               # CCURUEGO_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN      (device not open)
 ****************************************************************************/
```

2.2.27 ccurUEGO_Fast_Memcpy_Unlocked()

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

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

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

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

2.2.28 ccurUEGO_Fraction_To_Hex()

This converts a fractional decimal to a hexadecimal value.

```
/****************************************************************************
   int
   ccurUEGO_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)
 ****************************************************************************/
```

2.2.29 ccurUEGO_Get_Board_CSR()

This call returns information from the board status register.

```
/****************************************************************************
  _ccuruego_lib_error_number_t
  ccurUEGO_Get_Board_CSR (void                    *Handle,
                          ccuruego_board_csr_t    *bcsr)

   Description: Get Board Control and Status information

   Input:   void                                 *Handle    (Handle pointer)
   Output:  ccuruego_board_csr_t                 *bcsr      (pointer to board csr)
               _ccuruego_bcsr_identify_board_t   identify_board
                  # CCURUEGO_BCSR_IDENTIFY_BOARD_DISABLE
                  # CCURUEGO_BCSR_IDENTIFY_BOARD_ENABLE
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                (successful)
               # CCURUEGO_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                (device not open)
               # CCURUEGO_LIB_INVALID_ARG             (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION         (local region not present)
 ****************************************************************************/
```

2.2.30 ccurUEGO_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 *0x9300* and board type is *0x1*.

```
/****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Board_Info (void                   *Handle,
                            ccuruego_board_info_t *binfo)

   Description: Get Board Information

   Input:       void                               *Handle (handle pointer)
   Output:      ccuruego_board_info_t              *binfo  (pointer to board info)
                   int     board_id                        (board id)
                   int     board_type                      (board type)
                   int     firmware_rev                    (firmware revision)
                   ccuruego_sprom_header_t sprom_header
                      u_int32_t   board_serial_number      (serial number)
                      u_short     sprom_revision           (serial prom revision)
                   int     number_of_channels     (number of hardware channels)
                   int     all_channels_mask               (all channels mask)
                   double  cal_ref_voltage        (calibration reference voltage)
                   double  voltage_range                   (maximum voltage range)
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR                (successful)
                    # CCURUEGO_LIB_BAD_HANDLE              (no/bad handler
supplied)
                    # CCURUEGO_LIB_NOT_OPEN                (device not open)
                    # CCURUEGO_LIB_INVALID_ARG             (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION     (local region not present)
```

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

2.2.31 ccurUEGO_Get_CalibrationBus_Control()

This call returns the bus calibration control.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_CalibrationBus_Control (void                          *Handle,
                                _ccuruego_calibration_bus_control_t *bus_control)


   Description: Get Calibration Bus Control

   Input:       void                                    *Handle (handle pointer)
   Output:      _ccuruego_calibration_bus_control_t
                                    *bus_control  (pointer to control select)
                    # CCURUEGO_CALBUS_CONTROL_OPEN
                    # CCURUEGO_CALBUS_CONTROL_POSITIVE_REF_A
                    # CCURUEGO_CALBUS_CONTROL_POSITIVE_REF_B

                    # CCURUEGO_CALBUS_CONTROL_OFFSET_DAC_0
                    # CCURUEGO_CALBUS_CONTROL_OFFSET_DAC_1
                    # CCURUEGO_CALBUS_CONTROL_OFFSET_DAC_2
                    # CCURUEGO_CALBUS_CONTROL_OFFSET_DAC_3

                    # CCURUEGO_CALBUS_CONTROL_REFERENCE_DAC_0
                    # CCURUEGO_CALBUS_CONTROL_REFERENCE_DAC_1
                    # CCURUEGO_CALBUS_CONTROL_REFERENCE_DAC_2
                    # CCURUEGO_CALBUS_CONTROL_REFERENCE_DAC_3

                    # CCURUEGO_CALBUS_CONTROL_UN_0
                    # CCURUEGO_CALBUS_CONTROL_UN_1
                    # CCURUEGO_CALBUS_CONTROL_UN_2
                    # CCURUEGO_CALBUS_CONTROL_UN_3

                    # CCURUEGO_CALBUS_CONTROL_RESISTANCE_0
                    # CCURUEGO_CALBUS_CONTROL_RESISTANCE_1
                    # CCURUEGO_CALBUS_CONTROL_RESISTANCE_2
                    # CCURUEGO_CALBUS_CONTROL_RESISTANCE_3
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR           (successful)
                    # CCURUEGO_LIB_BAD_HANDLE         (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN           (device not open)
                    # CCURUEGO_LIB_INVALID_ARG        (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION    (local region not present)
*****************************************************************************/
```

2.2.32 ccurUEGO_Get_Driver_Error()

This call returns the last error generated by the driver.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Driver_Error (void                     *Handle,
                              ccuruego_user_error_t    *ret_err)

   Description: Get the last error generated by the driver.

   Input:   void                  *Handle             (Handle pointer)
   Output:  ccuruego_user_error_t *ret_err            (error struct pointer)
               uint error                             (error number)
               char name[CCURUEGO_ERROR_NAME_SIZE]  (error name used in driver)
```

```
                char desc[CCURUEGO_ERROR_DESC_SIZE]  (error description)
   Return:  _ccuruego_lib_error_number_t
                # CCURUEGO_LIB_NO_ERROR               (successful)
                # CCURUEGO_LIB_BAD_HANDLE             (no/bad handler supplied)
                # CCURUEGO_LIB_NOT_OPEN               (device not open)
                # CCURUEGO_LIB_INVALID_ARG            (invalid argument)
                # CCURUEGO_LIB_IOCTL_FAILED           (driver ioctl call failed)
 ***************************************************************************/

#define CCURUEGO_ERROR_NAME_SIZE    64
#define CCURUEGO_ERROR_DESC_SIZE    128

typedef struct _ccuruego_user_error_t
{
    uint error;                             /* error number */
    char name[CCURUEGO_ERROR_NAME_SIZE];    /* error name used in driver */
    char desc[CCURUEGO_ERROR_DESC_SIZE];    /* error description */
} ccuruego_user_error_t;

enum
{
    CCURUEGO_SUCCESS = 0,
    CCURUEGO_INVALID_PARAMETER,
    CCURUEGO_DMA_TIMEOUT,
    CCURUEGO_OPERATION_CANCELLED,
    CCURUEGO_RESOURCE_ALLOCATION_ERROR,
    CCURUEGO_INVALID_REQUEST,
    CCURUEGO_FAULT_ERROR,
    CCURUEGO_BUSY,
    CCURUEGO_ADDRESS_IN_USE,
    CCURUEGO_USER_INTERRUPT_TIMEOUT,
    CCURUEGO_DMA_INCOMPLETE,
    CCURUEGO_DATA_UNDERFLOW,
    CCURUEGO_DATA_OVERFLOW,
    CCURUEGO_IO_FAILURE,
    CCURUEGO_PCI_ABORT_INTERRUPT_ACTIVE,
};
```

2.2.33 ccurUEGO_Get_Driver_Info()

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

```
/***************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Driver_Info (void                    *Handle,
                             ccuruego_driver_info_t *info)

   Description: Get device information from driver.

   Input:       void                            *Handle (handle pointer)

   Output:      ccuruego_driver_info_t          *info   (info struct pointer)
                   char                    version[12]
                   char                    built[32]
                   char                    module_name[16]
                   int                     board_index
                   char                    board_desc[32]
                   int                     bus
                   int                     slot
                   int                     func
                   int                     vendor_id
                   int                     sub_vendor_id
                   int                     board_id
```

```
                    int                   board_type
                    int                   sub_device_id
                    int                   board_info
                    int                   msi_support
                    int                   irqlevel
                    int                   firmware
                    int                   number_of_channels
                    int                   all_channels_mask
                    double                cal_ref_voltage
                    int                   max_dma_samples
                    int                   dma_size
                    double                voltage_range
                    ccuruego_driver_int_t interrupt
                        unsigned long long count
                        u_int status
                        u_int mask
                        int timeout_seconds
                    int                   Ccuruego_Max_Region
                    ccuruego_dev_region_t mem_region[CCURUEGO_MAX_REGION]
                       uint physical_address
                       uint size
                       uint flags
                       uint *virtual_address
                    ccuruego_sprom_header_t sprom_header
                       u_int32_t   board_serial_number
                       u_short     sprom_revision
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR           (successful)
                    # CCURUEGO_LIB_BAD_HANDLE         (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN           (device not open)
                    # CCURUEGO_LIB_INVALID_ARG        (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION    (local region error)
                    # CCURUEGO_LIB_IOCTL_FAILED       (driver ioctl call failed)
*****************************************************************************/
```

2.2.34 ccurUEGO_Get_Driver_Read_Mode()

This call returns the driver read mode.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Driver_Read_Mode (void                        *Handle,
                                  _ccuruego_driver_rw_mode_t *mode)

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

   Input:       void                          *Handle  (handle pointer)
   Output:      _ccuruego_driver_rw_mode_t    *mode    (pointer to read mode)
                    # CCURUEGO_PIO_CHANNEL
                    # CCURUEGO_DMA_CHANNEL
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR           (successful)
                    # CCURUEGO_LIB_BAD_HANDLE         (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN           (device not open)
                    # CCURUEGO_LIB_INVALID_ARG        (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION    (local region error)
                    # CCURUEGO_LIB_IOCTL_FAILED       (driver ioctl call failed)
*****************************************************************************/
```

2.2.35 ccurUEGO_Get_Driver_Write_Mode()

This call returns the driver write mode.

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

```
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Driver_Write_Mode (void                        *Handle,
                                   _ccuruego_driver_rw_mode_t *mode)


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

   Input:       void                          *Handle  (handle pointer)
   Output:      _ccuruego_driver_rw_mode_t    *mode    (pointer to write mode)
                    # CCURUEGO_PIO_CHANNEL
                    # CCURUEGO_DMA_CHANNEL
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR           (successful)
                    # CCURUEGO_LIB_BAD_HANDLE         (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN           (device not open)
                    # CCURUEGO_LIB_INVALID_ARG        (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION    (local region error)
                    # CCURUEGO_LIB_IOCTL_FAILED       (driver ioctl call failed)
*************************************************************************/
```

2.2.36 ccurUEGO_Get_Interrupt_Control()

This call returns the interrupt control information.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Interrupt_Control (void                *Handle,
                                   ccuruego_interrupt_t *intr)

   Description: Get Interrupt Control information

   Input:       void                       *Handle  (handle pointer)
   Output:      ccuruego_interrupt_t       *intr    (pointer to interrupt control)
                   int global_int
                   int plx_local_int
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR        (successful)
                    # CCURUEGO_LIB_BAD_HANDLE      (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN        (device not open)
                    # CCURUEGO_LIB_INVALID_ARG     (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION (local region error)
 *************************************************************************/
```

2.2.37 ccurUEGO_Get_Interrupt_Status()

This call returns the current status of the PLX interrupt.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Interrupt_Status (void                 *Handle,
                                  ccuruego_interrupt_t *intr)

   Description: Get Interrupt Status information

   Input:       void                        *Handle  (handle pointer)
   Output:      ccuruego_interrupt_t        *intr    (pointer to interrupt status)
                   int plx_local_int
                       # CCURUEGO_ISR_LOCAL_PLX_NONE
                       # CCURUEGO_ISR_LOCAL_PLX_OCCURRED
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR         (successful)
                    # CCURUEGO_LIB_BAD_HANDLE       (no/bad handler supplied)
```

```
                    # CCURUEGO_LIB_NOT_OPEN         (device not open)
                    # CCURUEGO_LIB_INVALID_ARG      (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION  (local region error)
 ***************************************************************************/
```

2.2.38 ccurUEGO_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.

```
/***************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_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:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR           (successful)
               # CCURUEGO_LIB_BAD_HANDLE         (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN           (device not open)
               # CCURUEGO_LIB_INVALID_ARG        (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION    (local region not present)
               # CCURUEGO_LIB_IOCTL_FAILED       (ioctl error)
 ***************************************************************************/
```

2.2.39 ccurUEGO_Get_Lib_Error_Description()

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

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

   ccurUEGO_Get_Lib_Error_Description()

   Description: Get Error Description of supplied error number.

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

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

2.2.40 ccurUEGO_Get_Lib_Error()

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

```
/***************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Lib_Error (void                    *Handle,
                           ccuruego_lib_error_t    *lib_error)

   Description: Get last error generated by the library.

   Input:   void                 *Handle                   (Handle pointer)
   Output:  ccuruego_lib_error_t *lib_error                (error struct pointer)
               -- uint error                               (last library error number)
               -- char name[CCURUEGO_LIB_ERROR_NAME_SIZE]  (last library error name)
```

```
                -- char desc[CCURUEGO_LIB_ERROR_DESC_SIZE] (last library error description)
                -- int  line_number                        (last library error line number
                                                            in lib)
                -- char function[CCURUEGO_LIB_ERROR_FUNC_SIZE]
                                                          (library function in error)
                -- ccuruego_lib_error_backtrace_t BT[CCURUEGO_BACK_TRACE_DEPTH]
                                                          (backtrace of errors)
                   -- int  line_number                    (line number in library)
                   -- char function[CCURUEGO_LIB_ERROR_FUNC_SIZE]
                                                          (library function)
   Return:  _ccuruego_lib_error_number_t
                # CCURUEGO_LIB_NO_ERROR                   (successful)
                # CCURUEGO_LIB_BAD_HANDLE                 (no/bad handler supplied)
                # CCURUEGO_LIB_NOT_OPEN                   (device not open)
 ****************************************************************************/

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

typedef struct
{
    uint error;                                   /* last library error number */
    char name[CCURUEGO_LIB_ERROR_NAME_SIZE];      /* last library error name */
    char desc[CCURUEGO_LIB_ERROR_DESC_SIZE];      /* last libarary error description */
    int  line_number;                             /* last library  error line number in
                                                     lib */
    char function[CCURUEGO_LIB_ERROR_FUNC_SIZE]; /* library function in error */
    ccuruego_lib_error_backtrace_t BT[CCURUEGO_BACK_TRACE_DEPTH];
                                                  /* backtrace of errors */
} ccuruego_lib_error_t;
```

Possible library errors:

```
CCURUEGO_LIB_NO_ERROR                   =  0,  /* successful */
CCURUEGO_LIB_INVALID_ARG                = -1,  /* invalid argument */
CCURUEGO_LIB_ALREADY_OPEN               = -2,  /* already open */
CCURUEGO_LIB_OPEN_FAILED                = -3,  /* open failed */
CCURUEGO_LIB_BAD_HANDLE                 = -4,  /* bad handle */
CCURUEGO_LIB_NOT_OPEN                   = -5,  /* device not opened */
CCURUEGO_LIB_MMAP_SELECT_FAILED         = -6,  /* mmap selection failed */
CCURUEGO_LIB_MMAP_FAILED                = -7,  /* mmap failed */
CCURUEGO_LIB_MUNMAP_FAILED              = -8,  /* munmap failed */
CCURUEGO_LIB_NOT_MAPPED                 = -9,  /* not mapped */
CCURUEGO_LIB_ALREADY_MAPPED             = -10, /* already mapped */
CCURUEGO_LIB_IOCTL_FAILED               = -11, /* driver ioctl failed */
CCURUEGO_LIB_IO_ERROR                   = -12, /* i/o error */
CCURUEGO_LIB_INTERNAL_ERROR             = -13, /* internal library error */
CCURUEGO_LIB_NOT_IMPLEMENTED            = -14, /* call not implemented */
CCURUEGO_LIB_LOCK_FAILED                = -15, /* failed to get lib lock */
CCURUEGO_LIB_NO_LOCAL_REGION            = -16, /* local region not present */
CCURUEGO_LIB_NO_CONFIG_REGION           = -17, /* config region not present */
CCURUEGO_LIB_NO_SOLUTION_FOUND          = -18, /* no solution found */
CCURUEGO_LIB_CONVERTER_RESET            = -19, /* converter not active */
CCURUEGO_LIB_NO_RESOURCE                = -20, /* resource not available */
CCURUEGO_LIB_CALIBRATION_RANGE_ERROR    = -21, /* calibration voltage out of range */
CCURUEGO_LIB_CANNOT_OPEN_FILE           = -23, /* cannot open file */
CCURUEGO_LIB_BAD_DATA_IN_CAL_FILE       = -24, /* bad date in calibration file */
CCURUEGO_LIB_CJ135_BUSY                 = -25, /* CJ135 busy */
CCURUEGO_LIB_SERIAL_PROM_BUSY           = -26, /* serial prom busy */
CCURUEGO_LIB_SERIAL_PROM_FAILURE        = -27, /* serial prom failure - malfunction or
```

```
                                                  not present */
CCURUEGO_LIB_INVALID_CRC                = -28, /* invalid CRC read */
CCURUEGO_LIB_SERIAL_PROM_WRITE_PROTECTED = -29, /* serial prom is write protected */
CCURUEGO_LIB_DAC_CONTROL_IS_NOT_ACTIVE   = -30, /* DAC control is not active */
CCURUEGO_LIB_DIGITAL_POTENTIOMETER_CONTROL_IS_NCTIVE
                                        = -31, /* Digital Potentiometer control is not
                                                  active */
CCURUEGO_LIB_IO_CONTROL_IS_NOT_ACTIVE    = -32, /* I/O control is not active */
CCURUEGO_LIB_PWM_CONTROL_IS_NOT_ACTIVE   = -33, /* PWM control is not active */
CCURUEGO_LIB_CJ135_CONTROL_IS_NOT_ACTIVE = -34, /* CJ135 control is not active */
CCURUEGO_LIB_PWM_FREQUENCY_RANGE_EXCEEDED= -35, /* PWM Frequency Range Exceeded */
CCURUEGO_LIB_PWM_DUTY_RANGE_EXCEEDED     = -36, /* PWM Duty Range Exceeded */
```

2.2.41 ccurUEGO_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 *ccuruego_user.h* include file that is supplied with the driver.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Mapped_Config_Ptr (void                              *Handle,
                                   ccuruego_config_local_data_t    **config_ptr)

   Description: Get mapped configuration pointer.

   Input:   void                             *Handle         (Handle pointer)
   Output:  ccuruego_config_local_data_t    **config_ptr    (config struct ptr)
               -- structure in ccuruego_user.h
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR               (successful)
               # CCURUEGO_LIB_BAD_HANDLE             (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN               (device not open)
               # CCURUEGO_LIB_INVALID_ARG            (invalid argument)
               # CCURUEGO_LIB_NO_CONFIG_REGION       (config region not present)
*****************************************************************************/
```

2.2.42 ccurUEGO_Get_Mapped_Driver_Library_Ptr()

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

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Mapped_Driver_Library_Ptr (void                *Handle,
                           ccuruego_driver_library_common_t    **driver_lib_ptr)

   Description: Get mapped Driver/Library structure pointer.

   Input:   void                               *Handle  (Handle pointer)
   Output:  ccuruego_driver_library_common_t  **driver_lib_ptr
                                                (driver_lib struct ptr)
               uint dma_abort_count             (DMA abort count)
               ccuruego_sprom_header_t sprom_header
                  u_int32_t   board_serial_number     (serial number)
                  u_short     sprom_revision          (serial revision)
               uint library_needs_initialization
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR               (successful)
```

```
               # CCURUEGO_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                 (device not open)
               # CCURUEGO_LIB_INVALID_ARG              (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION          (local region not present)
*************************************************************************/
```

2.2.43 ccurUEGO_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 *ccuruego_user.h* include file that is supplied with the driver.

```
/************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Mapped_Local_Ptr (void                          *Handle,
                                  ccuruego_local_ctrl_data_t   **local_ptr)

   Description: Get mapped local pointer.

   Input:   void                         *Handle     (Handle pointer)
   Output:  ccuruego_local_ctrl_data_t  **local_ptr (local struct ptr)
               -- structure in ccuruego_user.h
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR              (successful)
               # CCURUEGO_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN              (device not open)
               # CCURUEGO_LIB_INVALID_ARG           (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION       (local region not present)
 ************************************************************************/
```

2.2.44 ccurUEGO_Get_Open_File_Descriptor()

When the library *ccurUEGO_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.

```
/************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_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:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR        (successful)
               # CCURUEGO_LIB_BAD_HANDLE      (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN        (device not open)
               # CCURUEGO_LIB_INVALID_ARG     (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION (local region not present)

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

2.2.45 ccurUEGO_Get_Physical_Memory()

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

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Physical_Memory (void                *Handle,
                                 ccuruego_phys_mem_t *phys_mem)

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

   Input:       void                *Handle             (handle pointer)
   Output:      ccuruego_phys_mem_t *phys_mem           (mem struct pointer)
                   void *phys_mem
                   uint  phys_mem_size
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION      (local region error)
*****************************************************************************/
```

2.2.46 ccurUEGO_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.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Get_Value (void             *Handle,
                       CCURUEGO_CONTROL cmd,
                       void             *value)

   Description: Return the value of the specified board register.

   Input:       void              *Handle               (handle pointer)
                CCURUEGO_CONTROL  cmd                   (register definition)
                    # CCURUEGO_CONTROL_BOARD_INFORMATION
                    # CCURUEGO_CONTROL_BOARD_CSR
                    # CCURUEGO_CONTROL_INTERRUPT_CONTROL
                    # CCURUEGO_CONTROL_INTERRUPT_STATUS
                    # CCURUEGO_CONTROL_CALIBRATION_BUS_CONTROL
                    # CCURUEGO_CONTROL_FIRMWARE_SPI_COUNTER_STATUS
                    # CCURUEGO_CONTROL_DAC_CONTROL_ENABLE

                    # CCURUEGO_CONTROL_OUTPUT_DATA_OFFSET_DAC
                    # CCURUEGO_CONTROL_OUTPUT_DATA_OFFSET_DAC_0
                    # CCURUEGO_CONTROL_OUTPUT_DATA_OFFSET_DAC_1
                    # CCURUEGO_CONTROL_OUTPUT_DATA_OFFSET_DAC_2
                    # CCURUEGO_CONTROL_OUTPUT_DATA_OFFSET_DAC_3

                    # CCURUEGO_CONTROL_OUTPUT_DATA_REFERENCE_DAC
                    # CCURUEGO_CONTROL_OUTPUT_DATA_REFERENCE_DAC_0
                    # CCURUEGO_CONTROL_OUTPUT_DATA_REFERENCE_DAC_1
                    # CCURUEGO_CONTROL_OUTPUT_DATA_REFERENCE_DAC_2
                    # CCURUEGO_CONTROL_OUTPUT_DATA_REFERENCE_DAC_3

                    # CCURUEGO_CONTROL_GAIN_CALIBRATION_OFFSET_DAC
                    # CCURUEGO_CONTROL_GAIN_CALIBRATION_REFERENCE_DAC
```

```
                    # CCURUEGO_CONTROL_OFFSET_CALIBRATION_OFFSET_DAC
                    # CCURUEGO_CONTROL_OFFSET_CALIBRATION_REFERENCE_DAC

                    # CCURUEGO_CONTROL_DIGITAL_POTENTIOMETER_CONTROL_ENABLE
                    # CCURUEGO_CONTROL_DIGITAL_POTENTIOMETER_VALUE
                    # CCURUEGO_CONTROL_IO_CONTROL_ENABLE
                    # CCURUEGO_CONTROL_IO_CONTROL

                    # CCURUEGO_CONTROL_PWM_CONTROL_ENABLE

                    # CCURUEGO_CONTROL_CJ135_CONTROL_ENABLE
                    # CCURUEGO_CONTROL_CJ135_EXTERNAL_CONTROL
                    # CCURUEGO_CONTROL_CJ135_COMMAND_DATA_STATUS
                    # CCURUEGO_CONTROL_CJ135_INTERRUPT_CONTROL_COUNTER
                    # CCURUEGO_CONTROL_CJ135_RAM_BANK_MULTI_READ

                    # CCURUEGO_CONTROL_SPROM_STAT_ADDR_WRITE_DATA
                    # CCURUEGO_CONTROL_SPROM_READ_DATA

                    # CCURUEGO_CONTROL_SPI_RAM
   Output:      void              *value                (pointer to value)
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION      (local region error)

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

2.2.47 ccurUEGO_Hex_To_Fraction()

This call converts a hexadecimal value to a fractional decimal.

```
/****************************************************************************
   double
   ccurUEGO_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)
 ****************************************************************************/
```

2.2.48 ccurUEGO_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.

```
/****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Identify_Board (void                 *Handle,
                            _ccuruego_identify_t Identify)

   Description: Identify the board by setting the front LED

   Input:   void                 *Handle              (Handle pointer)
            _ccuruego_identify_t Identify             (Identify board settings)
               # CCURUEGO_IDENTIFY_OFF                (turn off flashing)
               # CCURUEGO_IDENTIFY_ON                 (turn on flashing)
               # Number of seconds to flash           (flash for number of seconds)
   Output:  none
```

```
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                 (successful)
               # CCURUEGO_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURUEGO_LIB_NO_LOCAL_REGION          (local region not present)
               # CCURUEGO_LIB_NOT_OPEN                 (device not open)
               # CCURUEGO_LIB_INVALID_ARG              (invalid argument)
*************************************************************************/
```

2.2.49 ccurUEGO_Initialize_Board()

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

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Initialize_Board (void *Handle)

   Description: Initialize the board.

   Input:   void            *Handle             (Handle pointer)
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR          (successful)
               # CCURUEGO_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN          (device not open)
               # CCURUEGO_LIB_IOCTL_FAILED      (driver ioctl call failed)
               # CCURUEGO_LIB_NO_LOCAL_REGION   (local region not present)
 *************************************************************************/
```

2.2.50 ccurUEGO_IO_Control_Activate()

This is the first call that needs to be issued to the IO Control to activate the component. Until the IO Control has been activated, it will not respond to any commands. The user can also use this call to get the current state of the IO Control without accually changing it by supplying *CCURUEGO_IO_CONTROL_ENABLE_DO_NOT_CHANGE* to the *activate* argument and specifying a pointer to return the *current_state* argument.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_IO_Control_Activate (void                          *Handle,
                                 _ccuruego_io_control_enable_t activate,
                                 _ccuruego_io_control_enable_t *current_state)

   Description: Activate/DeActivate I/O Control module

   Input:   void                              *Handle        (Handle pointer)
            _ccuruego_io_control_enable_t     activate       (activate/deactivate)
               # CCURUEGO_IO_CONTROL_DISABLE
               # CCURUEGO_IO_CONTROL_ENABLE
               # CCURUEGO_IO_CONTROL_ENABLE_DO_NOT_CHANGE
   Output:  _ccuruego_io_control_enable_t     *current_state (active/deactive)
               # CCURUEGO_IO_CONTROL_DISABLE
               # CCURUEGO_IO_CONTROL_ENABLE
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                 (successful)
               # CCURUEGO_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                 (device not open)
               # CCURUEGO_LIB_INVALID_ARG              (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION          (local region not present)
*************************************************************************/
```

2.2.51 ccurUEGO_IO_Get_Control()

This call returns the IO Control information for the selected channels.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_IO_Get_Control (void                          *Handle,
                            _ccuruego_dac_channel_mask_t ChanMask,
                            ccuruego_io_select_signal_t  *IoSignal)

   Description: Get IO Control information

   Input:   void                              *Handle       (Handle pointer)
            _ccuruego_dac_channel_mask_t      ChanMask      (specify channel mask)
               # CCURUEGO_DAC_CHANNEL_MASK_0
               # CCURUEGO_DAC_CHANNEL_MASK_1
               # CCURUEGO_DAC_CHANNEL_MASK_2
               # CCURUEGO_DAC_CHANNEL_MASK_3
               # CCURUEGO_ALL_DAC_CHANNELS_MASK
   Output:  ccuruego_io_select_signal_t       IoSignal[CCURUEGO_MAX_DAC_CHANNELS]
                                                      (pointer to io signal struct)
                u_char UN
                   # CCURUEGO_IO_SIGNAL_SELECT_OPEN
                   # CCURUEGO_IO_SIGNAL_SELECT_EXTERNAL
                   # CCURUEGO_IO_SIGNAL_SELECT_CJ135
                   # CCURUEGO_IO_SIGNAL_SELECT_GROUND
                   # CCURUEGO_IO_SIGNAL_SELECT_V_PLUS
                u_char VM
                   # CCURUEGO_IO_SIGNAL_SELECT_OPEN
                   # CCURUEGO_IO_SIGNAL_SELECT_EXTERNAL
                   # CCURUEGO_IO_SIGNAL_SELECT_CJ135
                   # CCURUEGO_IO_SIGNAL_SELECT_GROUND
                   # CCURUEGO_IO_SIGNAL_SELECT_V_PLUS
                u_char IP
                   # CCURUEGO_IO_SIGNAL_SELECT_OPEN
                   # CCURUEGO_IO_SIGNAL_SELECT_EXTERNAL
                   # CCURUEGO_IO_SIGNAL_SELECT_CJ135
                   # CCURUEGO_IO_SIGNAL_SELECT_GROUND
                   # CCURUEGO_IO_SIGNAL_SELECT_V_PLUS
                u_char IA_RL
                   # CCURUEGO_IO_SIGNAL_SELECT_OPEN
                   # CCURUEGO_IO_SIGNAL_SELECT_EXTERNAL
                   # CCURUEGO_IO_SIGNAL_SELECT_CJ135
                   # CCURUEGO_IO_SIGNAL_SELECT_GROUND
                   # CCURUEGO_IO_SIGNAL_SELECT_V_PLUS
                u_int  ALL                          (union of UN, VM, IP, IA_RL)
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                 (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION          (local region not present)
               # CCURUEGO_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                 (library not open)
               # CCURUEGO_LIB_INVALID_ARG              (invalid argument)
               # CCURUEGO_LIB_IO_CONTROL_IS_NOT_ACTIVE (IO is not active)
*************************************************************************/
```

2.2.52 ccurUEGO_IO_Set_Control()

This call sets the IO Control for the selected channels.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_IO_Set_Control (void                          *Handle,
                            _ccuruego_dac_channel_mask_t ChanMask,
```

```
                            ccuruego_io_select_signal_t  *IoSignal)

   Description: Set IO Control information

   Input:   void                              *Handle       (Handle pointer)
            _ccuruego_dac_channel_mask_t      ChanMask      (specify channel mask)
               # CCURUEGO_DAC_CHANNEL_MASK_0
               # CCURUEGO_DAC_CHANNEL_MASK_1
               # CCURUEGO_DAC_CHANNEL_MASK_2
               # CCURUEGO_DAC_CHANNEL_MASK_3
               # CCURUEGO_ALL_DAC_CHANNELS_MASK
            ccuruego_io_select_signal_t       IoSignal[CCURUEGO_MAX_DAC_CHANNELS]
                                                      (pointer to io signal struct)
                u_char UN
                   # CCURUEGO_IO_SIGNAL_SELECT_OPEN
                   # CCURUEGO_IO_SIGNAL_SELECT_EXTERNAL
                   # CCURUEGO_IO_SIGNAL_SELECT_CJ135
                   # CCURUEGO_IO_SIGNAL_SELECT_GROUND
                   # CCURUEGO_IO_SIGNAL_SELECT_V_PLUS
                u_char VM
                   # CCURUEGO_IO_SIGNAL_SELECT_OPEN
                   # CCURUEGO_IO_SIGNAL_SELECT_EXTERNAL
                   # CCURUEGO_IO_SIGNAL_SELECT_CJ135
                   # CCURUEGO_IO_SIGNAL_SELECT_GROUND
                   # CCURUEGO_IO_SIGNAL_SELECT_V_PLUS
                u_char IP
                   # CCURUEGO_IO_SIGNAL_SELECT_OPEN
                   # CCURUEGO_IO_SIGNAL_SELECT_EXTERNAL
                   # CCURUEGO_IO_SIGNAL_SELECT_CJ135
                   # CCURUEGO_IO_SIGNAL_SELECT_GROUND
                   # CCURUEGO_IO_SIGNAL_SELECT_V_PLUS
                u_char IA_RL
                   # CCURUEGO_IO_SIGNAL_SELECT_OPEN
                   # CCURUEGO_IO_SIGNAL_SELECT_EXTERNAL
                   # CCURUEGO_IO_SIGNAL_SELECT_CJ135
                   # CCURUEGO_IO_SIGNAL_SELECT_GROUND
                   # CCURUEGO_IO_SIGNAL_SELECT_V_PLUS
                u_int  ALL                        (union of UN, VM, IP, IA_RL)
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURUEGO_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                (library not open)
               # CCURUEGO_LIB_INVALID_ARG             (invalid argument)
               # CCURUEGO_LIB_IO_CONTROL_IS_NOT_ACTIVE (IO is not active)

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

2.2.53 ccurUEGO_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 *ccurUEGO_Munmap_Physical_Memory()* API call to remove the previously allocated physical memory.

```
/***************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_MMap_Physical_Memory (void *Handle,
                                  int  size,
                                  void **mem_ptr)

   Description: Allocate a physical DMA memory for size bytes.
```

```
   Input:       void        *Handle                     (handle pointer)
                int         size                        (size in bytes)
   Output:      void        **mem_ptr                   (mapped memory pointer)
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_MMAP_SELECT_FAILED   (mmap selection failed)
                    # CCURUEGO_LIB_MMAP_FAILED          (mmap failed)
*************************************************************************/
```

2.2.54 ccurUEGO_Munmap_Physical_Memory()

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

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Munmap_Physical_Memory (void *Handle)


   Description: Unmap a previously mapped physical DMA memory.

   Input:       void        *Handle                     (handle pointer)
   Output:      None
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_MMAP_SELECT_FAILED   (mmap selection failed)
                    # CCURUEGO_LIB_MMAP_FAILED          (mmap failed)
*************************************************************************/
```

2.2.55 ccurUEGO_NanoDelay()

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

```
/*************************************************************************
   void
   ccurUEGO_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
 *************************************************************************/
```

2.2.56 ccurUEGO_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/ccuruego<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.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_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:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR              (successful)
               # CCURUEGO_LIB_INVALID_ARG           (invalid argument)
               # CCURUEGO_LIB_ALREADY_OPEN          (device already opened)
               # CCURUEGO_LIB_OPEN_FAILED           (device open failed)
               # CCURUEGO_LIB_ALREADY_MAPPED        (memory already mmapped)
               # CCURUEGO_LIB_MMAP_SELECT_FAILED    (mmap selection failed)
               # CCURUEGO_LIB_MMAP_FAILED           (mmap failed)
 *****************************************************************************/
```

2.2.57 ccurUEGO_PWM_CalcDutyCycle()

This routine returns the duty cycle for a given period width and period low count.

```
/*****************************************************************************
   double
   ccurUEGO_PWM_CalcDutyCycle(u_int32_t period_width_clock_count,
                              u_int32_t period_low_clock_count)

   Description: Calculate Duty Cycle in percent

   Input:       u_int32_t period_width_clock_count    (period width clock count)
                u_int32_t period_low_clock_count      (period low clock count)
   Output:      None
   Return:      double                                (Duty cycle)
 *****************************************************************************/
```

2.2.58 ccurUEGO_PWM_CalcFreqinHz()

Calculate frequency in Hetz for supplied period width.

```
/*****************************************************************************
   double
   ccurUEGO_PWM_CalcFreqinHz(u_int32_t period_width_clock_count)

   Description: Calculate Frequency in Hertz

   Input:       u_int32_t period_width_clock_count  (period width clock count)
   Output:      None
   Return:      double                              (Frequency in Hertz)
```

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

2.2.59 ccurUEGO_PWM_CalcPeriodinUsec()

Calculate period in micro-seconds for supplied width.

```
/****************************************************************************
   double
   ccurUEGO_PWM_CalcPeriodinUsec(u_int32_t period_width_clock_count)

   Description: Calculate Period in micro-seconds

   Input:       u_int32_t period_width_clock_count  (period width clock count)
   Output:      None
   Return:      double                              (Period in micro-seconds)
 ****************************************************************************/
```

2.2.60 ccurUEGO_PWM_Control_Activate()

This is the first call that needs to be issued to the PWM Control to activate the component. Until the PWM Control has been activated, it will not respond to any commands. The user can also use this call to get the current state of the PWM Control without accually changing it by supplying *CCURUEGO_PWM_CONTROL_ENABLE_DO_NOT_CHANGE* to the *activate* argument and specifying a pointer to return the *current_state* argument.

```
/****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_PWM_Control_Activate (void                            *Handle,
                                  _ccuruego_pwm_control_enable_t activate,
                                  _ccuruego_pwm_control_enable_t *current_state)

   Description: Activate/DeActivate PWM Control module

   Input:   void                            *Handle        (Handle pointer)
            _ccuruego_pwm_control_enable_t  activate       (activate/deactivate)
               # CCURUEGO_PWM_CONTROL_DISABLE
               # CCURUEGO_PWM_CONTROL_ENABLE
               # CCURUEGO_PWM_CONTROL_ENABLE_DO_NOT_CHANGE
   Output:  _ccuruego_pwm_control_enable_t  *current_state (active/deactive)
               # CCURUEGO_PWM_CONTROL_DISABLE
               # CCURUEGO_PWM_CONTROL_ENABLE
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                (successful)
               # CCURUEGO_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                (device not open)
               # CCURUEGO_LIB_INVALID_ARG             (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION         (local region not present)

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

2.2.61 ccurUEGO_PWM_Get_Count()

Get PWM low count and width count for selected channels.

```
/****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_PWM_Get_Count (void                              *Handle,
                           _ccuruego_dac_channel_mask_t      ChanMask,
                           ccuruego_pwm_low_width_count_t    *Count)

   Description: Get PWM Low and Width Counts

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

```
            _ccuruego_dac_channel_mask_t    ChanMask      (specify channel mask)
               # CCURUEGO_DAC_CHANNEL_MASK_0
               # CCURUEGO_DAC_CHANNEL_MASK_1
               # CCURUEGO_DAC_CHANNEL_MASK_2
               # CCURUEGO_DAC_CHANNEL_MASK_3
               # CCURUEGO_ALL_DAC_CHANNELS_MASK
   Output:  ccuruego_pwm_low_width_count_t  Count[CCURUEGO_MAX_DAC_CHANNELS]
                                            (pointer to count struct)
                u_int32_t Low               (low count)
                u_int32_t Width              (width count)
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR              (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION       (local region not present)
               # CCURUEGO_LIB_BAD_HANDLE            (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN              (library not open)
               # CCURUEGO_LIB_INVALID_ARG           (invalid argument)
               # CCURUEGO_LIB_PWM_CONTROL_IS_NOT_ACTIVE   (PWM is not active)
 ****************************************************************************/
```

2.2.62 ccurUEGO_PWM_Get_Diagnostic_Count()

Get PWM diagnostic low count and width count.

```
/****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_PWM_Get_Diagnostic_Count (void                          *Handle,
                         ccuruego_pwm_diagnostic_low_width_count_t *DiagCount)

   Description: Get PWM Diagnostic Low and Width Count

   Input:   void                                  *Handle       (Handle pointer)
   Output:  ccuruego_pwm_diagnostic_low_width_count_t  *DiagCount
               u_int32_t Low                           (low count)
               u_int32_t Width                         (width count)
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                 (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION          (local region not present)
               # CCURUEGO_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                 (library not open)
               # CCURUEGO_LIB_INVALID_ARG              (invalid argument)
               # CCURUEGO_LIB_PWM_CONTROL_IS_NOT_ACTIVE   (PWM is not active)
 ****************************************************************************/
```

2.2.63 ccurUEGO_PWM_Get_Diagnostic_Frequency_Duty()

Get PWM diagnostic frequency and duty cycle.

```
/****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_PWM_Get_Diagnostic_Frequency_Duty (void                *Handle,
                         ccuruego_pwm_diagnostic_frequency_duty_t *DiagFreqDuty)

   Description: Get PWM Diagnostic Frequency & Duty

   Input:   void                                *Handle       (Handle pointer)
   Output:  ccuruego_pwm_diagnostic_frequency_duty_t  *DiagFreqDuty
               double Frequency                        (frequency)
               double Duty                             (duty)
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                 (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION          (local region not present)
               # CCURUEGO_LIB_BAD_HANDLE               (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                 (library not open)
```

```
               # CCURUEGO_LIB_INVALID_ARG              (invalid argument)
               # CCURUEGO_LIB_PWM_CONTROL_IS_NOT_ACTIVE   (PWM is not active)
 ***************************************************************************/
```

2.2.64 ccurUEGO_PWM_Get_Diagnostic_Status()

Get PWM diagnostic and status information

```
/***************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_PWM_Get_Diagnostic_Status (void                          *Handle,
                                ccuruego_pwm_diagnostic_status_t    *DiagStatus)

   Description: Get PWM Diagnostic and Status Information

   Input:   void                              *Handle       (Handle pointer)
   Output:  ccuruego_pwm_diagnostic_status_t *DiagStatus
               _ccuruego_diagstat_diagnostic_signal_t
                            diagnostic_enable_ch[CCURUEGO_MAX_DAC_CHANNELS]
               _ccuruego_diagstat_volage_detect_t
                            battery_voltage_detect_ch01;
               _ccuruego_diagstat_volage_detect_t
                            battery_voltage_detect_ch23;
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURUEGO_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                (library not open)
               # CCURUEGO_LIB_INVALID_ARG             (invalid argument)
               # CCURUEGO_LIB_PWM_CONTROL_IS_NOT_ACTIVE   (PWM is not active)
 ***************************************************************************/
```

2.2.65 ccurUEGO_PWM_Set_Diagnostic_Count()

Set PWM diagnostics low and width counts. You can use the *ccurUEGO_PWM_Set_Diagnostic_frequency_Duty()* routine instead of this routine.

```
/***************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_PWM_Set_Diagnostic_Count (void                            *Handle,
                           ccuruego_pwm_diagnostic_low_width_count_t *DiagCount)

   Description: Set PWM Diagnostic Low and Width Count

   Input:   void                                      *Handle   (Handle pointer)
            ccuruego_pwm_diagnostic_low_width_count_t *DiagCount
               u_int32_t Low                          (low count)
               u_int32_t Width                        (width count)
   Output:  None
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURUEGO_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                (library not open)
               # CCURUEGO_LIB_INVALID_ARG             (invalid argument)
               # CCURUEGO_LIB_PWM_CONTROL_IS_NOT_ACTIVE   (PWM is not active)
 ***************************************************************************/
```

2.2.66 ccurUEGO_PWM_Set_Diagnostic_Frequency_Duty()

Set PWM diagnostics frequency and duty. You can use the *ccurUEGO_PWM_Set_Diagnostic_Count()* routine instead of this routine.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_PWM_Set_Diagnostic_Frequency_Duty (void                 *Handle,

                         ccuruego_pwm_diagnostic_frequency_duty_t *DiagFreqDuty)

   Description: Set PWM Diagnostic Frequency and Duty

   Input:   void                                    *Handle       (Handle pointer)
            ccuruego_pwm_diagnostic_frequency_duty_t  *DiagFreqDuty
               double Frequency                       (frequency)
               double Duty                            (duty)
   Output:  ccuruego_pwm_diagnostic_frequency_duty_t  *DiagFreqDuty
               double ActualFrequency                 (Actual frequency)
               double ActualDuty                      (Actual duty)
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                (no error)
               # CCURUEGO_LIB_NO_LOCAL_REGION         (local region not present)
               # CCURUEGO_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                (library not open)
               # CCURUEGO_LIB_INVALID_ARG             (invalid argument)
               # CCURUEGO_LIB_PWM_CONTROL_IS_NOT_ACTIVE   (PWM is not active)
 *****************************************************************************/
```

2.2.67 ccurUEGO_Read()

This call performs a programmed I/O driver read of the offset DAC registers. Prior to issuing this call, the user needs to set up the desired read mode of operation using the *ccurUEGO_Select_Driver_Read_Mode()* with *CCURUEGO_PIO_CHANNEL* or *CCURUEGO_DMA_CHANNEL* argument. Read size are limited to *CCURUEGO_MAX_DAC_CHANNELS* 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.

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.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_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:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR      (successful)
               # CCURUEGO_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN      (device not open)
               # CCURUEGO_LIB_IO_ERROR      (read failed)
 *****************************************************************************/
```

2.2.68 ccurUEGO_Read_Serial_Prom()

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

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Read_Serial_Prom(void                *Handle,
                             ccuruego_sprom_rw_t *spr)

   Description: Read Serial Prom for specified number of words

   Input:       void                    *Handle          (handle pointer)
                ccuruego_sprom_rw_t     *spr             (pointer to struct)
                   u_short word_offset
                   u_short num_words
   Output:      ccuruego_sprom_rw_t     *spr             (pointer to struct)
                   u_short *data_ptr
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION      (local region error)
                    # CCURUEGO_LIB_SERIAL_PROM_BUSY     (serial prom busy)
                    # CCURUEGO_LIB_SERIAL_PROM_FAILURE  (serial prom failure)

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

2.2.69 ccurUEGO_Read_Serial_Prom_Item()

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

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Read_Serial_Prom_Item(void                      *Handle,
                                  _ccuruego_sprom_access_t item,
                                  void                      *item_ptr)

   Description: Read Serial Prom for specified item

   Input:       void                    *Handle          (handle pointer)
                _ccuruego_sprom_access_t item            (select item)
                   CCURUEGO_SPROM_HEADER
   Output:      ccuruego_sprom_header_t sprom_header     (pinter to item struct)
                   u_int32_t   board_serial_number
                   u_short     sprom_revision
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION      (local region error)
                    # CCURUEGO_LIB_SERIAL_PROM_BUSY     (serial prom busy)
                    # CCURUEGO_LIB_SERIAL_PROM_FAILURE  (serial prom failure)
*************************************************************************/
```

2.2.70 ccurUEGO_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.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_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:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR      (successful)
               # CCURUEGO_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN      (device not open)
               # CCURUEGO_LIB_IOCTL_FAILED  (driver ioctl call failed)
 *****************************************************************************/
```

2.2.71 ccurUEGO_Reset_Board()

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

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Reset_Board (void *Handle)

   Description: Reset the board.

   Input:   void *Handle                      (Handle pointer)
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR        (successful)
               # CCURUEGO_LIB_BAD_HANDLE      (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN        (device not open)
               # CCURUEGO_LIB_IOCTL_FAILED    (driver ioctl call failed)
               # CCURUEGO_LIB_NO_LOCAL_REGION (local region not present)
 *****************************************************************************/
```

2.2.72 ccurUEGO_Select_Driver_Read_Mode()

This call can be used to select the driver read mode.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Select_Driver_Read_Mode (void                       *Handle,
                                     _ccuruego_driver_rw_mode_t mode)

   Description: Select Driver Read Mode

   Input:       void                       *Handle      (handle pointer)
                _ccuruego_driver_rw_mode_t mode         (select read mode)
                    # CCURUEGO_PIO_CHANNEL
                    # CCURUEGO_DMA_CHANNEL
   Output:      none
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
```

```
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION      (local region error)
*************************************************************************/
```

2.2.73 ccurUEGO_Select_Driver_Write_Mode()

This call can be used to select the driver write mode.

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Select_Driver_Write_Mode (void                        *Handle,
                                      _ccuruego_driver_rw_mode_t mode)

   Description: Select Driver Write Mode

   Input:       void                         *Handle      (handle pointer)
                _ccuruego_driver_rw_mode_t mode          (select write mode)
                    # CCURUEGO_PIO_CHANNEL
                    # CCURUEGO_DMA_CHANNEL
   Output:      none
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION      (local region error)
*************************************************************************/
```

2.2.74 ccurUEGO_Serial_Prom_Write_Override()

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

- ccurUEGO_Write_Serial_Prom()
- ccurUEGO_Write_Serial_Prom_Item()

```
/*************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Serial_Prom_Write_Override (void *Handle,
                                        int  action)

   Description: Set Serial Prom Write Override

   Input:       void                *Handle             (handle pointer)
                _ccuruego_bool_t    action              (override action)
                    # CCURUEGO_TRUE
                    # CCURUEGO_FALSE
   Output:      none
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION      (local region error)
*************************************************************************/
```

2.2.75 ccurUEGO_Set_Board_CSR()

This call sets the board control register.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Set_Board_CSR (void                    *Handle,
                           ccuruego_board_csr_t    *bcsr)

   Description: Set Board Control and Status information

   Input:   void                                *Handle (Handle pointer)
            ccuruego_board_csr_t                *bcsr   (pointer to board csr)
               _ccuruego_bcsr_identify_board_t identify_board
                  # CCURUEGO_BCSR_IDENTIFY_BOARD_DISABLE
                  # CCURUEGO_BCSR_IDENTIFY_BOARD_ENABLE
                  # CCURUEGO_BCSR_IDENTIFY_BOARD_ENABLE_DO_NOT_CHANGE
   Output:  none
   Return:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR                (successful)
               # CCURUEGO_LIB_BAD_HANDLE              (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN                (device not open)
               # CCURUEGO_LIB_INVALID_ARG             (invalid argument)
               # CCURUEGO_LIB_NO_LOCAL_REGION         (local region not present)
 *****************************************************************************/
```

2.2.76 ccurUEGO_Set_CalibrationBus_Control()

This call sets the calibration bus control.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Set_CalibrationBus_Control (void                          *Handle,
                                _ccuruego_calibration_bus_control_t bus_control)

   Description: Set Calibration Bus Control

   Input:       void                *Handle             (handle pointer)
                _ccuruego_calibration_bus_control_t
                                    bus_control         (control set)
                    # CCURUEGO_CALBUS_CONTROL_OPEN
                    # CCURUEGO_CALBUS_CONTROL_POSITIVE_REF_A
                    # CCURUEGO_CALBUS_CONTROL_POSITIVE_REF_B

                    # CCURUEGO_CALBUS_CONTROL_OFFSET_DAC_0
                    # CCURUEGO_CALBUS_CONTROL_OFFSET_DAC_1
                    # CCURUEGO_CALBUS_CONTROL_OFFSET_DAC_2
                    # CCURUEGO_CALBUS_CONTROL_OFFSET_DAC_3

                    # CCURUEGO_CALBUS_CONTROL_REFERENCE_DAC_0
                    # CCURUEGO_CALBUS_CONTROL_REFERENCE_DAC_1
                    # CCURUEGO_CALBUS_CONTROL_REFERENCE_DAC_2
                    # CCURUEGO_CALBUS_CONTROL_REFERENCE_DAC_3

                    # CCURUEGO_CALBUS_CONTROL_UN_0
                    # CCURUEGO_CALBUS_CONTROL_UN_1
                    # CCURUEGO_CALBUS_CONTROL_UN_2
                    # CCURUEGO_CALBUS_CONTROL_UN_3

                    # CCURUEGO_CALBUS_CONTROL_RESISTANCE_0
                    # CCURUEGO_CALBUS_CONTROL_RESISTANCE_1
                    # CCURUEGO_CALBUS_CONTROL_RESISTANCE_2
                    # CCURUEGO_CALBUS_CONTROL_RESISTANCE_3
```

```
   Output:      none
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION      (local region error)
*****************************************************************************/
```

2.2.77 ccurUEGO_Set_Interrupt_Control()

This call sets the interrupt control.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Set_Interrupt_Control (void                 *Handle,
                                   ccuruego_interrupt_t *intr)

   Description: Set Interrupt Control information

   Input:       void                    *Handle     (handle pointer)
                ccuruego_interrupt_t    *intr       (pointer to interrupt control)
                   int global_int
                       # CCURUEGO_ICSR_GLOBAL_DISABLE
                       # CCURUEGO_ICSR_GLOBAL_ENABLE
                       # CCURUEGO_DO_NOT_CHANGE
                   int plx_local_int
                       # CCURUEGO_ICSR_LOCAL_PLX_DISABLE
                       # CCURUEGO_ICSR_LOCAL_PLX_ENABLE
                       # CCURUEGO_DO_NOT_CHANGE
   Output:      none
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR          (successful)
                    # CCURUEGO_LIB_BAD_HANDLE        (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN          (device not open)
                    # CCURUEGO_LIB_INVALID_ARG       (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION   (local region error)
 *****************************************************************************/
```

2.2.78 ccurUEGO_Set_Interrupt_Status()

This call sets/clears the PLX interrupt.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Set_Interrupt_Status (void                 *Handle,
                                  ccuruego_interrupt_t *intr)

   Description: Set Interrupt Status information

   Input:       void                    *Handle      (handle pointer)
                ccuruego_interrupt_t    *intr        (pointer to interrupt status)
                   int plx_local_int
                       # CCURUEGO_INTSTAT_LOCAL_PLX_NONE
                       # CCURUEGO_INTSTAT_LOCAL_PLX_RESET
                       # CCURUEGO_DO_NOT_CHANGE
   Output:      none
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR         (successful)
                    # CCURUEGO_LIB_BAD_HANDLE       (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN         (device not open)
                    # CCURUEGO_LIB_INVALID_ARG      (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION  (local region error)
```

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

2.2.79 ccurUEGO_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.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_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:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR      (successful)
               # CCURUEGO_LIB_BAD_HANDLE    (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN      (device not open)
               # CCURUEGO_LIB_INVALID_ARG   (invalid argument)
 *****************************************************************************/
```

2.2.80 ccurUEGO_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.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Set_Value (void             *Handle,
                       CCURUEGO_CONTROL cmd,
                       void             *value)

   Description: Set the value of the specified board register.

   Input:       void             *Handle                (handle pointer)
                CCURUEGO_CONTROL cmd                    (register definition)
                    # CCURUEGO_CONTROL_BOARD_INFORMATION
                    # CCURUEGO_CONTROL_BOARD_CSR
                    # CCURUEGO_CONTROL_INTERRUPT_CONTROL
                    # CCURUEGO_CONTROL_INTERRUPT_STATUS
                    # CCURUEGO_CONTROL_CALIBRATION_BUS_CONTROL
                    # CCURUEGO_CONTROL_FIRMWARE_SPI_COUNTER_STATUS
                    # CCURUEGO_CONTROL_DAC_CONTROL_ENABLE

                    # CCURUEGO_CONTROL_OUTPUT_DATA_OFFSET_DAC
                    # CCURUEGO_CONTROL_OUTPUT_DATA_OFFSET_DAC_0
                    # CCURUEGO_CONTROL_OUTPUT_DATA_OFFSET_DAC_1
                    # CCURUEGO_CONTROL_OUTPUT_DATA_OFFSET_DAC_2
                    # CCURUEGO_CONTROL_OUTPUT_DATA_OFFSET_DAC_3

                    # CCURUEGO_CONTROL_OUTPUT_DATA_REFERENCE_DAC
                    # CCURUEGO_CONTROL_OUTPUT_DATA_REFERENCE_DAC_0
                    # CCURUEGO_CONTROL_OUTPUT_DATA_REFERENCE_DAC_1
```

```
                    # CCURUEGO_CONTROL_OUTPUT_DATA_REFERENCE_DAC_2
                    # CCURUEGO_CONTROL_OUTPUT_DATA_REFERENCE_DAC_3

                    # CCURUEGO_CONTROL_GAIN_CALIBRATION_OFFSET_DAC
                    # CCURUEGO_CONTROL_GAIN_CALIBRATION_REFERENCE_DAC

                    # CCURUEGO_CONTROL_OFFSET_CALIBRATION_OFFSET_DAC
                    # CCURUEGO_CONTROL_OFFSET_CALIBRATION_REFERENCE_DAC

                    # CCURUEGO_CONTROL_DIGITAL_POTENTIOMETER_CONTROL_ENABLE
                    # CCURUEGO_CONTROL_DIGITAL_POTENTIOMETER_VALUE
                    # CCURUEGO_CONTROL_IO_CONTROL_ENABLE
                    # CCURUEGO_CONTROL_IO_CONTROL

                    # CCURUEGO_CONTROL_PWM_CONTROL_ENABLE

                    # CCURUEGO_CONTROL_CJ135_CONTROL_ENABLE
                    # CCURUEGO_CONTROL_CJ135_EXTERNAL_CONTROL
                    # CCURUEGO_CONTROL_CJ135_COMMAND_DATA_STATUS
                    # CCURUEGO_CONTROL_CJ135_INTERRUPT_CONTROL_COUNTER
                    # CCURUEGO_CONTROL_CJ135_RAM_BANK_MULTI_READ

                    # CCURUEGO_CONTROL_SPROM_STAT_ADDR_WRITE_DATA
                    # CCURUEGO_CONTROL_SPROM_READ_DATA

                    # CCURUEGO_CONTROL_SPI_RAM
                void            *value              (pointer to value to be set)
   Output:      None
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR         (successful)
                    # CCURUEGO_LIB_BAD_HANDLE       (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN         (device not open)
                    # CCURUEGO_LIB_INVALID_ARG      (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION  (local region error)

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

2.2.81 ccurUEGO_VoltsToData()

This call converts user supplied volts to raw data.

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

   Description: Convert Volts to data

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

2.2.82 ccurUEGO_VoltsToDataChanCal()

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

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

   Description: Convert Volts to Data (for Channel Calibration)

   Input:       double   volts                  (volts to convert)
```

```
   Output:      none
   Return:      uint     data                    (returned data)
*****************************************************************************/
```

2.2.83 ccurUEGO_Wait_For_Interrupt()

This call is made available to advanced users to bypass the API and perform their own interrupt handling. 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.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Wait_For_Interrupt (void                  *Handle,
                                ccuruego_driver_int_t *drv_int)

   Description: Wait For Interrupt

   Input:       void                   *Handle        (handle pointer)
   Output:      ccuruego_driver_int_t  *drv_int       (pointer to drv_int struct)
                   unsigned long long count
                   u_int status
                   u_int mask
                        # CCURUEGO_INTSTAT_LOCAL_PLX_MASK
                   int timeout_seconds
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR           (successful)
                    # CCURUEGO_LIB_BAD_HANDLE         (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN           (device not open)
                    # CCURUEGO_LIB_INVALID_ARG        (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION    (local region error)
*****************************************************************************/
```

2.2.84 ccurUEGO_Write()

This call performs a programmed I/O driver write of the offset DAC registers. Prior to issuing this call, the user needs to set up the desired write mode of operation using the *ccurUEGO_Select_Driver_Write_Mode()* with *CCURUEGO_PIO_CHANNEL* or *CCURUEGO_DMA_CHANNEL* argument. Write size are limited to *CCURUEGO_MAX_DAC_CHANNELS* words.

```
/*****************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_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:  _ccuruego_lib_error_number_t
               # CCURUEGO_LIB_NO_ERROR          (successful)
               # CCURUEGO_LIB_BAD_HANDLE        (no/bad handler supplied)
               # CCURUEGO_LIB_NOT_OPEN          (device not open)
               # CCURUEGO_LIB_IO_ERROR          (write failed)
*****************************************************************************/
```

2.2.85 ccurUEGO_Write_Serial_Prom()

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

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

```
/************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Write_Serial_Prom(void                *Handle,
                              ccuruego_sprom_rw_t *spw)

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

   Input:       void                   *Handle          (handle pointer)
                ccuruego_sprom_rw_t    *spw             (pointer to struct)
                   # u_short word_offset
                   # u_short num_words
                   # u_short *data_ptr
   Output:      none
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION      (local region error)
                    # CCURUEGO_LIB_SERIAL_PROM_BUSY     (serial prom busy)
                    # CCURUEGO_LIB_SERIAL_PROM_FAILURE  (serial prom failure)
************************************************************************/
```

2.2.86 ccurUEGO_Write_Serial_Prom_Item()

This call is used to write well defined sections in the serial prom. The user supplies the serial prom section that needs to be written and the data points to the section specific structure. This call should normally not be used by the user.

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

```
/************************************************************************
   _ccuruego_lib_error_number_t
   ccurUEGO_Write_Serial_Prom_Item(void                       *Handle,
                                   _ccuruego_sprom_access_t item,
                                   void                       *item_ptr)

   Description: Write Serial Prom with specified item

   Input:       void                   *Handle          (handle pointer)
                _ccuruego_sprom_access_t item           (select item)
                    # CCURUEGO_SPROM_HEADER
   Output:      ccuruego_sprom_header_t sprom_header    (pinter to item struct)
                   u_int32_t   board_serial_number
                   u_short     sprom_revision
   Return:      _ccuruego_lib_error_number_t
                    # CCURUEGO_LIB_NO_ERROR             (successful)
                    # CCURUEGO_LIB_BAD_HANDLE           (no/bad handler supplied)
                    # CCURUEGO_LIB_NOT_OPEN             (device not open)
                    # CCURUEGO_LIB_INVALID_ARG          (invalid argument)
                    # CCURUEGO_LIB_NO_LOCAL_REGION      (local region error)
                    # CCURUEGO_LIB_SERIAL_PROM_BUSY     (serial prom busy)
                    # CCURUEGO_LIB_SERIAL_PROM_FAILURE  (serial prom failure)
```

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

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 ccuruego_dump

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

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

Example display:

./ccuruego_dump

```
Device Name    : /dev/ccuruego0
Board Serial No: 672340 (0x000a4254)

LOCAL Register 0x7ffff7ff6000 Offset=0x0
CONFIG Register 0x7ffff7ff5000 Offset=0x0

======= LOCAL BOARD REGISTERS =========
LBR: @0x0000 --> 0x93000102
LBR: @0x0004 --> 0x00000000
LBR: @0x0008 --> 0x00000000
LBR: @0x000c --> 0x00000000
LBR: @0x0010 --> 0x00000000
LBR: @0x0014 --> 0x00000001
LBR: @0x0018 --> 0x00000001
LBR: @0x001c --> 0x00000001
LBR: @0x0020 --> 0x00000001
LBR: @0x0024 --> 0x00000001
LBR: @0x0028 --> 0x00000001
LBR: @0x002c --> 0x00000001
LBR: @0x0030 --> 0x00000001
.
.
.
LBR: @0x07c0 --> 0x00000000
LBR: @0x07c4 --> 0x00000000
LBR: @0x07c8 --> 0x00000000
LBR: @0x07cc --> 0x00000000
LBR: @0x07d0 --> 0x00000000
LBR: @0x07d4 --> 0x00000000
LBR: @0x07d8 --> 0x00000000
LBR: @0x07dc --> 0x00000000
LBR: @0x07e0 --> 0x00000000
LBR: @0x07e4 --> 0x00000000
LBR: @0x07e8 --> 0x00000000
LBR: @0x07ec --> 0x00000000
LBR: @0x07f0 --> 0x00000000
```

```
LBR: @0x07f4 --> 0x00000000
LBR: @0x07f8 --> 0x00000000
LBR: @0x07fc --> 0x00000000

======= LOCAL CONFIG REGISTERS =========
LCR: @0x0000 --> 0xfffff800
LCR: @0x0004 --> 0x00000001
LCR: @0x0008 --> 0x00200000
LCR: @0x000c --> 0x00300400
LCR: @0x0010 --> 0x00000000
LCR: @0x0014 --> 0x00000000
LCR: @0x0018 --> 0x42430343
LCR: @0x001c --> 0x00000000
LCR: @0x0020 --> 0x00000000
LCR: @0x0024 --> 0x00000000
LCR: @0x0028 --> 0x00000000
LCR: @0x002c --> 0x00000000
LCR: @0x0030 --> 0x00000000
.
.
.
LCR: @0x00c0 --> 0x00000002
LCR: @0x00c4 --> 0x00000000
LCR: @0x00c8 --> 0x00000000
LCR: @0x00cc --> 0x00000000
LCR: @0x00d0 --> 0x00000000
LCR: @0x00d4 --> 0x00000000
LCR: @0x00d8 --> 0x00000000
LCR: @0x00dc --> 0x00000000
LCR: @0x00e0 --> 0x00000000
LCR: @0x00e4 --> 0x00000000
LCR: @0x00e8 --> 0x00000050
LCR: @0x00ec --> 0x00000000
LCR: @0x00f0 --> 0x00000000
LCR: @0x00f4 --> 0x00000000
LCR: @0x00f8 --> 0x00000043
LCR: @0x00fc --> 0x00000000
LCR: @0x0100 --> 0x00000000
LCR: @0x0104 --> 0x00000000

======= PCI CONFIG REG ADDR MAPPING =========
PCR: @0x0000 --> 0x93001542
PCR: @0x0004 --> 0x02b00017
PCR: @0x0008 --> 0x08800001
PCR: @0x000c --> 0x00006008
PCR: @0x0010 --> 0xc4e01000
PCR: @0x0014 --> 0x00000000
PCR: @0x0018 --> 0xc4e00000
PCR: @0x001c --> 0x00000000
PCR: @0x0020 --> 0x00000000
PCR: @0x0024 --> 0x00000000
PCR: @0x0028 --> 0x00000000
PCR: @0x002c --> 0x905610b5
PCR: @0x0030 --> 0x00000000
PCR: @0x0034 --> 0x00000040
PCR: @0x0038 --> 0x00000000
PCR: @0x003c --> 0x0000010b
```

```
PCR: @0x0040 --> 0x00024801
PCR: @0x0044 --> 0x00000000
PCR: @0x0048 --> 0x00004c00
PCR: @0x004c --> 0x00000003
PCR: @0x0050 --> 0x00000000

======= PCI BRIDGE REGISTERS =========
PBR: @0x0000 --> 0x811110b5
PBR: @0x0004 --> 0x00100417
PBR: @0x0008 --> 0x06040021
PBR: @0x000c --> 0x00010010
PBR: @0x0010 --> 0xc4a0000c
PBR: @0x0014 --> 0x00000000
PBR: @0x0018 --> 0x00050504
PBR: @0x001c --> 0x220000f0
PBR: @0x0020 --> 0xc4e0c4e0
PBR: @0x0024 --> 0x0000fff0
PBR: @0x0028 --> 0x00000000
PBR: @0x002c --> 0x00000000
PBR: @0x0030 --> 0x00000000
.
.
.
PBR: @0x00c0 --> 0x00000000
PBR: @0x00c4 --> 0x00000000
PBR: @0x00c8 --> 0x00000000
PBR: @0x00cc --> 0x00000000
PBR: @0x00d0 --> 0x00000000
PBR: @0x00d4 --> 0x00000000
PBR: @0x00d8 --> 0x00000000
PBR: @0x00dc --> 0x00000000
PBR: @0x00e0 --> 0x00000000
PBR: @0x00e4 --> 0x00000000
PBR: @0x00e8 --> 0x00000000
PBR: @0x00ec --> 0x00000000
PBR: @0x00f0 --> 0x00000000
PBR: @0x00f4 --> 0x00000000
PBR: @0x00f8 --> 0x00000000
PBR: @0x00fc --> 0x00000000
PBR: @0x0100 --> 0x00010004
PBR: @0x0104 --> 0x00000000
PBR: @0x0108 --> 0x00000000
PBR: @0x010c --> 0x00000000
PBR: @0x0110 --> 0x00000000
PBR: @0x0114 --> 0x00000000
PBR: @0x0118 --> 0x00000000

======= MAIN CONTROL REGISTERS =========
MCR: @0x0000 --> 0x00000033
MCR: @0x0004 --> 0x8000ff00
MCR: @0x0008 --> 0x00000000
MCR: @0x000c --> 0x1b008090
MCR: @0x0010 --> 0x80000002
MCR: @0x0014 --> 0x00000000
MCR: @0x0018 --> 0x00000000
MCR: @0x001c --> 0x00000000
MCR: @0x0020 --> 0x0000141f
```

```
MCR: @0x0024 --> 0x00000000
MCR: @0x0028 --> 0x00000000
MCR: @0x002c --> 0x00000000
MCR: @0x0030 --> 0xfeedface
MCR: @0x0034 --> 0x00000000
MCR: @0x0038 --> 0x00000000
MCR: @0x003c --> 0x00000000
MCR: @0x0040 --> 0x00000201
MCR: @0x0044 --> 0x00000000
MCR: @0x0048 --> 0x00810a20
MCR: @0x004c --> 0x000000d4
MCR: @0x0050 --> 0x00010400
MCR: @0x0054 --> 0x00000000
MCR: @0x0058 --> 0x080a2c2a
MCR: @0x005c --> 0x0000029a
MCR: @0x0060 --> 0x00000019
MCR: @0x0064 --> 0x00000000
```

3.1.2 ccuruego_rdreg

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

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

Example display:
./ccuruego_rdreg –s64

```
Device Name    : /dev/ccuruego0
Board Serial No: 672340 (0x000a4254)

#### LOCAL REGS #### (length=100)
+LCL+        0   93000102  00000000  00000000  00000000 *................*
+LCL+     0x10   00000000  00000001  00000001  00000001 *................*
+LCL+     0x20   00000001  00000001  00000001  00000001 *................*
+LCL+     0x30   00000001  00000001  00000001  00000001 *................*
+LCL+     0x40   00000001  00000001  00000001  00000001 *................*
+LCL+     0x50   00000001  00000001  00000001  00000001 *................*
+LCL+     0x60   00000001                               *....
```

3.1.3 ccuruego_reg

This call displays all the boards local and configuration registers.

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

Example display:

./ccuruego_reg

```
Device Name    : /dev/ccuruego0
Board Serial No: 672340 (0x000a4254)

LOCAL Register 0x7ffff7ff6000 Offset=0x0

#### LOCAL REGS #### (length=2048)
+LCL+        0   93000102  00000000  00000000  00000000 *................*
+LCL+     0x10   00000000  00000001  00000001  00000001 *................*
```

```
+LCL+     0x20   00000001  00000001  00000001  00000001 *................*
+LCL+     0x30   00000001  00000001  00000001  00000001 *................*
+LCL+     0x40   00000001  00000001  00000001  00000001 *................*
+LCL+     0x50   00000001  00000001  00000001  00000001 *................*
+LCL+     0x60   00000001  00000001  00000001  00000001 *................*
+LCL+     0x70   00000001  00000001  00000001  00000001 *................*
+LCL+     0x80   00000001  00000001  00000001  00000001 *................*
+LCL+     0x90   00000001  00000001  00000001  00000001 *................*
+LCL+     0xa0   00000001  00000001  00000001  00000001 *................*
+LCL+     0xb0   00000000  00000001  00000001  00000001 *................*
+LCL+     0xc0   00000001  00000001  00000001  00000001 *................*
+LCL+     0xd0   00000001  00000001  00000001  00000001 *................*
+LCL+     0xe0   00000001  00000001  00000001  00000001 *................*
+LCL+     0xf0   00000000  00000000  00000001  00000001 *................*
+LCL+    0x100   00000000  00000001  00000001  00000001 *................*
+LCL+    0x110   00000000  00000000  00000000  00000000 *................*
+LCL+    0x120   00000000  00000000  00000000  00000000 *................*
+LCL+    0x130   00000001  00000001  00000001  00000001 *................*
+LCL+    0x140   00000001  00000001  00000001  00000001 *................*
+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   00000001  00000001  00000001  00000001 *................*
+LCL+    0x1a0   00000000  00000001  00000001  00000001 *................*
+LCL+    0x1b0   0000007f  0000007f  0000007f  0000007f *................*
+LCL+    0x1c0   00000001  00000001  00000001  00000001 *................*
+LCL+    0x1d0   00000001  00000001  00000001  00000001 *................*
+LCL+    0x1e0   00000001  00000001  00000001  00000001 *................*
+LCL+    0x1f0   00000001  00000001  00000001  00000001 *................*
+LCL+    0x200   00000000  00000001  00000001  00000001 *................*
.
.
.
+LCL+    0x700   00000000  00000000  00000000  00000000 *................*
+LCL+    0x710   00000000  00000000  00000000  00000000 *................*
+LCL+    0x720   00000000  00000000  00000000  00000000 *................*
+LCL+    0x730   00000000  00000000  00000000  00000000 *................*
+LCL+    0x740   00000000  00000000  00000000  00000000 *................*
+LCL+    0x750   00000000  00000000  00000000  00000000 *................*
+LCL+    0x760   00000000  00000000  00000000  00000000 *................*
+LCL+    0x770   00000000  00000000  00000000  00000000 *................*
+LCL+    0x780   00000000  00000000  00000000  00000000 *................*
+LCL+    0x790   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7a0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7b0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7c0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7d0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7e0   00000000  00000000  00000000  00000000 *................*
+LCL+    0x7f0   00000000  00000000  00000000  00000000 *................*

CONFIG Register 0x7ffff7ff5000 Offset=0x0

#### CONFIG REGS #### (length=264)
+CFG+       0   fffff800  00000001  00200000  00300400 *........ ...0..*
+CFG+     0x10  00000000  00000000  42430343  00000000 *........BC.C....*
+CFG+     0x20  00000000  00000000  00000000  00000000 *................*
+CFG+     0x30  00000000  00000008  00000000  00000000 *................*
+CFG+     0x40  00000000  00000000  00000000  00000000 *................*
+CFG+     0x50  00000000  00000000  00000000  00000000 *................*
+CFG+     0x60  00000000  00000000  0f000080  100f767c *..............v|*
+CFG+     0x70  905610b5  000000ba  00000000  00000000 *.V..............*
+CFG+     0x80  00000003  00000000  00000000  00000000 *................*
```

```
+CFG+     0x90   00000000  00000003  00000000  00000000 *................*
+CFG+     0xa0   00000000  00000000  00001010  00200000 *.............. ..*
+CFG+     0xb0   00000000  00000000  00000000  00000000 *................*
+CFG+     0xc0   00000002  00000000  00000000  00000000 *................*
+CFG+     0xd0   00000000  00000000  00000000  00000000 *................*
+CFG+     0xe0   00000000  00000000  00000050  00000000 *...........P....*
+CFG+     0xf0   00000000  00000000  00000043  00000000 *...........C....*
+CFG+    0x100   00000000  00000000                      *........        *

======= CONFIG REGISTERS =========
    las0rr                                =0xfffff800     @0x00000000
    las0ba                                =0x00000001     @0x00000004
    marbr                                 =0x00200000     @0x00000008
    bigend                                =0x00300400     @0x0000000c
    eromrr                                =0x00000000     @0x00000010
    eromba                                =0x00000000     @0x00000014
    lbrd0                                 =0x42430343     @0x00000018
    dmrr                                  =0x00000000     @0x0000001c
    dmlbam                                =0x00000000     @0x00000020
    dmlbai                                =0x00000000     @0x00000024
    dmpbam                                =0x00000000     @0x00000028
    dmcfga                                =0x00000000     @0x0000002c
    oplfis                                =0x00000000     @0x00000030
    oplfim                                =0x00000008     @0x00000034
    mbox0                                 =0x00000000     @0x00000040
    mbox1                                 =0x00000000     @0x00000044
    mbox2                                 =0x00000000     @0x00000048
    mbox3                                 =0x00000000     @0x0000004c
    mbox4                                 =0x00000000     @0x00000050
    mbox5                                 =0x00000000     @0x00000054
    mbox6                                 =0x00000000     @0x00000058
    mbox7                                 =0x00000000     @0x0000005c
    p2ldbell                              =0x00000000     @0x00000060
    l2pdbell                              =0x00000000     @0x00000064
    intcsr                                =0x0f000080     @0x00000068
    cntrl                                 =0x100f767c     @0x0000006c
    pcihidr                               =0x905610b5     @0x00000070
    pcihrev                               =0x000000ba     @0x00000074
    dmamode0                              =0x00000003     @0x00000080
    dmapadr0                              =0x00000000     @0x00000084
    dmaladr0                              =0x00000000     @0x00000088
    dmasiz0                               =0x00000000     @0x0000008c
    dmadpr0                               =0x00000000     @0x00000090
    dmamode1                              =0x00000003     @0x00000094
    dmapadr1                              =0x00000000     @0x00000098
    dmaladr1                              =0x00000000     @0x0000009c
    dmasiz1                               =0x00000000     @0x000000a0
    dmadpr1                               =0x00000000     @0x000000a4
    dmacsr0                               =0x00000010     @0x000000a8
    dmacsr1                               =0x00000010     @0x000000a9
    dmaaarb                               =0x00200000     @0x000000ac
    dmathr                                =0x00000000     @0x000000b0
    dmadac0                               =0x00000000     @0x000000b4
    dmadac1                               =0x00000000     @0x000000b8
    las1rr                                =0x00000000     @0x000000f0
    las1ba                                =0x00000000     @0x000000f4
    lbrd1                                 =0x00000043     @0x000000f8
    dmdac                                 =0x00000000     @0x000000fc
    pciarb                                =0x00000000     @0x00000100
    pabtadr                               =0x00000000     @0x00000104

======= LOCAL REGISTERS =========
    board_info                            =0x93000102     @0x00000000
```

```
    board_csr                           =0x00000000     @0x00000004
    interrupt_control                   =0x00000000     @0x00000008
    interrupt_status                    =0x00000000     @0x0000000c
    calib_bus_control                   =0x00000000     @0x000000b0
    spi_counter_status                  =0x00000000     @0x000000f0
    DAC_Control_Enable                  =0x00000000     @0x00000100
    Output_Data_Offset_DAC[0]           =0x00000000     @0x00000110
    Output_Data_Offset_DAC[1]           =0x00000000     @0x00000114
    Output_Data_Offset_DAC[2]           =0x00000000     @0x00000118
    Output_Data_Offset_DAC[3]           =0x00000000     @0x0000011c
    Output_Data_Reference_DAC[0]        =0x00000000     @0x00000120
    Output_Data_Reference_DAC[1]        =0x00000000     @0x00000124
    Output_Data_Reference_DAC[2]        =0x00000000     @0x00000128
    Output_Data_Reference_DAC[3]        =0x00000000     @0x0000012c
    Gain_Calibration_Offset_DAC[0]      =0x00000000     @0x00000150
    Gain_Calibration_Offset_DAC[1]      =0x00000000     @0x00000154
    Gain_Calibration_Offset_DAC[2]      =0x00000000     @0x00000158
    Gain_Calibration_Offset_DAC[3]      =0x00000000     @0x0000015c
    Gain_Calibration_Reference_DAC[0]   =0x00000000     @0x00000160
    Gain_Calibration_Reference_DAC[1]   =0x00000000     @0x00000164
    Gain_Calibration_Reference_DAC[2]   =0x00000000     @0x00000168
    Gain_Calibration_Reference_DAC[3]   =0x00000000     @0x0000016c
    Offset_Calibration_Offset_DAC[0]    =0x00000000     @0x00000170
    Offset_Calibration_Offset_DAC[1]    =0x00000000     @0x00000174
    Offset_Calibration_Offset_DAC[2]    =0x00000000     @0x00000178
    Offset_Calibration_Offset_DAC[3]    =0x00000000     @0x0000017c
    Offset_Calibration_Reference_DAC[0] =0x00000000     @0x00000180
    Offset_Calibration_Reference_DAC[1] =0x00000000     @0x00000184
    Offset_Calibration_Reference_DAC[2] =0x00000000     @0x00000188
    Offset_Calibration_Reference_DAC[3] =0x00000000     @0x0000018c
    Digital_Potentiometer_Control_Enable=0x00000000     @0x000001a0
    Digital_Potentiometer_Value[0]      =0x0000007f     @0x000001b0
    Digital_Potentiometer_Value[1]      =0x0000007f     @0x000001b4
    Digital_Potentiometer_Value[2]      =0x0000007f     @0x000001b8
    Digital_Potentiometer_Value[3]      =0x0000007f     @0x000001bc
    IO_Control_Enable                   =0x00000000     @0x00000200
    IO_Control[0]                       =0x00000000     @0x00000210
    IO_Control[1]                       =0x00000000     @0x00000214
    IO_Control[2]                       =0x00000000     @0x00000218
    IO_Control[3]                       =0x00000000     @0x0000021c
    PWM_Control_Enable                  =0x00000000     @0x00000240
    CJ135_Control_Enable                =0x00000000     @0x000002a0
    CJ135_External_Control              =0x00000000     @0x000002a4
    CJ135_Command_Data_Status           =0x00000000     @0x000002b0
    CJ135_Interrupt_Control_Counter     =0x00000000     @0x000002c0

        CJ135_RAM_Bank_Multi_Read[0..63]
@0x0300 0000c00b 00000608 00000aa7 000002c9 000002c3 0000062f 00000aa1 000002c5
@0x0320 0000063b 00000218 000002c6 0000f6c1 00000aa2 000002c5 0000fcd5 00000aa1
@0x0340 000002c9 0000141f 00000aa2 00001544 00000b12 0000ffbf 00000af7 00000ba9
@0x0360 00000aa4 0000180b 000008d7 00000aa2 0000ffc4 000007a3 00000aa4 000006f1
@0x0380 00000aa2 00000689 00000aa2 0000064f 00000aa7 0000062c 00000aa2 00000615
@0x03a0 00000aa1 0000060d 00000aa4 0000fffe 00000001 000007c0 00000001 00000001
@0x03c0 00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
@0x03e0 00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001

    sprom_stat_addr_write_data          =0x001f0000     @0x00000500
    sprom_read_data                     =0x001f0000     @0x00000504

        spi_ram[0..63]
@0x0700 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x0720 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x0740 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
@0x0760 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x0780 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x07a0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x07c0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x07e0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

3.1.4 ccuruego_regedit

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

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

Example display:

./ccuruego_regedit

```
Device Name     : /dev/ccuruego0
Board Serial No : 672340 (0x000a4254)
Initialize_Board: Firmware Rev. 0x1 successful

Virtual Address: 0x7ffff7ff6000
   1 = Create Physical Memory         2 = Destroy Physical memory
   3 = Display Channel Data           4 = Display Driver Information
   5 = Display Firmware RAM           6 = Display Physical Memory Info
   7 = Display Registers (CONFIG)     8 = Display Registers (LOCAL)
   9 = Dump Physical Memory          10 = Reset Board
  11 = Write Register (LOCAL)        12 = Write Register (CONFIG)
  13 = Write Physical Memory

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

3.1.5 ccuruego_tst

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

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

Example display:

./ccuruego_tst

```
Device Name     : /dev/ccuruego0
Board Serial No : 672340 (0x000a4254)
Initialize_Board: Firmware Rev. 0x1 successful

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

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

3.1.6 ccuruego_wreg

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

```
Usage: ./ccuruego_wreg [-b board] [-o offset] [-s size] [-v value] [-x]
 -b board : board selection --  default board is 0
 -o offset: hex offset to write to -- default offset is 0x0
 -s size  : number of bytes to write -- default size is 0x4
 -v value : hex value to write at offset -- default value is 0x0
 -x       : Do not read back just written values -- default read back values
```

Example display:

./ccuruego_wreg -v12345678 -o0x700 –s100

```
Device Name    : /dev/ccuruego0
Board Serial No: 12345678 (0x00bc614e)

Writing 0x12345678 to offset 0x0700 for 256 bytes

#### LOCAL REGS #### (length=256)
+LCL+    0x700   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x710   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x720   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x730   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x740   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x750   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x760   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x770   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x780   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x790   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x7a0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x7b0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x7c0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x7d0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x7e0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
+LCL+    0x7f0   12345678  12345678  12345678  12345678 *.4Vx.4Vx.4Vx.4Vx*
```

3.1.7 Flash/ccuruego_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.

```
./ccuruego_flash -[rw] -b[board] -q -s[start] -e[end] file_name
-b [board]        : board number. Default=-1
-e [end address]  : Default=0x408c83
-q                : Quite (non-interactive) mode
-r                : Read Flash and write to output file created by
./ccuruego_flash
-s [start address]: Default=0x0
-w                : Read input file and Flash the board
Use either -r or -w to read or write the ccuruego spi flash
The file_name is required

e.g. ./ccuruego_flash -w -s 0x0 -e 0x408C88 -b0 FIRMWARE/ccuruego.bin
     ./ccuruego_flash -r -s 0x0 -e 0x408c88 -b0 /tmp/ccuruego.out
```

3.1.8 Flash/ccuruego_fwreload

This program reloads the firmware. This is normally performed after a new firmware is burnt.

```
./ccuruego_fwreload -b[board]
-b [board]        : board number. Default=-1

e.g. ./ccuruego_fwreload -b0
```

3.1.9 Eeprom/ccuruego_eeprom

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

```
./ccuruego_eeprom -b[board]
-b [board]        : board number. Default=-1

e.g. ./ccuruego_eeprom -b0
```

Example display:

./ccuruego_eeprom –b0

```
Device Name    : /dev/ccuruego0
Board Serial No: 672340 (0x000a4254)

Dumping EEPROM: (0x00 - 0x3f)
@0x00:  9300 1542 0880 0001 0000 0100 0000 0000
@0x08:  0000 0000 ffff f800 0000 0001 0020 0000
@0x10:  0030 0400 0000 0000 0000 0000 4243 0343
@0x18:  0000 0000 0000 0000 0000 0000 0000 0000
@0x20:  0000 0000 9056 10b5 0000 0000 0000 0000
@0x28:  0000 0043 0000 4c00 0000 0000 0002 0000
@0x30:  0000 0000 0000 0000 0000 0000 0000 0000
@0x38:  0000 0000 0000 0000 0000 0000 0000 0100

device id           = 0x9300
vendor id           = 0x1542
subsystem device id = 0x9056
subsystem vendor id = 0x10b5
eeprom revision     = 0x0100
eeprom size         = 128 bytes
eeprom crc16        = 0xedbc

  d = Dump EEPROM                   p = Pattern Fill EEPROM
  r = Restore EEPROM to default     w = Write EEPROM

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

3.2 Application Program Interface (API) Access Example Tests

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

3.2.1 lib/ccuruego_disp

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

```
Usage: ./ccuruego_disp [-b board] [-c Channel] [-d Delay] [-F DebugFile]
                       [-l LoopCnt] [-m CJ135_Macro] [-r ChResistance] [-v] [-w]
                       [-z DisplayLegend]
 -b Board         (board # -- default = 0)
 -c Channel       (channel # -- default = 0) - [Only for -m51 or -m52]
 -d Delay         (Delay between screen refresh in milli-seconds -- default is
                                                                        100)
 -F DebugFile     (Debug file with menu display -- default "=== None ===")
    @DebugFile    (Debug file without display)
    @             (No debug file and no display)
 -l LoopCnt       (Loop Count - default = 0)
 -m CJ135_Macro   (Program CJ135 chip - default "=== None ===")
    50            (Program CJ135 to IDLE)
    51            (Program CJ135 to SWITCH ON [INTERNAL])
```

```
          52           (Program CJ135 to SWITCH ON [EXTERNAL])
          53           (Program CJ135 to WARMUP)
          54           (Program CJ135 to NORMAL-1)
       -r ChResistance (channel resistance -- default = 300) [Only for -m51 or -m52]
       -v              (Validate Macro Data -- default "=== None ===")
       -w              (Loopback Cable Connected -- default "=== None ===")
       -z              (Zap Legend Display - default = "=== Display Legend ===")
      e.g.  ./ccuruego_disp -c1 -m51 -r400 :Internal Switch On using channel 1,
                                                                  resistance=400 ohms
            ./ccuruego_disp -m54 -v        :Internal Normal-1 with data verification
      e.g.  ./ccuruego_disp -c2 -m52 -r400 :External Switch On using channel 2,
                                                                  resistance=400 ohms
            ./ccuruego_disp -m54 -v -w     :External Normal-1 with external loopback
                                                                  data verification
```

Example display:

./ccuruego_disp -b0 -c1 -m51 -r400 -d0 -v

```
 Board Number          [-b]: 0
 Channel Number        [-c]: 1 (only used for switch-on macros)
 Delay                 [-d]: 0 milli-seconds
 Debug File            [-F]: ===None===
 Loop Count            [-l]: ===Forever===
 Channel Resistance    [-r]: 400 (only used for switch-on macros)
 CJ135 Macro           [-m]: 51 (Program CJ135 to SWITCH ON [INTERNAL]))
 CJ135 Macro Validation[-v]: ===Enabled=== (Verify_Table_SWITCH_ON_INTERNAL)
 Loopback Cable        [-w]: ===Open===

 Mode                      : Set->0x0002 (Switch On), Actual->0x0002 (Switch On    [INTERNAL])
 Driver Proc Write Status  : 0. No Driver Proc Write Issued
 Connection (Channel=1)    : UN->INT, VM->INT, IP->INT, IA_RL->INT
 Interrupt Counter         : 2823 (0x0b07)
 Scan Count                : 19199
 Failed Count              : 0
 Read Duration (microsecs) :  660.471 (min= 659.235/max=1269.924/ave= 676.922)

   StatusWord1_@37 (0x7fff): IPTM=0x7fff
   StatusWord2_@38 (0x0000)
   StatusWord3_@39 (0x03c0): Cycle=15, ACTBNK0

    ID=0069   Rev=0003   HwRev=0000  ChpSta=0001   Diag0=0000  DspSta=0000   Diag1=0000 IntStat=1
  Mode=c004 UnSet=00d8      KP=0008      KI=0040      KD=0010 Up0Lean=0ee8 Up0Rich=f590
 IpSet=0000 IpBlk=0000     KRF=0000   IpOff=0000 SterPat=7fff  CfgDsp=000b    Trim=107f
RefPat=ffff  Free=0000 ActMode=c005

         <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< (Switch On    [INTERNAL]) >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
            0        1        2        3        4        5        6        7        8        9
         ====     ====     ====     ====     ====     ====     ====     ====     ====     ====
    0    6002     02e5     014c     014d     014e     0195     7fad     0547     7fcf     7fb7
    1    7fd0     0613     01e7     7fde     0210     7fb5     7fcd     7fde     7fcf     014c
    2    012f     014e     012f     014d     012f     014c     012e     014d     012d     014c
    3    012e     014c     012f     014d     014c     012e     012e     7fff     0000     03c0
    4    0000     0000     0000     0000     0000     0000     0000     0000     0000     0000
    5    0000     0000     0000     0000     0000     0000     0000     0000     0000     0000
    6    0000     0000     0000     0000

            0        1        2        3        4        5        6        7        8        9
       ======   ======   ======   ======   ======   ======   ======   ======   ======   ======
    0    6002    4.297    0.689    0.691    0.693    0.840   -0.066    1.072   -0.102   -0.151
    1  -0.100    3.227    1.011   -0.197    3.062   -0.060   -0.106   -0.197   -0.102    0.689
    2   0.629    0.693    0.629    0.691    0.629    0.689    0.627    0.691    0.625    0.689
    3   0.627    0.689    0.629    0.691    0.689    0.627    0.627     7fff     0000     03c0
    4    0000     0000     0000     0000     0000     0000     0000     0000     0000     0000
    5    0000     0000     0000     0000     0000     0000     0000     0000     0000     0000
    6    0000     0000     0000     0000

            0        1        2        3        4        5        6        7        8        9
```

```
    ======== ======== ======== ======== ======== ======== ======== ======== ======== ========
  0      mode   UipeOn    Ug0On     Ug0a     Ug0i     Ug0e Uoffgnd1    Uref1      Up0 Uria/Uof
  1  Uoffgnd2   Uref21   Ureg22 Uoffgnd3    Uref3 Uoffvcc1 Uoffvcc2 Uoffvcc3  Uoffmes      Ugk
  2    Uipcal      Ugk   Uipcal      Ugk   Uipcal      Ugk   Uipcal      Ugk   Uipcal      Ugk
  3    Uipcal      Ugk   Uipcal      Ugk   Ugk_av   Uipcal UipcalAv  StatusI StatusII StatuIII
  4     -----    -----    -----    -----    -----    -----    -----    -----    -----    -----
  5     -----    -----    -----    -----    -----    -----    -----    -----    -----    -----
  6     -----    -----    -----    -----
```

```
Board:          0
Channel:        1
Macro:          Switch On [Internal]
Validation:     Verify_Table_SWITCH_ON_INTERNAL
Loopback Cable: Connected option '-w' not selected
```

Entry	Descript	<------ HWM -----> Low	High	<-- Valid Range -> Min	Max	FailCount
0:	mode	24578.00	24578.00	24578.00	24578.00	0
1:	UipeOn	4.28	4.31	4.20	4.80	0
2:	Ug0On	0.68	0.70	0.66	0.86	0
3:	Ug0a	0.69	0.70	0.66	0.86	0
4:	Ug0i	0.69	0.70	0.66	0.86	0
5:	Ug0e	0.83	0.84	0.60	1.60	0
6:	Uoffgnd1	-0.07	-0.06	-0.30	0.30	0
7:	Uref1	1.07	1.08	1.04	1.16	0
8:	Up0	-0.11	-0.10	-0.30	0.30	0
9:	Uria/Uof	-0.16	-0.14	-0.30	0.30	0
10:	Uoffgnd2	-0.10	-0.09	-0.30	0.30	0
11:	Uref21	3.22	3.24	3.20	3.40	0
12:	Ureg22	1.00	1.01	1.00	1.20	0
13:	Uoffgnd3	-0.21	-0.17	-0.30	0.30	0
14:	Uref3	3.05	3.08	3.00	3.60	0
15:	Uoffvcc1	-0.06	-0.05	-0.30	0.30	0
16:	Uoffvcc2	-0.11	-0.10	-0.30	0.30	0
17:	Uoffvcc3	-0.21	-0.18	-0.30	0.30	0
18:	Uoffmes	-0.11	-0.10	-0.30	0.30	0
19:	Ugk	0.69	0.70	0.66	0.86	0
20:	Uipcal	0.62	0.63	0.50	0.90	0
21:	Ugk	0.69	0.70	0.66	0.86	0
22:	Uipcal	0.62	0.63	0.50	0.90	0
23:	Ugk	0.69	0.70	0.66	0.86	0
24:	Uipcal	0.62	0.63	0.50	0.90	0
25:	Ugk	0.69	0.70	0.66	0.86	0
26:	Uipcal	0.62	0.63	0.50	0.90	0
27:	Ugk	0.69	0.70	0.66	0.86	0
28:	Uipcal	0.62	0.63	0.50	0.90	0
29:	Ugk	0.69	0.70	0.66	0.86	0
30:	Uipcal	0.62	0.63	0.50	0.90	0
31:	Ugk	0.68	0.70	0.66	0.86	0
32:	Uipcal	0.62	0.63	0.50	0.90	0
33:	Ugk	0.68	0.70	0.66	0.86	0
34:	Ugk_av	0.69	0.69	0.66	0.86	0
35:	Uipcal	0.62	0.63	0.50	0.90	0
36:	UipcalAv	0.62	0.63	0.50	0.90	0
37:	StatusI	32767.00	32767.00	--	--	
38:	StatusII	0.00	0.00	--	--	
39:	StatuIII	960.00	992.00	--	--	

./ccuruego_disp -b0 -m53 -v

```
 Board Number          [-b]: 0
 Channel Number        [-c]: 1 (actual selected channel)
 Delay                 [-d]: 100 milli-seconds
 Debug File            [-F]: ===None===
 Loop Count            [-l]: ===Forever===
 Channel Resistance    [-r]: 409 (actual selected resistance)
 CJ135 Macro           [-m]: 53 (Program CJ135 to WARMUP))
 CJ135 Macro Validation[-v]: ===Enabled=== (Verify_Table_WARMUP_FROM_INTERNAL)
 Loopback Cable        [-w]: ===Open===

 Mode                      : Set->0x0005 (Warmup), Actual->0x0005 (Warmup       [INTERNAL])
 Driver Proc Write Status  : 0. No Driver Proc Write Issued
 Connection (Channel=1)    : UN->INT, VM->INT, IP->INT, IA_RL->INT
 Interrupt Counter         : 19227 (0x4b1b)
 Scan Count                : 529
 Failed Count              : 0
 Read Duration (microsecs) :  659.925 (min= 659.401/max=1257.392/ave= 689.697)

   StatusWord1_@43 (0x7fff): IPTM=0x7fff
   StatusWord2_@44 (0x0000)
   StatusWord3_@45 (0x03c0): Cycle=15, ACTBNK0

    ID=0069   Rev=0003   HwRev=0000  ChpSta=0001   Diag0=0000  DspSta=0000   Diag1=0000 IntStat=1
  Mode=c00a UnSet=00d8      KP=0008      KI=0040      KD=0010 Up0Lean=0ee8 Up0Rich=f590
 IpSet=0000 IpBlk=0000     KRF=0000   IpOff=0000 SterPat=7fff  CfgDsp=000b    Trim=107f
RefPat=ffff  Free=0000 ActMode=c00b       ComputedResistance[@25-@24]=413

         <<<<<<<<<<<<<<<<<<<<<<<<<<<<<< (Warmup       [INTERNAL]) >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
            0        1        2        3        4        5        6        7        8        9
         ====     ====     ====     ====     ====     ====     ====     ====     ====     ====
    0    6005     7fce     00af     014e     014e     7fce     00af     028d     7fce     00ae
    1    014c     7fce     00ad     014e     7fcf     00ad     0183     7fce     00af     0017
    2    055d     7fad     0548     7fcf     00b0     018c     7fcf     00ae     7fcd     7fcf
    3    00b0     7fcf     00af     7fd0     00b0     7fcf     00ae     7fd0     00b0     7fcf
    4    00ae     7fd0     00b0     7fff     0000     03c0     0000     0000     0000     0000
    5    0000     0000     0000     0000     0000     0000     0000     0000     0000     0000
    6    0000     0000     0000     0000

            0        1        2        3        4        5        6        7        8        9
       ======   ======   ======   ======   ======   ======   ======   ======   ======   ======
    0    6005   -0.104    0.363    0.693    0.693   -0.104    0.363    1.355   -0.104    0.361
    1   0.689   -0.104    0.359    0.693   -0.102    0.359    0.803   -0.104    0.363    0.048
    2   1.089   -0.066    1.073   -0.102    0.365    0.822   -0.102    0.361   -0.106   -0.102
    3   0.365   -0.102    0.363   -0.100    0.365   -0.102    0.361   -0.100    0.365   -0.102
    4   0.361   -0.100    0.365     7fff     0000     03c0     0000     0000     0000     0000
    5    0000     0000     0000     0000     0000     0000     0000     0000     0000     0000
    6    0000     0000     0000     0000

            0        1        2        3        4        5        6        7        8        9
     ======== ======== ======== ======== ======== ======== ======== ======== ======== ========
  0      mode     Up00    Un0WU      Ug0     Ug0i     Up00    Un0WU     Ug0e     Up00    Un0WU
  1      Ug0a     Up00    Un0WU   Ug0i_ei     Up00    Un0WU  Ug0i_ai     Up00    Un0WU     Uria
  2      Ucal UOFF_OFF UREF_REF     Up00    Un0WU   UrieWU     Up00    Un0WU Uoffvcci     Up00
  3     Un0WU     Up00    Un0WU     Up00    Un0WU     Up00    Un0WU     Up00    Un0WU     Up00
  4     Un0WU     Up00    Un0WU  StatusI StatusII StatuIII    -----    -----    -----    -----
  5     -----    -----    -----    -----    -----    -----    -----    -----    -----    -----
  6     -----    -----    -----    -----
```

```
Board:          0
Channel:        1
Macro:          Warmup
Validation:     Verify_Table_WARMUP_FROM_INTERNAL
Loopback Cable: Connected option '-w' not selected
```

		<------ HWM ----->		<-- Valid Range ->		
Entry	Descript	Low	High	Min	Max	FailCount
0:	mode	24581.00	24581.00	24581.00	24581.00	0
1:	Up00	-0.11	-0.10	-0.20	0.20	0
2:	Un0WU	0.36	0.37	0.33	0.53	0
3:	Ug0	0.68	0.70	0.66	0.86	0
4:	Ug0i	0.69	0.70	0.66	0.86	0
5:	Up00	-0.11	-0.10	-0.20	0.20	0
6:	Un0WU	0.36	0.37	0.33	0.53	0
7:	Ug0e	1.35	1.36	0.55	3.15	0
8:	Up00	-0.11	-0.10	-0.20	0.20	0
9:	Un0WU	0.36	0.37	0.33	0.53	0
10:	Ug0a	0.69	0.70	0.66	0.86	0
11:	Up00	-0.11	-0.10	-0.20	0.20	0
12:	Un0WU	0.35	0.36	0.33	0.53	0
13:	Ug0i_ei	0.69	0.70	0.66	0.86	0
14:	Up00	-0.11	-0.10	-0.20	0.20	0
15:	Un0WU	0.35	0.36	0.33	0.53	0
16:	Ug0i_ai	0.80	0.81	0.78	0.98	0
17:	Up00	-0.11	-0.10	-0.20	0.20	0
18:	Un0WU	0.36	0.37	0.33	0.53	0
19:	Uria	0.04	0.05	-0.30	0.30	0
20:	Ucal	1.09	1.09	1.00	1.20	0
21:	UOFF_OFF	-0.07	-0.06	-0.10	0.10	0
22:	UREF_REF	1.07	1.08	1.00	1.20	0
23:	Up00	-0.11	-0.10	-0.20	0.20	0
24:	Un0WU	0.36	0.37	0.33	0.53	0
25:	UrieWU	0.82	0.82	--	--	
26:	Up00	-0.11	-0.10	-0.20	0.20	0
27:	Un0WU	0.36	0.37	0.33	0.53	0
28:	Uoffvcci	-0.11	-0.10	-0.20	0.20	0
29:	Up00	-0.11	-0.10	-0.20	0.20	0
30:	Un0WU	0.36	0.37	0.33	0.53	0
31:	Up00	-0.11	-0.10	-0.20	0.20	0
32:	Un0WU	0.36	0.37	0.33	0.53	0
33:	Up00	-0.11	-0.10	-0.20	0.20	0
34:	Un0WU	0.36	0.37	0.33	0.53	0
35:	Up00	-0.11	-0.10	-0.20	0.20	0
36:	Un0WU	0.36	0.37	0.33	0.53	0
37:	Up00	-0.11	-0.10	-0.20	0.20	0
38:	Un0WU	0.36	0.37	0.33	0.53	0
39:	Up00	-0.11	-0.10	-0.20	0.20	0
40:	Un0WU	0.35	0.37	0.33	0.53	0
41:	Up00	-0.11	-0.10	-0.20	0.20	0
42:	Un0WU	0.36	0.37	0.33	0.53	0
43:	StatusI	32767.00	32767.00	--	--	
44:	StatusII	0.00	0.00	--	--	
45:	StatuIII	960.00	992.00	--	--	

./ccuruego_disp -b0 -m54 -v

```
Board Number          [-b]: 0
Channel Number        [-c]: 1 (actual selected channel)
Delay                 [-d]: 100 milli-seconds
Debug File            [-F]: ===None===
Loop Count            [-l]: ===Forever===
Channel Resistance    [-r]: 409 (actual selected resistance)
CJ135 Macro           [-m]: 54 (Program CJ135 to NORMAL-1))
```

```
 CJ135 Macro Validation[-v]: ===Enabled=== (Verify_Table_NORMAL_1_FROM_INTERNAL)
 Loopback Cable        [-w]: ===Open===

 Mode                      : Set->0x0003 (Normal-1), Actual->0x0003 (Normal-1     [INTERNAL])
 Driver Proc Write Status  : 0. No Driver Proc Write Issued
 Connection (Channel=1)    : UN->INT, VM->INT, IP->INT, IA_RL->INT
 Interrupt Counter         : 35215 (0x898f)
 Scan Count                : 383
 Failed Count              : 0
 Read Duration (microsecs) :  660.040 (min= 659.377/max=1260.734/ave= 708.708)

   StatusWord1_@58 (0x7fff): IPTM=0x7fff
   StatusWord2_@59 (0x0000)
   StatusWord3_@60 (0x03c0): Cycle=15, ACTBNK0

    ID=0069   Rev=0003   HwRev=0000  ChpSta=0001   Diag0=0000  DspSta=0000   Diag1=0000 IntStat=1
  Mode=c006 UnSet=00d8      KP=0008      KI=0040      KD=0010 Up0Lean=0ee8 Up0Rich=f590
 IpSet=0000 IpBlk=0000     KRF=0000   IpOff=0000 SterPat=7fff  CfgDsp=000b    Trim=107f
RefPat=ffff  Free=0000 ActMode=c007       ComputedResistance[@51-@50]=417

         <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< (Normal-1     [INTERNAL]) >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
            0        1        2        3        4        5        6        7        8        9
         ====     ====     ====     ====     ====     ====     ====     ====     ====     ====
    0    6003     7fdf     0843     7fdf     0843     7fe0     00d7     01b3     0847     014d
    1    014d     7fe0     083b     7fe0     014d     084b     7fe0     00d8     01b6     083f
    2    014f     014d     7fe0     0843     007b     007a     7fe0     0849     008c     7fe0
    3    00d7     01b4     0845     014f     014d     7fe0     0845     7fe0     0841     7fe0
    4    00d8     01b4     0845     014f     014d     7fdf     0849     7fe0     0841     7fe0
    5    00d8     01b5     0845     014e     055b     7fad     0547     014d     7fff     0000
    6    03c0     0000     0000     0000

           0        1        2        3        4        5        6        7        8        9
       ======   ======   ======   ======   ======   ======   ======   ======   ======   ======
    0    6003   -0.191     1057   -0.191     1057   -0.186    0.446    0.903     1059    0.691
    1   0.691   -0.186     1053   -0.186    0.691     1061   -0.186    0.448    0.909     1055
    2   0.695    0.691   -0.186     1057    0.713    0.707   -0.186     1060    0.812   -0.186
    3   0.446    0.905     1058    0.695    0.691   -0.186     1058   -0.186     1056   -0.186
    4   0.448    0.905     1058    0.695    0.691   -0.191     1060   -0.186     1056   -0.186
    5   0.448    0.907     1058    0.693    1.088   -0.066    1.072    0.691     7fff     0000
    6    03c0     0000     0000     0000

           0        1        2        3        4        5        6        7        8        9
     ======== ======== ======== ======== ======== ======== ======== ======== ======== ========
  0      mode  Up0(LP)       IP  Up0(LP)       IP Up0r(LP)  Un0(LN) Urie(LN)       IP  Ugi/Uga
  1   Uga/Ugi  Up0(LP)       IP Up0r(LP)      Ug0       IP Up0r(LP)  Un0(LN) UriaUrie       IP
  2   Ugi/Uga  Uga/Ugi  Up0(LP)       IP     Uape UipeUoff  Up0(LP)       IP   Up(LP) Up0r(LP)
  3   Un0(LN) UrieUoff       IP  Ugi/Uga  Uga/Ugi  Up0(LP)       IP  Up0(LP)       IP Up0r(LP)
  4   Un0(LN) Urie(LN)       IP  Ugi/Uga  Uga/Ugi  Up0(LP)       IP  Up0(LP)       IP Up0r(LP)
  5   Un0(LN) Urie(LN)       IP  Ugi/Uga  Ucal_av  UOFF_av  UREF_av  Uga/Ugi  StatusI StatusII
  6  StatuIII    -----    -----    -----

Board:          0
Channel:        1
Macro:          Normal 1
Validation:     Verify_Table_NORMAL_1_FROM_INTERNAL
Loopback Cable: Connected option '-w' not selected

                <------ HWM ----->    <-- Valid Range ->
Entry Descript  Low       High        Min       Max      FailCount
===== ========  ========  ========    ========  ======== =========
   0:     mode  24579.00  24579.00    24579.00  24579.00         0
   1:  Up0(LP)     -0.20     -0.18       -0.25      0.25         0
   2:       IP    326.00   1064.00    -2500.00   2500.00         0
   3:  Up0(LP)     -0.20     -0.18       -0.25      0.25         0
   4:       IP    333.00   1062.00    -2500.00   2500.00         0
   5: Up0r(LP)     -0.20     -0.17       -0.25      0.25         0
   6:  Un0(LN)      0.37      0.45        0.35      0.55         0
```

```
 7: Urie(LN)       0.83      0.91          --        --
 8:       IP     346.00   1063.00    -2500.00   2500.00         0
 9:  Ugi/Uga       0.69      0.70        0.67      0.87         0
10:  Uga/Ugi       0.69      0.70        0.67      0.87         0
11:  Up0(LP)      -0.20     -0.18       -0.25      0.25         0
12:       IP     351.00   1063.00    -2500.00   2500.00         0
13: Up0r(LP)      -0.20     -0.17       -0.25      0.25         0
14:      Ug0       0.69      0.70        0.57      0.97         0
15:       IP     362.00   1065.00    -2500.00   2500.00         0
16: Up0r(LP)      -0.20     -0.17       -0.25      0.25         0
17:  Un0(LN)       0.37      0.45        0.35      0.55         0
18: UriaUrie       0.83      0.91          --        --
19:       IP     371.00   1065.00    -2500.00   2500.00         0
20:  Ugi/Uga       0.69      0.70        0.67      0.87         0
21:  Uga/Ugi       0.69      0.70        0.67      0.87         0
22:  Up0(LP)      -0.20     -0.18       -0.25      0.25         0
23:       IP     380.00   1064.00    -2500.00   2500.00         0
24:     Uape       0.70      0.72        0.66      1.06         0
25: UipeUoff       0.69      0.71        0.56      1.16         0
26:  Up0(LP)      -0.20     -0.18       -0.25      0.25         0
27:       IP     384.00   1065.00    -2500.00   2500.00         0
28:   Up(LP)       0.80      0.82        0.47      1.47         0
29: Up0r(LP)      -0.20     -0.17       -0.25      0.25         0
30:  Un0(LN)       0.38      0.45        0.35      0.55         0
31: UrieUoff       0.83      0.91          --        --
32:       IP     396.00   1062.00    -2500.00   2500.00         0
33:  Ugi/Uga       0.69      0.70        0.67      0.87         0
34:  Uga/Ugi       0.69      0.70        0.67      0.87         0
35:  Up0(LP)      -0.20     -0.18       -0.25      0.25         0
36:       IP     402.00   1063.00    -2500.00   2500.00         0
37:  Up0(LP)      -0.20     -0.17       -0.25      0.25         0
38:       IP     415.00   1065.00    -2500.00   2500.00         0
39: Up0r(LP)      -0.20     -0.17       -0.25      0.25         0
40:  Un0(LN)       0.38      0.45        0.35      0.55         0
41: Urie(LN)       0.84      0.91          --        --
42:       IP     419.00   1064.00    -2500.00   2500.00         0
43:  Ugi/Uga       0.69      0.70        0.67      0.87         0
44:  Uga/Ugi       0.69      0.70        0.67      0.87         0
45:  Up0(LP)      -0.20     -0.18       -0.25      0.25         0
46:       IP     429.00   1063.00    -2500.00   2500.00         0
47:  Up0(LP)      -0.20     -0.17       -0.25      0.25         0
48:       IP     435.00   1067.00    -2500.00   2500.00         0
49: Up0r(LP)      -0.21     -0.17       -0.25      0.25         0
50:  Un0(LN)       0.38      0.45        0.35      0.55         0
51: Urie(LN)       0.84      0.91          --        --
52:       IP     445.00   1064.00    -2500.00   2500.00         0
53:  Ugi/Uga       0.69      0.70        0.67      0.87         0
54:  Ucal_av       1.09      1.09        1.00      1.20         0
55:  UOFF_av      -0.07     -0.07       -0.10      0.10         0
56:  UREF_av       1.07      1.07        1.00      1.20         0
57:  Uga/Ugi       0.69      0.70        0.67      0.87         0
58:  StatusI   32767.00  32767.00          --        --
59: StatusII       0.00      0.00          --        --
60: StatuIII     960.00    992.00          --        --
```

3.2.1.1 lib/ccuruego_display – Injecting Faults

Faults can normally be injected with the help of the *Ccuruego_IO_Set_Control()* API and a turn-around cable. In-order to test this feature, a hook has been implemented into the driver to forward a fault generation string to a running application. The syntax of the string follows a strict format. The running application *(in this case the ccuruego_display test)*, periodically queries whether a fault injection request has been received and generates the appropriate fault.

The syntax to generate faults are as follows. This command can be input from any active terminal on the system:

echo [Connection][Board#]=[Fault] > /proc/ccuruego

where: *[Connection]* is one of "UN", "VM", "IP" or "RL"
[Board#] is a valid number "0..n" representing a board that is present in the system. If only one board is present, the user must enter '0'.
[Fault] is one of "OPN", "GND", "V+".

e.g. `echo UN0=GND > /proc/ccuruego`
`echo vm2=V+ > /proc/ccuruego`

All entries are case in-sensetive.

In-order for the fault to be recognized by the *ccuruego_disp* application the following are minimum requirements:

1. The *ccuruego* driver must be installed and running.
2. The *ccuruego_disp* application needs to be running in the correct mode.
3. The mode must be External Normal-1 mode before the application considers the fault request.
4. An external loopback cable must be connected to the active channel where the fault is being generated.

To test a fault, you need to do the following steps:

1. Connect a loopback cable to the board that needs to generate a fault.
2. The loopback option '-w' must be specified in the display application.
3. Switch-on the display with the *external '-m52*' option, e.g *ccuruego_disp -b0 -m52 -c2 -v -w*
4. Switch to Normal-1 mode with the '-m54' option, e.g. *ccuruego_disp -b0 -m54 -v -w*
5. Issue a fault, e.g. *echo UN0=GND > /proc/ccuruego*

Note that if the '*-v'* verify option is specified, the display will switch the validation table to an appropriate one that is based on the fault being generated. In the above example, you should see the verification table on the display at the '-v' option change from *Verify_Table_NORMAL_1_FROM_INTERNAL* to the *Verify_Table_ERR_INJ_UN_GROUND_NORMAL_1_FROM_EXTERNAL* table.

3.2.2 lib/ccuruego_identify

This test is useful in identifying a particular card by displaying its LED.

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

Example display:

./ccuruego_identify

```
Device Name    : /dev/ccuruego0
Board ID       : 9300
```

```
Board Type     : 01
Board NumChans : 4
Board Serial No: 672335 (0x000a424f)

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)
```

3.2.3 lib/ccuruego_info

This test is useful in getting information for all the *ccuruego* devices in the system.

```
Usage: ./ccuruego_info -[bpv]
          -b <board>   (board #, default = 0)
          -p <a|d>     (Program Activate(a)/Disable(d) All, default = no program)
          -v           (Verbose, default = no verbose)
```

Example display:

./ccuruego_info

```
                      Version:  23.1.1
                        Build:  Tue Apr 14 10:17:53 EDT 2020
                       Module:  ccuruego
                  Board Index:  0 (PLX-CCURUEGO)
              Board Serial No:  672340 (0x000a4254)
              Serial Prom Rev:  0x0000
                          Bus:  5
                         Slot:  4
                         Func:  0
                    Vendor ID:  0x1542
                Sub-Vendor ID:  0x10b5
                     Board ID:  0x9300
                   Board Type:  0x0001
                Sub-Device ID:  0x9056
                   Board Info:  0x93000102
                  MSI Support:  Enabled
                    IRQ Level:  55
                     Firmware:  0x0002
              Interrupt Count:  0
            Interrupt Status:  0x0000
                     Board ID:  0x9300
                   Board Type:  0x01
          Number of Channels:  4
            All Channel Mask:  0xf
      Cal Reference Voltage:  2.400000 volts
              Voltage Range:  2.400000 volts
                   Region 0:  Addr=0xc4c01000  Size=512   (0x200)
                   Region 2:  Addr=0xc4c00000  Size=2048  (0x800)
    Calibration Bus Control:  00 (0x0) Bus Open
                DAC Control:  === Active ===
Digital Potentiometer Control:  === Active ===
                 IO Control:  === Active ===
               PWM Control:  === Disabled ===
              CJ135 Control:  === Active ===
```

./ccuruego_info –v

```
                      Version:  23.1.1
                        Build:  Tue Apr 14 10:17:53 EDT 2020
                       Module:  ccuruego
                  Board Index:  0 (PLX-CCURUEGO)
              Board Serial No:  672340 (0x000a4254)
              Serial Prom Rev:  0x0000
                          Bus:  5
                         Slot:  4
                         Func:  0
                    Vendor ID:  0x1542
                Sub-Vendor ID:  0x10b5
```

```
                     Board ID:  0x9300
                   Board Type:  0x0001
                Sub-Device ID:  0x9056
                   Board Info:  0x93000102
                  MSI Support:  Enabled
                    IRQ Level:  55
                     Firmware:  0x0002
              Interrupt Count:  0
             Interrupt Status:  0x0000
                     Board ID:  0x9300
                   Board Type:  0x01
         Number of Channels:  4
           All Channel Mask:  0xf
      Cal Reference Voltage:  2.400000 volts
             Voltage Range:  2.400000 volts
                  Region 0:  Addr=0xc4c01000  Size=512   (0x200)
                  Region 2:  Addr=0xc4c00000  Size=2048  (0x800)
    Calibration Bus Control:  00 (0x0) Bus Open
                DAC Control:  === Active ===
                 DAC Offset:  Ch0:  0.00000 (0x0000)
                           :  Ch1:  0.45000 (0x1800)
                           :  Ch2:  0.00000 (0x0000)
                           :  Ch3:  0.00000 (0x0000)
              DAC Reference:  Ch0:  0.00000 (0x0000)
                           :  Ch1:  0.00000 (0x0000)
                           :  Ch2:  0.00000 (0x0000)
                           :  Ch3:  0.00000 (0x0000)
Digital Potentiometer Control:  === Active ===
      Digital Potentiometer:  Ch0: Resistance Value=127 (Ohms=2423)
                           :  Ch1: Resistance Value=021 (Ohms=0409)
                           :  Ch2: Resistance Value=127 (Ohms=2423)
                           :  Ch3: Resistance Value=127 (Ohms=2423)
                 IO Control:  === Active ===
               I/O Control:  Ch0:   UN Signal:   0: Open (Fault or Disabled)
                           :         VM Signal:   0: Open (Fault or Disabled)
                           :         IP Signal:   0: Open (Fault or Disabled)
                           :      IA_RL Signal:   0: Open (Fault or Disabled)
                           :  Ch1:   UN Signal:   7: CJ135 (Test)
                           :         VM Signal:   7: CJ135 (Test)
                           :         IP Signal:   7: CJ135 (Test)
                           :      IA_RL Signal:   7: CJ135 (Test)
                           :  Ch2:   UN Signal:   0: Open (Fault or Disabled)
                           :         VM Signal:   0: Open (Fault or Disabled)
                           :         IP Signal:   0: Open (Fault or Disabled)
                           :      IA_RL Signal:   0: Open (Fault or Disabled)
                           :  Ch3:   UN Signal:   0: Open (Fault or Disabled)
                           :         VM Signal:   0: Open (Fault or Disabled)
                           :         IP Signal:   0: Open (Fault or Disabled)
                           :      IA_RL Signal:   0: Open (Fault or Disabled)
                PWM Control:  === Disabled ===
              CJ135 Control:  === Active ===
               CJ135 Status:  Device ID       = 0x69
                           :  Revision        = 0x03
                           :  HW Revision     = 0x00
                           :  Chip Status     = 0x01
                           :  Diagnostic 0    = 0x00
                           :  DSP Status      = 0x00
                           :  Diagnostic 1    = 0x00
                           :  Interrupt State = 0x01
                           :  Interrupt Count = 0x7d2b
    CJ135 Interrupt Control:  === Active ===
    CJ135 Interrupt Counter:  32043 (0x7d2b)
      CJ315 External Signal:  === Disabled ===
  CJ315 Fault Ground Signal:  === Connected ===
       CJ135 Address Values:   0 - MODE        = 0xc006
                           :   1 - UNSET       = 0x00d8
                           :   2 - KP          = 0x0008
                           :   3 - KI          = 0x0040
                           :   4 - KD          = 0x0010
                           :   5 - UP0LEAN     = 0x0ee8
                           :   6 - UP0RICH     = 0xf590
```

```
                          :   7 - IP_SET      = 0x0000
                          :   8 - IP_BLACK    = 0x0000
                          :   9 - KRF         = 0x0000
                          :  10 - IPOFF       = 0x0000
                          :  11 - STEERPAT    = 0x7fff
                          :  12 - CONFIGDSP   = 0x000b
                          :  13 - TRIM        = 0x107f
                          :  14 - REFPAT      = 0xffff
                          :  15 - FREE        = 0x0000
                          :  16 - ACTUAL_MODE = 0xc007
                 Bank Multi-Read:  0x00 0x0000c007  0x0000ffd3  0x00000976  0x0000ffd3
                          :  0x10 0x0000097f  0x0000ffd5  0x000001af  0x0000036d
                          :  0x20 0x0000098a  0x000002c9  0x000002c6  0x0000ffd0
                          :  0x30 0x00000967  0x0000ffd3  0x000002c3  0x0000098f
                          :  0x40 0x0000ffd3  0x000001b1  0x0000036d  0x0000097f
                          :  0x50 0x000002c9  0x000002c6  0x0000ffd3  0x0000097f
                          :  0x60 0x0000010c  0x0000010a  0x0000ffd0  0x00000976
                          :  0x70 0x0000012e  0x0000ffd5  0x000001b1  0x0000036b
                          :  0x80 0x00000986  0x000002c9  0x000002c6  0x0000ffd3
                          :  0x90 0x00000986  0x0000ffd3  0x00000962  0x0000ffd5
                          :  0xa0 0x000001b1  0x0000036b  0x0000098f  0x000002c9
                          :  0xb0 0x000002c6  0x0000ffd3  0x00000986  0x0000ffd5
                          :  0xc0 0x0000096b  0x0000ffd6  0x000001b1  0x0000036b
                          :  0xd0 0x00000986  0x000002c6  0x00000b0f  0x0000ffbf
                          :  0xe0 0x00000af8  0x000002c5  0x0000fffe  0x00000001
                          :  0xf0 0x00000781  0x00000001  0x00000001  0x00000001
```

3.2.4 lib/ccuruego_pwm

This test is useful in displaying the Pulse Width Modulation Input information and testing it using the on-board diagnostic PWM out.

```
 Usage: ./ccuruego_pwm [-b Board] [-c Channels] [-d Delay] [-F DebugFile]
                       [-l LoopCnt] [-t TrackCnt] [-V] [-X FreqRange]
                       [-Y DutyRange] [-Z ChangeHoldCount]
 -b Board       (board number -- default = 0)
 -c Channels    (Comma ',' separated channel numbers  -- default = all channels)
 -d Delay       (Delay between screen refresh in milli-seconds -- default is 10)
 -F DebugFile   (Debug file with menu display -- default "=== None ===")
    @DebugFile  (Debug file without display)
    @           (No debug file and no display)
 -l LoopCnt     (Loop Count - default = 0)
 -t TrackCnt    (Track Count only applicable with range options - default = 0)
 -v             (Verify Frequency and Duty -- default = no verification
 -X FreqRange   (Frequency Range Start,End,Increment -- default
                 10.24,73142.86,10.00)
 -Y DutyRange   (Duty Range Start,End,Increment -- default 0.01,99.99,1.00)
 -Z ChangeHoldCount (Change Hold Count -- default 10)

e.g. ./ccuruego_pwm        : Display PWM information forever
     ./ccuruego_pwm -c1,3  : Display PWM information forever for channels 1 and 3
     ./ccuruego_pwm -v     : Modify PWM Frequency and Duty using default values
                             and display and verify
     ./ccuruego_pwm -X 100,5000,1 -v : Modify PWM frequency from 100Hz to 5000Hz
                                       with increment of 1 Hz
                                     : and using default Duty and verify
     ./ccuruego_pwm -X500,500 -Y71,71: Modify PWM frequency to 500Hz and Duty to
                                       71% and display forever
     ./ccuruego_pwm -v -t1 -d0       : Modify PWM Frequency and Duty using
                                       default values and display and verify
                                     : and stop after the first track (i.e. Duty
                                       end) is complete
```

Example display:

./ccuruego_pwm

```
 Board Number          [-b]: 0
 Channels              [-c]: ch0 ch1 ch2 ch3
 Delay                 [-d]: 10 milli-seconds
 Debug File            [-F]: ===None===
 Loop Count            [-l]: ===Forever===
 Diagnostic Signal    [ch0]: (0) External
                      [ch1]: (0) External
                      [ch2]: (0) External
                      [ch3]: (0) External
 Battery Voltage    [ch0/1]: (0) No Battery Voltage
                    [ch2/3]: (0) No Battery Voltage
 Scan Count                : 7976
 Read Duration (microsecs) :    9.619 (min=   9.518/max=  19.680/ave=   9.649)

    #### Diagnostic Count ####
    <---------- (hex) ---------->  <-------- (decimal) -------->
    LowCount HighCount WidthCount  LowCount HighCount WidthCount  Duty(%)  Freq (Hz)  Period (us)
    ======== ========= ==========  ======== ========= ==========  =======  =========  ===========
    00000000  00000000   00000000         0         0          0     0.00       0.00         0.00

    #### PWM Count ####
    <---------- (hex)----------->  <-------- (decimal) -------->
Ch  LowCount HighCount WidthCount  LowCount HighCount WidthCount  Duty(%)  Freq (Hz)  Period (us)
==  ======== ========= ==========  ======== ========= ==========  =======  =========  ===========
 0  dead0000  dead0000   dead0000   stuckLO   stuckLO    stuckLO     0.00       0.00         0.00
 1  dead0000  dead0000   dead0000   stuckLO   stuckLO    stuckLO     0.00       0.00         0.00
 2  dead0000  dead0000   dead0000   stuckLO   stuckLO    stuckLO     0.00       0.00         0.00
 3  dead0000  dead0000   dead0000   stuckLO   stuckLO    stuckLO     0.00       0.00         0.00
```

./ccuruego_pwm -v

```
 Board Number          [-b]: 0
 Channels              [-c]: ch0 ch1 ch2 ch3
 Delay                 [-d]: 10 milli-seconds
 Debug File            [-F]: ===None===
 Loop Count            [-l]: ===Forever===
 Track Count           [-t]: ===Forever===
 Verify Selected       [-V]: ===True===
 Frequency Range       [-X]: Start=10.24, End=73142.86, Increment=10.00
 Duty Cycle Range      [-Y]: Start=0.01, End=99.99, Increment=1.00
 Change Hold Count     [-Z]: 10
 Diagnostic Signal    [ch0]: (1) Diagnostic
                      [ch1]: (1) Diagnostic
                      [ch2]: (1) Diagnostic
                      [ch3]: (1) Diagnostic
 Battery Voltage    [ch0/1]: (0) No Battery Voltage
                    [ch2/3]: (0) No Battery Voltage
 Failed Count   (Frequency): ch0=0 ch1=0 ch2=0 ch3=0  === PASSED ===
                    (Duty): ch0=0 ch1=0 ch2=0 ch3=0  === PASSED ===
 Scan Count                : 308
 Track Complete Count      : 0
 Read Duration (microsecs) :    9.647 (min=   9.535/max=  20.475/ave=   9.690)

    #### Diagnostic Count ####
    <---------- (hex) ---------->  <-------- (decimal) -------->
    LowCount HighCount WidthCount  LowCount HighCount WidthCount  Duty(%)  Freq (Hz)  Period (us)
    ======== ========= ==========  ======== ========= ==========  =======  =========  ===========
    00000d2a  00000023   00000d4d      3370        35       3405     1.03     300.73      3325.20

    #### PWM Count ####
    <---------- (hex)----------->  <-------- (decimal) -------->
Ch  LowCount HighCount WidthCount  LowCount HighCount WidthCount  Duty(%)  Freq (Hz)  Period (us)
==  ======== ========= ==========  ======== ========= ==========  =======  =========  ===========
 0  00000d2a  00000023   00000d4d      3370        35       3405     1.03     300.73      3325.20
 1  00000d2a  00000023   00000d4d      3370        35       3405     1.03     300.73      3325.20
 2  00000d2a  00000023   00000d4d      3370        35       3405     1.03     300.73      3325.20
 3  00000d2a  00000023   00000d4d      3370        35       3405     1.03     300.73      3325.20
```

./ccuruego_pwm -X500,500 -Y71,71 -v

```
 Board Number          [-b]: 0
 Channels              [-c]: ch0 ch1 ch2 ch3
 Delay                 [-d]: 10 milli-seconds
 Debug File            [-F]: ===None===
 Loop Count            [-l]: ===Forever===
 Track Count           [-t]: ===Forever===
 Verify Selected       [-V]: ===True===
 Frequency Range       [-X]: Start=500.00, End=500.00, Increment=0.00
 Duty Cycle Range      [-Y]: Start=71.00, End=71.00, Increment=0.00
 Change Hold Count     [-Z]: 10
 Diagnostic Signal    [ch0]: (1) Diagnostic
                      [ch1]: (1) Diagnostic
                      [ch2]: (1) Diagnostic
                      [ch3]: (1) Diagnostic
 Battery Voltage    [ch0/1]: (0) No Battery Voltage
                    [ch2/3]: (0) No Battery Voltage
 Failed Count   (Frequency): ch0=0 ch1=0 ch2=0 ch3=0  === PASSED ===
                     (Duty): ch0=0 ch1=0 ch2=0 ch3=0  === PASSED ===
 Scan Count                : 468
 Track Complete Count      : 0
 Read Duration (microsecs) :    9.592 (min=   9.534/max=  18.665/ave=   9.644)

    #### Diagnostic Count ####
    <---------- (hex) ---------->  <-------- (decimal) -------->
    LowCount HighCount WidthCount  LowCount HighCount WidthCount  Duty(%)  Freq (Hz)  Period (us)
    ======== ========= ==========  ======== ========= ==========  =======  =========  ===========
    00000251  000005af   00000800       593      1455       2048    71.04     500.00      2000.00

    #### PWM Count ####
    <---------- (hex)----------->  <-------- (decimal) -------->
Ch  LowCount HighCount WidthCount  LowCount HighCount WidthCount  Duty(%)  Freq (Hz)  Period (us)
==  ======== ========= ==========  ======== ========= ==========  =======  =========  ===========
 0  00000251  000005af   00000800       593      1455       2048    71.04     500.00      2000.00
 1  00000251  000005af   00000800       593      1455       2048    71.04     500.00      2000.00
 2  00000251  000005af   00000800       593      1455       2048    71.04     500.00      2000.00
 3  00000251  000005af   00000800       593      1455       2048    71.04     500.00      2000.00
```

3.2.5 lib/ccuruego_tst_lib

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

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

Example display:

./ccuruego_tst_lib

```
Device Name: /dev/ccuruego0
  01 = Abort DMA                              02 = Clear Driver Error
  03 = Clear Library Error                    04 = Display BOARD Registers
  05 = Display CONFIG Registers               06 = Get Board CSR
  07 = Get Board Information                  08 = Get Calibration Bus Control
  09 = Get Driver Error                       10 = Get Driver Information
  11 = Get Driver Read Mode                   12 = Get Driver Write Mode
  13 = Get Library Error                      14 = Get Mapped Config Pointer
  15 = Get Mapped Driver/Library Pointer      16 = Get Mapped Local Pointer
  17 = Get Physical Memory                    18 = Get Value
  19 = Initialize Board                       20 = MMap Physical Memory
  21 = Munmap Physical Memory                 22 = Read Operation
  23 = Read Channels                          24 = Read Single Channel
  25 = Reset Board                            26 = Select Driver Read Mode
  27 = Select Driver Write Mode               28 = Set Board CSR
  29 = Set Calibration Bus Control            30 = Set Value
  31 = ### CJ135 CONTROL MENU ###             32 = ### DAC CONTROL MENU ###
```

```
   33 = ### DIGITAL POTENTIOMETER MENU ###    34 = ### INTERRUPT MENU ###
   35 = ### IO CONTROL MENU ###               36 = ### PWM CONTROL MENU ###
   37 = ### SERIAL PROM MENU ###

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

```
Main Selection ('h'=display menu, 'q'=quit)-> 31
   Command: CJ135_control_menu()
  01 = CJ135 Activate                       02 = CJ135 Disable
  03 = CJ135 Get External Control           04 = CJ135 Interrupt Activate
  05 = CJ135 Interrupt Disable              06 = CJ135 RAM Bank Multi-Read
  07 = CJ135 Read Address                   08 = CJ135 Read RAM Address
  09 = CJ135 Set External Control           10 = CJ135 Software Reset
  11 = CJ135 Status                         12 = CJ135 Write Address

CJ135 Selection ('h'=display menu, 'q'=quit)->
```

```
Main Selection ('h'=display menu, 'q'=quit)-> 32
   Command: DAC_control_menu()
  01 = DAC Activate                         02 = DAC Disable
  03 = DAC Read Channels                    04 = DAC Write Channels

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

```
Main Selection ('h'=display menu, 'q'=quit)-> 33
   Command: digital_potentiometer_control_menu()
  01 = Digital Potentiometer Activate       02 = Digital Potentiometer Disable
  03 = Digital Potentiometer Get Resistance 04 = Digital Potentiometer Set Resistance

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

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

```
Main Selection ('h'=display menu, 'q'=quit)-> 35
   Command: IO_control_menu()
  01 = I/O Activate                         02 = I/O Disable
  03 = I/O Get Control                      04 = I/O Set Control

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

```
Main Selection ('h'=display menu, 'q'=quit)-> 36
   Command: PWM_control_menu()
  01 = PWM Activate                         02 = PWM Disable
  03 = PWM Display                          04 = PWM Get Count
  05 = PWM Get Diagnostic Count             06 = PWM Get Diagnostic Status
  07 = PWM Set Diagnostic Count             08 = PWM Set Diagnostic Frequency/Duty
  09 = PWM Set Diagnostic Signal

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

```
Main Selection ('h'=display menu, 'q'=quit)-> 37
   Command: serial_prom_menu()
  01 = Clear Serial Prom                    02 = Read Serial PROM
  03 = Serial PROM Write Override           04 = Write Serial PROM

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

3.2.6 lib/Sprom/ccuruego_sprom

This is a simple program to demonstrate sprom access.

```
Usage: ./ccuruego_sprom [-b board] [-C] [-D] [-S serialNo]
 -b <board>            (Board #, default = 0)
 -C                    (Clear ENTIRE serial PROM first)
 -D                    (Dump entire serial prom)
 -S <serialNo>         (Program board serial number)

   e.g.  ./ccuruego_sprom -C                 -> Clear Entire Serial Prom First
         ./ccuruego_sprom -D                 -> Dump Entire Serial Prom
         ./ccuruego_sprom -S 12345678        -> Write Serial Number
```

Example display:

./Sprom/ccuruego_sprom

```
Device Name:          /dev/ccuruego0
Board Serial Number:     98765 (0x000181cd)
Serial PROM Revision:        0 (0x0000)
```

This page intentionally left blank