Software Interface
CCURPWMIN (WC-PWM-1112 Input)

PCIe 12-Channel Pulse Width Modulation Input Card (PWMIN)

Driver	ccurpwmin (WC-PWM-1112)	
Platform	RedHawk Linux® *(CentOS/Rocky/RHEL & Ubuntu)*, Native Ubuntu® and Native Red Hat Enterprise Linux®[1]	
Vendor	Concurrent Real-Time	
Hardware	PCIe 12-Channel Pulse Width Modulation Input Card (CP-PWM-1112)	
Author	Darius Dubash	
Date	November 10th, 2025	Rev 2025.1

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

This page intentionally left blank

Table of Contents

1. Introduction

This document provides the software interface to the ***ccurpwmin*** driver which communicates with the Concurrent Real-Time PCI Express 12-Channel Pulse Width Modulation Input Card (CP-PWM-1112).

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

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

Various example tests have been provided in the *test* directorie to assist the user in writing their applications.

1.1 Related Documents

- Pulse Width Input Card 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/ccurpwmin0", O_RDWR);
```

The file pointer '*fp'* is then used as an argument to other system calls. The device name specified is of the format "/dev/ccurpwmin<num>" where *num* is a digit 0..9 which represents the board number that is to be accessed.

2.1.2 ioctl(2) system call

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

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

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

```
IOCTL_CCURPWMIN_ADD_IRQ
```

```
IOCTL_CCURPWMIN_DISABLE_PCI_INTERRUPTS
IOCTL_CCURPWMIN_ENABLE_PCI_INTERRUPTS
IOCTL_CCURPWMIN_GET_DRIVER_ERROR
IOCTL_CCURPWMIN_GET_DRIVER_INFO
IOCTL_CCURPWMIN_GET_PHYSICAL_MEMORY
IOCTL_CCURPWMIN_GET_READ_MODE
IOCTL_CCURPWMIN_INIT_BOARD
IOCTL_CCURPWMIN_MAIN_CONTROL_REGISTERS
IOCTL_CCURPWMIN_MMAP_SELECT
IOCTL_CCURPWMIN_NO_COMMAND
IOCTL_CCURPWMIN_PCI_BRIDGE_REGISTERS
IOCTL_CCURPWMIN_PCI_CONFIG_REGISTERS
IOCTL_CCURPWMIN_READ_EEPROM
IOCTL_CCURPWMIN_REMOVE_IRQ
IOCTL_CCURPWMIN_RESET_BOARD
IOCTL_CCURPWMIN_SELECT_READ_MODE
IOCTL_CCURPWMIN_WRITE_EEPROM
```

IOCTL_CCURPWMIN_ADD_IRQ: This *ioctl* does not have any arguments. Its purpose is to setup the driver interrupt handler to handle interrupts. This driver currently does not use interrupts for DMA and hence there is no need to use this call. This *ioctl* is only invoked if the user has issued the *IOCTL_CCURPWMIN_REMOVE_IRQ* call earlier to remove the interrupt handler.

IOCTL*_CCURPWMIN_DISABLE_PCI_INTERRUPTS:* This *ioctl* does not have any arguments. Currently, it does not perform any operation.

IOCTL_CCURPWMIN_ENABLE_PCI_INTERRUPTS: This *ioctl* does not have any arguments. Currently, it does not perform any operation.

IOCTL_CCURPWMIN_GET_DRIVER_ERROR: The argument supplied to this *ioctl* is a pointer to the *ccurpwmin_user_error_t* structure. Information on the structure is located in the *ccurpwmin_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 *CCURPWMIN_SUCCESS.*

IOCTL_CCURPWMIN_GET_DRIVER_INFO: The argument supplied to this *ioctl* is a pointer to the *ccurpwmin_ ccurpwmin_driver_info_t* structure. Information on the structure is located in the *ccurpwmin_user.h* include file. This *ioctl* provides useful driver information.

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

IOCTL_CCURPWMIN_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 CCURPWMIN_DRIVER_READ_MODE* located in the *ccurpwmin_user.h* include file.

IOCTL_CCURPWMIN_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_CCURPWMIN_RESET_BOARD* call.

IOCTL_CCURPWMIN_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

ccurpwmin_main_control_register_t structure. Raw 32-bit data values are read from the board and loaded into this structure.

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

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

IOCTL_CCURPWMIN_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 *ccurpwmin_pci_bridge_register_t* structure. Raw 32-bit data values are read from the board and loaded into this structure.

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

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

IOCTL_CCURPWMIN_REMOVE_IRQ: This *ioctl* does not have any arguments. Its purpose is to remove the interrupt handler that was previously setup. This driver currently does not use interrupts for DMA and hence there is no need to use this call. The user should not issue this call, otherwise reads will time out.

IOCTL_CCURPWMIN_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_CCURPWMIN_INIT_BOARD* call.

IOCTL_CCURPWMIN_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 CCURPWMIN_DRIVER_READ_MODE* located in the *ccurpwmin_user.h* include file.

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

2.1.3 mmap(2) system call

This system call provides the ability to map either the local board registers, the configuration board registers or create and map a physical memory that can be used for user DMA. Prior to making this system call, the user needs to issue the *ioctl(2)* system call with the *IOCTL_CCURPWMIN_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;
ccurpwmin_local_ctrl_data_t  *local_ptr;
ccurpwmin_mmap_select_t  mmap_select;
unsigned long mmap_local_size;
```

```
mmap_select.select = CCURPWMIN_SELECT_LOCAL_MMAP;
mmap_select.offset=0;
mmap_select.size=0;

ioctl(fp, IOCTL_CCURPWMIN_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 = (ccurpwmin_local_ctrl_data_t *)munmap_local_ptr;
local_ptr = (ccurpwmin_local_ctrl_data_t *)((char *)local_ptr +
                                            mmap_select.offset);
.
.
.

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

2.1.4 read(2) system call

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

CCURPWMIN_PIO_CHANNEL: This mode returns the data from 1 to 12 channels. The relative offset within the returned buffer determines the channel number. The data content is raw register values represented by the *ccurpwmin_raw_indiv_t* structure located in the *ccurpwmin_user.h* file. The driver uses Programmed I/O to perform this operation. In this mode, registers read are the latest data that are being continuously collected by the hardware. During the read operation, all data is frozen from any changes.

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

2.2 Application Program Interface (API) Access

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

```
ccurPWMIN_Add_Irq()
ccurPWMIN_CalcDutyCycle()
ccurPWMIN_CalcFreqinHz()
ccurPWMIN_CalcPeriodinUsec()
ccurPWMIN_Clear_Driver_Error()
ccurPWMIN_Clear_Lib_Error()
ccurPWMIN_Close()
ccurPWMIN_Disable_Pci_Interrupts()
ccurPWMIN_Enable_Pci_Interrupts()
ccurPWMIN_Fast_Memcpy()
ccurPWMIN_Fast_Memcpy_Unlocked()
ccurPWMIN_Flush_Fifo()
ccurPWMIN_Format_Raw_Data()
ccurPWMIN_Freeze_Output
ccurPWMIN_Fraction_To_Hex()
ccurPWMIN_Get_Driver_Error()
ccurPWMIN_Get_Driver_Read_Mode()
ccurPWMIN_Get_Info()
ccurPWMIN_Get_Lib_Error_Description()
ccurPWMIN_Get_Lib_Error()
ccurPWMIN_Get_Mapped_Config_Ptr()
ccurPWMIN_Get_Mapped_Local_Ptr()
ccurPWMIN_Get_Noise_Filter_Count()
ccurPWMIN_Get_Open_File_Descriptor()
ccurPWMIN_Get_Period_Average_Count()
ccurPWMIN_Get_Physical_Memory()
ccurPWMIN_Get_PWM()
ccurPWMIN_Get_Value()
ccurPWMIN_Initialize_Board()
ccurPWMIN_MMap_Physical_Memory()
ccurPWMIN_Munmap_Physical_Memory()
ccurPWMIN_NanoDelay()
ccurPWMIN_Open()
ccurPWMIN_Read()
ccurPWMIN_Remove_Irq()
ccurPWMIN_Reset_Board()
ccurPWMIN_Reset_PulseCount()
ccurPWMIN_Select_Driver_Read_Mode()
ccurPWMIN_Set_Noise_Filter_Count()
ccurPWMIN_Set_Period_Average_Count()
ccurPWMIN_Set_Value()
ccurPWMIN_Unfreeze_Output()
ccurPWMIN_Write()
```

2.2.1 ccurPWMIN_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 ccurPWMIN_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 ccurPWMIN_Remove_Irq(), then this call adds it back.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR          (successful)
                CCURPWMIN_LIB_BAD_HANDLE        (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN          (device not open)
                CCURPWMIN_LIB_IOCTL_FAILED      (driver ioctl call failed)
 ************************************************************************/
```

2.2.2 ccurPWMIN_CalcDutyCycle()

This call simply returns to the user the duty cycle for the raw supplied period width clock count and the period high clock count. Both these values can be returned by the hardware for each channel via programmed I/O. Normally, the user does not need to use this call as the other API *ccurPWMIN_Format_Raw_Data()* returns the duty cycle for requested channels.

```
/************************************************************************
   double ccurPWMIN_CalcDutyCycle(u_int32_t period_width_clock_count,
                                  u_int32_t period_high_clock_count)

   Description: Calculate Duty Cycle in percent

   Input:       u_int32_t period_width_clock_count    (period width clock count)
                u_int32_t period_high_clock_count     (period high clock count)
   Output:      None
   Return:      double    Calculated Duty Cycle
 ************************************************************************/
```

2.2.3 ccurPWMIN_CalcFreqinHz()

This call simply returns to the user the frequency in Hz for the raw supplied period width clock count. This value can be returned by the hardware for each channel via programmed I/O. Normally, the user does not need to use this call as the other API *ccurPWMIN_Format_Raw_Data()* returns the frequency for requested channels.

```
/************************************************************************
   double ccurPWMIN_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
 ************************************************************************/
```

2.2.4 ccurPWMIN_CalcPeriodinUsec()

This call simply returns to the user the period in micro-seconds for the raw supplied period width clock count. This value can be returned by the hardware for each channel via programmed I/O. Normally, the user does not need to use this call as the other API *ccurPWMIN_Format_Raw_Data()* returns the period for requested channels.

```
/************************************************************************
   double ccurPWMIN_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
 ************************************************************************/
```

2.2.5 ccurPWMIN_Clear_Driver_Error()

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

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

   Description: Clear any previously generated driver related error.

   Input:       void *Handle                     (handle pointer)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
                CCURPWMIN_LIB_IOCTL_FAILED       (driver ioctl call failed)
 ************************************************************************/
```

2.2.6 ccurPWMIN_Clear_Lib_Error()

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

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

   Description: Clear any previously generated library related error.

   Input:       void *Handle                     (handle pointer)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
 ************************************************************************/
```

2.2.7 ccurPWMIN_Close()

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

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

   Description: Close a previously opened device.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR          (successful)
                CCURPWMIN_LIB_BAD_HANDLE        (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN          (device not open)
 ************************************************************************/
```

2.2.8 ccurPWMIN_Disable_Pci_Interrupts()

The purpose of this call is to disable PCI interrupts. Currently, this call performs no action.

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

   Description: Disable interrupts being generated by the board.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR          (successful)
                CCURPWMIN_LIB_BAD_HANDLE        (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN          (device not open)
                CCURPWMIN_LIB_IOCTL_FAILED      (driver ioctl call failed)
 ************************************************************************/
```

2.2.9 ccurPWMIN_Enable_Pci_Interrupts()

The purpose of this call is to enable PCI interrupts. Currently this call performs no action.

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

   Description: Enable interrupts being generated by the board.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR          (successful)
                CCURPWMIN_LIB_BAD_HANDLE        (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN          (device not open)
                CCURPWMIN_LIB_IOCTL_FAILED      (driver ioctl call failed)
 ************************************************************************/
```

2.2.10 ccurPWMIN_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.

```
/*********************************************************************
   ccurPWMIN_Fast_Memcpy()
```

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

   Input:       void          *Handle       (handle pointer)
                volatile void *Source       (pointer to source buffer)
                int            SizeInBytes  (transfer size in bytes)
   Output:      volatile void *Destination  (pointer to destination buffer)
   Return:       _ccurpwmin_lib_error_number_t
                   - CCURPWMIN_LIB_NO_ERROR     (successful)
                   - CCURPWMIN_LIB_BAD_HANDLE   (no/bad handler supplied)
                   - CCURPWMIN_LIB_NOT_OPEN     (device not open)
 ********************************************************************/
```

2.2.11 ccurPWMIN_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.

```
/********************************************************************
   ccurPWMIN_Fast_Memcpy_Unlocked()

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

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

2.2.12 ccurPWMIN_Flush_Fifo()

The hardware maintains an internal FIFO of maximum size of 127 entries that holds the last *N* pulse width counts for each of the input channels. These pulse width counts are used to provide to the user a running sum of these pulse width counts which can be used to determine the average pulse width over the specified interval. This call provides the user the ability to clear this FIFO for specific channels by supplying the appropriate channel mask.

```
/************************************************************************
   int ccurPWMIN_Flush_Fifo(void *Handle, u_int32_t channel_mask)

   Description: Flush Fifo

   Input:       void         *Handle              (handle pointer)
                u_int32_t     channel_mask        (which channels)
   Return:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
                CCURPWMIN_LIB_INVALID_ARG        (invalid argument)
 ************************************************************************/

// Channel masks that can be supplied to the call
- CCURPWMIN_CH0_MASK
- CCURPWMIN_CH1_MASK
- CCURPWMIN_CH2_MASK
- CCURPWMIN_CH3_MASK
- CCURPWMIN_CH4_MASK
- CCURPWMIN_CH5_MASK
- CCURPWMIN_CH6_MASK
- CCURPWMIN_CH7_MASK
```

```
- CCURPWMIN_CH8_MASK
- CCURPWMIN_CH9_MASK
- CCURPWMIN_CH10_MASK
- CCURPWMIN_CH11_MASK
- CCURPWMIN_ALL_CH_MASK
```

2.2.13 ccurPWMIN_Format_Raw_Data()

When the user issues the *read(2)* system call to retrieve the channel information, the information returned for each channel is in a raw format in the *ccurpwmin_raw_indiv_t* structure. This call takes as input, the raw channel information read from the hardware and converts it to a more user friendly channel information and returned in the *ccurpwmin_channel_t* structure. Users can supply 1 to maximum number of channel to this call. They need to ensure that the returned value is large enough in size to receive the formatted channels.

```
/*****************************************************************************
   int ccurPWMIN_Format_Raw_Data(void *Handle, u_int32_t numChans,
                                 ccurpwmin_raw_indiv_t *RawData,
                                 ccurpwmin_channel_t *value)

   Description: Format raw data and return to user.

   Input:       void                  *Handle     (handle pointer)
                u_int32_t              numChans   (number of channels)
                ccurpwmin_raw_indiv_t  *RawData   (pointer to raw data)
   Output:      ccurpwmin_channel_t    *value;    (pointer to value)
   Return:      CCURPWMIN_LIB_NO_ERROR            (successful)
                CCURPWMIN_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN            (device not open)
                CCURPWMIN_LIB_INVALID_ARG         (invalid argument)
 *****************************************************************************/

/*** PWM individual channels ***/
typedef volatile struct {
    u_int32_t pwm_period_high_clock_count;  /* PWM period high clock count */
    u_int32_t pwm_period_width_clock_count; /* PWM width clock count */
    u_int32_t pwm_number_rising_edges;      /* PWM number of rising edges */
    u_int32_t pwm_period_sum;               /* PWM period sum */
    u_int32_t pwm_period_average_count_rcvd;/* PWM period average count received
*/
} ccurpwmin_raw_indiv_t;

typedef struct
{
    u_int32_t   pwm_period_high_clock_count;   /* PWM period high clock count */
    u_int32_t   pwm_period_width_clock_count;  /* PWM period width clock count */
    u_int32_t   pwm_number_rising_edges;       /* PWM number of rising edges */
    double      pwm_period;                    /* PWM period in micro-seconds */
    double      pwm_average_period;            /* PWM period in micro-seconds */
    double      pwm_frequency;                 /* PWM frequency Hz */
    double      pwm_duty_cycle;                /* PWM duty cycle */
    u_int32_t   pwm_period_average_count;      /* PWM period average count */
} ccurpwmin_channel_t;
```

2.2.14 ccurPWMIN_Freeze_Output()

The hardware is continuously gathering, computing and supplying to the user the most current values in various registers for each channel during each clock cycle. In order to ensure that all the data for a specific channel is not changing while being accessed by the user, this call provides the ability to freeze a selected set of channels while the information is being gathered from the hardware. Though this data for the channel is “frozen” by this

call, the board is continuing to gather and compute date for all the channels and is ready to return to the user when the freeze is removed.

```
/*****************************************************************************
   int ccurPWMIN_Freeze_Output(void *Handle, u_int32_t channel_mask)

   Description: Freeze Output

   Input:       void         *Handle              (handle pointer)
                u_int32_t     channel_mask        (which channels)
   Return:      CCURPWMIN_LIB_NO_ERROR            (successful)
                CCURPWMIN_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN            (device not open)
                CCURPWMIN_LIB_INVALID_ARG         (invalid argument)
 *****************************************************************************/

// Channel masks that can be supplied to the call
- CCURPWMIN_CH0_MASK
- CCURPWMIN_CH1_MASK
- CCURPWMIN_CH2_MASK
- CCURPWMIN_CH3_MASK
- CCURPWMIN_CH4_MASK
- CCURPWMIN_CH5_MASK
- CCURPWMIN_CH6_MASK
- CCURPWMIN_CH7_MASK
- CCURPWMIN_CH8_MASK
- CCURPWMIN_CH9_MASK
- CCURPWMIN_CH10_MASK
- CCURPWMIN_CH11_MASK
- CCURPWMIN_ALL_CH_MASK
```

2.2.15 ccurPWMIN_Get_Driver_Error()

This call returns the last error generated by the driver.

```
/*****************************************************************************
   int ccurPWMIN_Get_Driver_Error(void *Handle, ccurpwmin_user_error_t *ret_err)

   Description: Get the last error generated by the driver.

   Input:       void *Handle                      (handle pointer)
   Output:      ccurpwmin_user_error_t *ret_err   (error struct pointer)
   Return:      CCURPWMIN_LIB_NO_ERROR            (successful)
                CCURPWMIN_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN            (device not open)
                CCURPWMIN_LIB_INVALID_ARG         (invalid argument)
                CCURPWMIN_LIB_IOCTL_FAILED        (driver ioctl call failed)
 *****************************************************************************/

#define CCURPWMIN_ERROR_NAME_SIZE    64
#define CCURPWMIN_ERROR_DESC_SIZE    128
typedef struct _ccurpwmin_user_error_t {
    uint    error;                              /* error number */
    char    name[CCURPWMIN_ERROR_NAME_SIZE];    /* error name used in driver */
    char    desc[CCURPWMIN_ERROR_DESC_SIZE];    /* error description */
} ccurpwmin_user_error_t;

enum    {
    CCURPWMIN_SUCCESS = 0,
    CCURPWMIN_INVALID_PARAMETER,
    CCURPWMIN_TIMEOUT,
    CCURPWMIN_OPERATION_CANCELLED,
```

```
    CCURPWMIN_RESOURCE_ALLOCATION_ERROR,
    CCURPWMIN_INVALID_REQUEST,
    CCURPWMIN_FAULT_ERROR,
    CCURPWMIN_BUSY,
    CCURPWMIN_ADDRESS_IN_USE,
    CCURPWMIN_DMA_TIMEOUT,
};
```

2.2.16 ccurPWMIN_Get_Driver_Read_Mode()

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

```
/************************************************************************
   int ccurPWMIN_Get_Driver_Read_Mode(void *Handle,
                                      CCURPWMIN_DRIVER_READ_MODE *mode)

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

   Input:       void *Handle                       (handle pointer)
   Output:      CCURPWMIN_DRIVER_READ_MODE *mode   (pointer to read mode)
   Return:      CCURPWMIN_LIB_NO_ERROR             (successful)
                CCURPWMIN_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN             (device not open)
                CCURPWMIN_LIB_INVALID_ARG          (invalid argument)
                CCURPWMIN_LIB_NO_LOCAL_REGION      (local region error)
                CCURPWMIN_LIB_IOCTL_FAILED         (ioctl error)
 ************************************************************************/


typedef enum {
    CCURPWMIN_PIO_CHANNEL,
    CCURPWMIN_DMA_CHANNEL,
}  CCURPWMIN_DRIVER_READ_MODE;
```

2.2.17 ccurPWMIN_Get_Info()

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

```
/************************************************************************
   int ccurPWMIN_Get_Info(void *Handle,  ccurpwmin_driver_info_t *info)

   Description: Get device information from driver.

   Input:       void *Handle                     (handle pointer)
   Output:      ccurpwmin_driver_info_t *info    (info struct pointer)
                -- char  info.version
                -- char *info.built
                -- char *info.module_name[16]
                -- int   info.board_type
                -- char *info.board_desc[32]
                -- int   info.bus
                -- int   info.slot
                -- int   info.func
                -- int   info.vendor_id
                -- int   info.device_id
                -- int   info.board_id
                -- int   info.firmware
                -- int   info.interrupt_count
                -- U_int info.mem_region[].physical_address
                -- U_int info.mem_region[].size
```

```
                -- U_int info.mem_region[].flags
                -- U_int info.mem_region[].virtual_address
   Return:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
                CCURPWMIN_LIB_INVALID_ARG        (invalid argument)
                CCURPWMIN_LIB_IOCTL_FAILED       (driver ioctl call failed)
 ****************************************************************************/

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

#define CCURPWMIN_MAX_REGION 32

typedef struct
{
    char                    version[12];        /* driver version */
    char                    built[32];          /* driver date built */
    char                    module_name[16];    /* driver name */
    int                     board_type;         /* board type */
    char                    board_desc[32];     /* board description */
    int                     bus;                /* bus number */
    int                     slot;               /* slot number */
    int                     func;               /* function number */
    int                     vendor_id;          /* vendor id */
    int                     device_id;          /* device id */
    int                     board_id;           /* board id */
    int                     firmware;           /* firmware number if applicable*/
    int                     interrupt_count;    /* interrupt count */
    int                     Ccurpwmin_Max_Region;/*kernel DEVICE_COUNT_RESOURCE*/

    ccurpwmin_dev_region_t mem_region[CCURPWMIN_MAX_REGION];
} ccurpwmin_driver_info_t;
```

2.2.18 ccurPWMIN_Get_Lib_Error_Description()

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

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

   ccurPWMIN_Get_Lib_Error_Description()

   Description: Get Error Description of supplied error number.

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

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

2.2.19 ccurPWMIN_Get_Lib_Error()

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

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

```
   int ccurPWMIN_Get_Lib_Error(void *Handle, ccurpwmin_lib_error_t *lib_error)

   Description: Get last error generated by the library.

   Input:       void *Handle                      (handle pointer)
   Output:      ccurpwmin_lib_error_t *lib_error (error struct pointer)
                -- uint error                     (error number)
                -- char name[CCURPWMIN_LIB_ERROR_NAME_SIZE] (error name)
                -- char desc[CCURPWMIN_LIB_ERROR_DESC_SIZE] (error description)
                -- int  line_number              (error line number in lib)
                -- char function[CCURPWMIN_LIB_ERROR_FUNC_SIZE]
                                                 (library function in error)
   Return:      CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
                Last Library Error
 ****************************************************************************/

typedef struct _ccurpwmin_lib_error_t {
    uint    error;                              /* lib error number */
    char    name[CCURPWMIN_LIB_ERROR_NAME_SIZE]; /* error name used in lib */
    char    desc[CCURPWMIN_LIB_ERROR_DESC_SIZE]; /* error description */
    int     line_number;                        /* line number in library */
    char    function[CCURPWMIN_LIB_ERROR_FUNC_SIZE];
                                                /* library function */
} ccurpwmin_lib_error_t;
```

2.2.20 ccurPWMIN_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 *ccurpwmin_user.h* include file that is supplied with the driver.

```
/****************************************************************************
   int ccurPWMIN_Get_Mapped_Config_Ptr(void *Handle,
                                       ccurpwmin_config_local_data_t **config_ptr)

   Description: Get mapped configuration pointer.

   Input:       void *Handle                      (handle pointer)
   Output:      ccurpwmin_config_local_data_t **config_ptr (config struct ptr)
                -- structure in ccurpwmin_user.h
   Return:      CCURPWMIN_LIB_NO_ERROR            (successful)
                CCURPWMIN_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN            (device not open)
                CCURPWMIN_LIB_INVALID_ARG         (invalid argument)
                CCURPWMIN_LIB_NO_CONFIG_REGION    (config region not present)
 ****************************************************************************/
```

2.2.21 ccurPWMIN_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 *ccurpwmin_user.h* include file that is supplied with the driver.

```
/*****************************************************************************
   int ccurPWMIN_Get_Mapped_Local_Ptr(void *Handle,
                                      ccurpwmin_local_ctrl_data_t **local_ptr)

   Description: Get mapped local pointer.

   Input:       void *Handle                        (handle pointer)
   Output:      ccurpwmin_local_ctrl_data_t **local_ptr (local struct ptr)
                -- structure in ccurpwmin_user.h
   Return:      CCURPWMIN_LIB_NO_ERROR              (successful)
                CCURPWMIN_LIB_BAD_HANDLE            (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN              (device not open)
                CCURPWMIN_LIB_INVALID_ARG           (invalid argument)
                CCURPWMIN_LIB_NO_LOCAL_REGION       (local region not present)
 *****************************************************************************/
```

2.2.22 ccurPWMIN_Get_Noise_Filter_Count()

The board is capable of filtering out some very high frequency noise spikes if the user so desires. The users can set this filter count from 0 (i.e. no filter) to the maximum allowable filter count specified by the define *CCURPWMIN_MAX_NOISE_FILTER_COUNT*. This call returns the noise filter count that has been previously set by the *ccurPWMIN_Set_Noise_Filter_Count().* The count is the number of noise transitions that are to be skipped within the duration of the clock ticks specified in this filter.

```
/*****************************************************************************
   int ccurPWMIN_Get_Noise_Filter_Count(void *Handle, u_int32_t channel,
                                        u_int32_t *value)

   Description: Get Noise Filter Count

   Input:       void *Handle                        (handle pointer)
                u_int32_t    channel                (channel selection)
                u_int32_t    *value                 (value to be set)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR              (successful)
                CCURPWMIN_LIB_BAD_HANDLE            (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN              (device not open)
                CCURPWMIN_LIB_INVALID_ARG           (invalid argument)
 *****************************************************************************/
```

2.2.23 ccurPWMIN_Get_Open_File_Descriptor()

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

```
/*****************************************************************************
   int ccurPWMIN_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:      CCURPWMIN_LIB_NO_ERROR              (successful)
                CCURPWMIN_LIB_BAD_HANDLE            (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN              (device not open)
                CCURPWMIN_LIB_INVALID_ARG           (invalid argument)
```

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

2.2.24 ccurPWMIN_Get_Period_Average_Count()

The board maintains an internal FIFO for each channel that holds the last *N* pulse width counts. This call returns the number of pulse width counts that the hardware is using to save the last set of pulse widths encountered. This list is maintained by the hardware to provide a running sum of the last *N* pulse widths that is then used by the API to determine the average of the last *N* pulse widths encountered by the channel.

```
/***************************************************************************
   int ccurPWMIN_Get_Period_Average_Count(void *Handle, u_int32_t channel,
                                          u_int32_t *value)

   Description: Get Period Average Count

   Input:       void *Handle                     (handle pointer)
                u_int32_t    channel             (channel selection)
                u_int32_t    *value              (value to be set)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
                CCURPWMIN_LIB_INVALID_ARG        (invalid argument)
 ***************************************************************************/
```

2.2.25 ccurPWMIN_Get_Physical_Memory()

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

```
/***************************************************************************
   int ccurPWMIN_Get_Physical_Memory(void *Handle,
                                     ccurpwmin_phys_mem_t *phys_mem)

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

   Input:       void *Handle                     (handle pointer)
   Output:      ccurpwmin_phys_mem_t *phys_mem   (mem struct pointer)
                -- void *phys_mem
                -- u_int phys_mem_size
   Return:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
                CCURPWMIN_LIB_INVALID_ARG        (invalid argument)
                CCURPWMIN_LIB_IOCTL_FAILED       (driver ioctl call failed)
 ***************************************************************************/

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

2.2.26 ccurPWMIN_Get_PWM()

This call returns to the user information about a particular channel or all the channels. Additionally, the hardware maintains a continuous pulse count for each channel which latches the pulse counts since the last reset and then

clears the counter. The user can optionally set the *reset_pulsecount* argument to '1' to request the API to perform to latch the pulse count and the clear it.

The user can specify a single channel number from 0 to (*CCURPWMIN_MAX_CHANNELS – 1)* to receive the contents of a specific channel. If the user wishes to receive information for ALL channels, then they can specify *CCURPWMIN_MAX_CHANNELS* as the argument to *channel.* In this case, the *ccurpwmin_channel_t* structure pointed to by *value* must be large enough to receive all the channels.

```
/************************************************************************
   int ccurPWMIN_Get_PWM(void *Handle, u_int32_t channel,
                         ccurpwmin_channel_t *value, int reset_pulsecount)

   Description: Return the individual settings of the specified channel.

   Input:       void          *Handle             (handle pointer)
                u_int32_t     channel             (which channel)
                int           reset_pulsecount    (reset pulse count flag)
   Output:      ccurpwmin_channel_t *value;       (pointer to value)
   Return:      CCURPWMIN_LIB_NO_ERROR            (successful)
                CCURPWMIN_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN            (device not open)
                CCURPWMIN_LIB_INVALID_ARG         (invalid argument)
 ************************************************************************/

typedef struct
{
    u_int32_t   pwm_period_high_clock_count;   /* PWM period high clock count */
    u_int32_t   pwm_period_width_clock_count;  /* PWM period width clock count */
    u_int32_t   pwm_number_rising_edges;       /* PWM number of rising edges */
    double      pwm_period;                    /* PWM period in micro-seconds */
    double      pwm_average_period;            /* PWM period in micro-seconds */
    double      pwm_frequency;                 /* PWM frequency Hz */
    double      pwm_duty_cycle;                /* PWM duty cycle */
    u_int32_t   pwm_period_average_count;      /* PWM period average count */
} ccurpwmin_channel_t;
```

2.2.27 ccurPWMIN_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.

```
/************************************************************************
   int ccurPWMIN_Get_Value(void *Handle, CCURPWMIN_CONTROL cmd, void *value)

   Description: Return the value of the specified board register.

   Input:       void              *Handle        (handle pointer)
                CCURPWMIN_CONTROL  cmd           (register definition)
   Output:      void              *value;        (pointer to value)
   Return:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
                CCURPWMIN_LIB_INVALID_ARG        (invalid argument)
                CCURPWMIN_LIB_NO_LOCAL_REGION    (local region not present)
 ************************************************************************/

typedef enum {
    CCURPWMIN_STATUS,
    CCURPWMIN_REVISION,
```

```
CCURPWMIN_RESET,
CCURPWMIN_RESET_PULSECOUNT,
CCURPWMIN_FREEZE_OUTPUT,
CCURPWMIN_FLUSH_FIFO,

CCURPWMIN_INDIV0_PERIOD_HIGH_CLOCK_COUNT,
CCURPWMIN_INDIV0_PERIOD_WIDTH_CLOCK_COUNT,
CCURPWMIN_INDIV0_NUMBER_RISING_EDGES,
CCURPWMIN_INDIV0_PERIOD_SUM,
CCURPWMIN_INDIV0_PWM_PERIOD_SUM_COUNT_RECEIVED,
CCURPWMIN_INDIV0_PWM_PERIOD_SUM_COUNT_SET,
CCURPWMIN_INDIV0_PWM_NOISE_FILTER_COUNT,

CCURPWMIN_INDIV1_PERIOD_HIGH_CLOCK_COUNT,
CCURPWMIN_INDIV1_PERIOD_WIDTH_CLOCK_COUNT,
CCURPWMIN_INDIV1_NUMBER_RISING_EDGES,
CCURPWMIN_INDIV1_PERIOD_SUM,
CCURPWMIN_INDIV1_PWM_PERIOD_SUM_COUNT_RECEIVED,
CCURPWMIN_INDIV1_PWM_PERIOD_SUM_COUNT_SET,
CCURPWMIN_INDIV1_PWM_NOISE_FILTER_COUNT,

CCURPWMIN_INDIV2_PERIOD_HIGH_CLOCK_COUNT,
CCURPWMIN_INDIV2_PERIOD_WIDTH_CLOCK_COUNT,
CCURPWMIN_INDIV2_NUMBER_RISING_EDGES,
CCURPWMIN_INDIV2_PERIOD_SUM,
CCURPWMIN_INDIV2_PWM_PERIOD_SUM_COUNT_RECEIVED,
CCURPWMIN_INDIV2_PWM_PERIOD_SUM_COUNT_SET,
CCURPWMIN_INDIV2_PWM_NOISE_FILTER_COUNT,

CCURPWMIN_INDIV3_PERIOD_HIGH_CLOCK_COUNT,
CCURPWMIN_INDIV3_PERIOD_WIDTH_CLOCK_COUNT,
CCURPWMIN_INDIV3_NUMBER_RISING_EDGES,
CCURPWMIN_INDIV3_PERIOD_SUM,
CCURPWMIN_INDIV3_PWM_PERIOD_SUM_COUNT_RECEIVED,
CCURPWMIN_INDIV3_PWM_PERIOD_SUM_COUNT_SET,
CCURPWMIN_INDIV3_PWM_NOISE_FILTER_COUNT,

CCURPWMIN_INDIV4_PERIOD_HIGH_CLOCK_COUNT,
CCURPWMIN_INDIV4_PERIOD_WIDTH_CLOCK_COUNT,
CCURPWMIN_INDIV4_NUMBER_RISING_EDGES,
CCURPWMIN_INDIV4_PERIOD_SUM,
CCURPWMIN_INDIV4_PWM_PERIOD_SUM_COUNT_RECEIVED,
CCURPWMIN_INDIV4_PWM_PERIOD_SUM_COUNT_SET,
CCURPWMIN_INDIV4_PWM_NOISE_FILTER_COUNT,

CCURPWMIN_INDIV5_PERIOD_HIGH_CLOCK_COUNT,
CCURPWMIN_INDIV5_PERIOD_WIDTH_CLOCK_COUNT,
CCURPWMIN_INDIV5_NUMBER_RISING_EDGES,
CCURPWMIN_INDIV5_PERIOD_SUM,
CCURPWMIN_INDIV5_PWM_PERIOD_SUM_COUNT_RECEIVED,
CCURPWMIN_INDIV5_PWM_PERIOD_SUM_COUNT_SET,
CCURPWMIN_INDIV5_PWM_NOISE_FILTER_COUNT,

CCURPWMIN_INDIV6_PERIOD_HIGH_CLOCK_COUNT,
CCURPWMIN_INDIV6_PERIOD_WIDTH_CLOCK_COUNT,
CCURPWMIN_INDIV6_NUMBER_RISING_EDGES,
CCURPWMIN_INDIV6_PERIOD_SUM,
CCURPWMIN_INDIV6_PWM_PERIOD_SUM_COUNT_RECEIVED,
CCURPWMIN_INDIV6_PWM_PERIOD_SUM_COUNT_SET,
CCURPWMIN_INDIV6_PWM_NOISE_FILTER_COUNT,

CCURPWMIN_INDIV7_PERIOD_HIGH_CLOCK_COUNT,
CCURPWMIN_INDIV7_PERIOD_WIDTH_CLOCK_COUNT,
```

```
    CCURPWMIN_INDIV7_NUMBER_RISING_EDGES,
    CCURPWMIN_INDIV7_PERIOD_SUM,
    CCURPWMIN_INDIV7_PWM_PERIOD_SUM_COUNT_RECEIVED,
    CCURPWMIN_INDIV7_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV7_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV8_PERIOD_HIGH_CLOCK_COUNT,
    CCURPWMIN_INDIV8_PERIOD_WIDTH_CLOCK_COUNT,
    CCURPWMIN_INDIV8_NUMBER_RISING_EDGES,
    CCURPWMIN_INDIV8_PERIOD_SUM,
    CCURPWMIN_INDIV8_PWM_PERIOD_SUM_COUNT_RECEIVED,
    CCURPWMIN_INDIV8_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV8_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV9_PERIOD_HIGH_CLOCK_COUNT,
    CCURPWMIN_INDIV9_PERIOD_WIDTH_CLOCK_COUNT,
    CCURPWMIN_INDIV9_NUMBER_RISING_EDGES,
    CCURPWMIN_INDIV9_PERIOD_SUM,
    CCURPWMIN_INDIV9_PWM_PERIOD_SUM_COUNT_RECEIVED,
    CCURPWMIN_INDIV9_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV9_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV10_PERIOD_HIGH_CLOCK_COUNT,
    CCURPWMIN_INDIV10_PERIOD_WIDTH_CLOCK_COUNT,
    CCURPWMIN_INDIV10_NUMBER_RISING_EDGES,
    CCURPWMIN_INDIV10_PERIOD_SUM,
    CCURPWMIN_INDIV10_PWM_PERIOD_SUM_COUNT_RECEIVED,
    CCURPWMIN_INDIV10_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV10_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV11_PERIOD_HIGH_CLOCK_COUNT,
    CCURPWMIN_INDIV11_PERIOD_WIDTH_CLOCK_COUNT,
    CCURPWMIN_INDIV11_NUMBER_RISING_EDGES,
    CCURPWMIN_INDIV11_PERIOD_SUM,
    CCURPWMIN_INDIV11_PWM_PERIOD_SUM_COUNT_RECEIVED,
    CCURPWMIN_INDIV11_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV11_PWM_NOISE_FILTER_COUNT,
} CCURPWMIN_CONTROL;
```

2.2.28 ccurPWMIN_Initialize_Board()

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

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

   Description: Initialize the board.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR          (successful)
                CCURPWMIN_LIB_BAD_HANDLE        (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN          (device not open)
                CCURPWMIN_LIB_IOCTL_FAILED      (driver ioctl call failed)
                CCURPWMIN_LIB_NO_LOCAL_REGION   (local region not present)
 *************************************************************************/
```

2.2.29 ccurPWMIN_MMap_Physical_Memory()

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

```
/*************************************************************************
   int ccurPWMIN_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:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
                CCURPWMIN_LIB_INVALID_ARG        (invalid argument)
                CCURPWMIN_LIB_MMAP_SELECT_FAILED (mmap selection failed)
                CCURPWMIN_LIB_MMAP_FAILED        (mmap failed)
 *************************************************************************/
```

2.2.30 ccurPWMIN_Munmap_Physical_Memory()

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

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

   Description: Unmap a previously mapped physical DMA memory.

   Input:       void *Handle                     (handle pointer)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
                CCURPWMIN_LIB_MUNMAP_FAILED      (failed to un-map memory)
                CCURPWMIN_LIB_NOT_MAPPED         (memory not mapped)
 *************************************************************************/
```

2.2.31 ccurPWMIN_NanoDelay()

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

```
/*************************************************************************
   void ccurPWMIN_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

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

2.2.32 ccurPWMIN_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/ccurpwmin<Board_Number>* for the call to be successful. One character special file is created for each board found when the driver is successfully loaded.

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

```
/*************************************************************************
   int ccurPWMIN_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:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_INVALID_ARG        (invalid argument)
                CCURPWMIN_LIB_ALREADY_OPEN       (device already opened)
                CCURPWMIN_LIB_OPEN_FAILED        (device open failed)
                CCURPWMIN_LIB_ALREADY_MAPPED     (memory already mmapped)
                CCURPWMIN_LIB_MMAP_SELECT_FAILED (mmap selection failed)
                CCURPWMIN_LIB_MMAP_FAILED        (mmap failed)
 *************************************************************************/
```

2.2.33 ccurPWMIN_Read()

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

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

```
/*************************************************************************
   int ccurPWMIN_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:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
                CCURPWMIN_LIB_IO_ERROR           (read failed)
                CCURPWMIN_LIB_FIFO_OVERFLOW      (FIFO overflow)
 *************************************************************************/
```

2.2.34 ccurPWMIN_Remove_Irq()

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

```
/************************************************************************
   int ccurPWMIN_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:      CCURPWMIN_LIB_NO_ERROR          (successful)
                CCURPWMIN_LIB_BAD_HANDLE        (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN          (device not open)
                CCURPWMIN_LIB_IOCTL_FAILED      (driver ioctl call failed)
 ************************************************************************/
```

2.2.35 ccurPWMIN_Reset_Board()

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

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

   Description: Reset the board.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR          (successful)
                CCURPWMIN_LIB_BAD_HANDLE        (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN          (device not open)
                CCURPWMIN_LIB_IOCTL_FAILED      (driver ioctl call failed)
                CCURPWMIN_LIB_NO_LOCAL_REGION   (local region not present)
 ************************************************************************/
```

2.2.36 ccurPWMIN_Reset_PulseCount()

The driver maintains a continuous number of pulse counts that are being detected on each channel. This call allows the user to latch the contents of the pulse counts since the last pulse reset. After latching the contents, the hardware resets the counter and continues pulse count detection.

```
/************************************************************************
   ccurPWMIN_Reset_PulseCount()

   Description: Issue reset pulse count

   Input:       void         *Handle             (handle pointer)
                u_int32_t     channel_mask       (which channels)
   Return:      CCURPWMIN_LIB_NO_ERROR           (successful)
                CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN           (device not open)
```

```
                CCURPWMIN_LIB_INVALID_ARG         (invalid argument)
 ***************************************************************************/
```

2.2.37 ccurPWMIN_Select_Driver_Read_Mode()

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

```
/***************************************************************************
   int ccurPWMIN_Select_Driver_Read_Mode(void *Handle,
                                         CCURPWMIN_DRIVER_READ_MODE mode)

   Description: Reset Fifo

   Input:       void *Handle                       (handle pointer)
                CCURPWMIN_DRIVER_READ_MODE mode    (select read mode)
   Output:      none
   Return:      CCURPWMIN_LIB_NO_ERROR             (successful)
                CCURPWMIN_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN             (device not open)
                CCURPWMIN_LIB_INVALID_ARG          (invalid argument)
                CCURPWMIN_LIB_NO_LOCAL_REGION      (local region not present)
 ***************************************************************************/

typedef enum {
    CCURPWMIN_PIO_CHANNEL,
    CCURPWMIN_DMA_CHANNEL,
}  CCURPWMIN_DRIVER_READ_MODE;
```

2.2.38 ccurPWMIN_Set_Noise_Filter_Count()

The hardware can perform some basic noise filtering on a per-channel basis. Users can set the noise filter count anywhere from *CCURPWMIN_MIN_NOISE_FILTER_COUNT* (where no noise rejection will occur) to *CCURPWMIN_MAX_NOISE_FILTER_COUNT*. The value supplied requests the hardware to skip high frequency noise transitions that occur within the number of clock ticks supplied to this call. The user can specify a single channel number from 0 to (*CCURPWMIN_MAX_CHANNELS – 1)* to set the filter for a specific channel. If the user wishes to set filter for ALL channels, then they can specify *CCURPWMIN_MAX_CHANNELS* as the argument to *channel*.

```
/***************************************************************************
   ccurPWMIN_Set_Noise_Filter_Count()

   Description: Set Noise Filter Count

   Input:       void *Handle                       (handle pointer)
                u_int32_t    channel               (channel selection)
                u_int32_t    value                 (value to be set)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR             (successful)
                CCURPWMIN_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN             (device not open)
                CCURPWMIN_LIB_INVALID_ARG          (invalid argument)
 ***************************************************************************/
```

2.2.39 ccurPWMIN_Set_Period_Average_Count()

This call sets the count of the number that is required for determining the most recent period average. The driver maintains an internal FIFO for each channel that hold the most recent period widths and provides this information to the user in the form of the sum of these periods. The sum of the periods is supplied to the user in a 32-bit register. Users need to ensure that the window size of average selection times the period width count must not exceed the 32-bit register, otherwise, incorrect averaging will result. This is only true when the input pulse is of a very low frequency.(less than 0.52Hz) with the maximum window size of 127. As the frequency is reduced, the user needs to reduce the window size accordingly. The *ccurPWMIN_Get_PWM()* API uses this information to return to the user the average of the collected pulse widths.

```
/*************************************************************************
   ccurPWMIN_Set_Period_Average_Count()

   Description: Set Period Average Count

   Input:       void *Handle                    (handle pointer)
                u_int32_t    channel            (channel selection)
                u_int32_t    value              (value to be set)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR          (successful)
                CCURPWMIN_LIB_BAD_HANDLE        (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN          (device not open)
                CCURPWMIN_LIB_INVALID_ARG       (invalid argument)
 *************************************************************************/
```

2.2.40 ccurPWMIN_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.

```
/*************************************************************************
   int ccurPWMIN_Set_Value(void *Handle, CCURPWMIN_CONTROL cmd, int value)

   Description: Set the value of the specified board register.

   Input:       void *Handle                    (handle pointer)
                CCURPWMIN_CONTROL cmd           (register definition)
                int value                       (value to be set)
   Output:      None
   Return:      CCURPWMIN_LIB_NO_ERROR          (successful)
                CCURPWMIN_LIB_BAD_HANDLE        (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN          (device not open)
                CCURPWMIN_LIB_INVALID_ARG       (invalid argument)
 *************************************************************************/
typedef enum {
    CCURPWMIN_STATUS,
    CCURPWMIN_RESET,
    CCURPWMIN_RESET_PULSECOUNT,
    CCURPWMIN_FREEZE_OUTPUT,
    CCURPWMIN_FLUSH_FIFO,

    CCURPWMIN_INDIV0_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV0_PWM_NOISE_FILTER_COUNT,
```

```
    CCURPWMIN_INDIV1_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV1_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV2_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV2_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV3_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV3_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV4_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV4_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV5_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV5_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV6_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV6_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV7_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV7_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV8_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV8_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV9_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV9_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV10_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV10_PWM_NOISE_FILTER_COUNT,

    CCURPWMIN_INDIV11_PWM_PERIOD_SUM_COUNT_SET,
    CCURPWMIN_INDIV11_PWM_NOISE_FILTER_COUNT,
} CCURPWMIN_CONTROL;
```

2.2.41 ccurPWMIN_Unfreeze_Output()

This call un-freezes data collection that was previously frozen by the *ccurPWMIN_Freeze_Output()* call. User can specify a set of channels to un-freeze.

```
/*************************************************************************
   ccurPWMIN_Unfreeze_Output()

   Description: Unfreeze Output

   Input:       void         *Handle              (handle pointer)
                u_int32_t     channel_mask        (which channels)
   Return:      CCURPWMIN_LIB_NO_ERROR            (successful)
                CCURPWMIN_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWMIN_LIB_NOT_OPEN            (device not open)
                CCURPWMIN_LIB_INVALID_ARG         (invalid argument)
 *************************************************************************/
```

2.2.42 ccurPWMIN_Write()

This call is not supported for this Analog Input card.

```
/*************************************************************************
   int ccurPWMIN_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:      CCURPWMIN_LIB_NO_ERROR           (successful)
                 CCURPWMIN_LIB_BAD_HANDLE         (no/bad handler supplied)
                 CCURPWMIN_LIB_NOT_OPEN           (device not open)
                 CCURPWMIN_LIB_IO_ERROR           (write failed)
                 CCURPWMIN_LIB_NOT_IMPLEMENTED    (call not implemented)
 ************************************************************************/
```

3. Test Programs

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

3.1 Direct Driver Access Example Tests

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

3.1.1 ccurpwmin_dump

This is a simple program that dumps the local, configuration, PCI bridge, PCI config and main control registers.

Usage: ccurpwmin_dump <device number>

Example display:

```
Device Name    : /dev/ccurpwmin0
LOCAL Register 0x7ffff7ff5000 Offset=0x0
CONFIG Register 0x7ffff7ff4000 Offset=0x0

======= LOCAL BOARD REGISTERS =========
LBR: @0x0000 --> 0x00010000
LBR: @0x0004 --> 0x00020002
LBR: @0x0008 --> 0x00000000
LBR: @0x000c --> 0x00000000
LBR: @0x0010 --> 0x00000000
LBR: @0x0014 --> 0x00000000
...
LBR: @0x1000 --> 0x00000000
LBR: @0x1004 --> 0x00000000
LBR: @0x1008 --> 0x00000000
LBR: @0x100c --> 0x00000000
LBR: @0x1010 --> 0x00000000
LBR: @0x1014 --> 0x00000000
...
LBR: @0x38ec --> 0x00000000
LBR: @0x38f0 --> 0x00000000
LBR: @0x38f4 --> 0x00000000
LBR: @0x38f8 --> 0x00000000
LBR: @0x38fc --> 0x00000000

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

```
LCR: @0x0000 --> 0xffff8000
LCR: @0x0004 --> 0x00000001
LCR: @0x0008 --> 0x00200000
LCR: @0x000c --> 0x00000400
LCR: @0x0010 --> 0x00000000
LCR: @0x0014 --> 0x00000011
LCR: @0x0018 --> 0xf20301db
LCR: @0x001c --> 0x00000000
LCR: @0x0020 --> 0x00000000
LCR: @0x0024 --> 0x00000000
LCR: @0x0028 --> 0x00001009
LCR: @0x002c --> 0x00000000
LCR: @0x0030 --> 0x00000000
LCR: @0x0034 --> 0x00000008
LCR: @0x0038 --> 0x00000000
LCR: @0x003c --> 0x00000000
LCR: @0x0040 --> 0x00000000
LCR: @0x0044 --> 0x00000000
LCR: @0x0048 --> 0x00000000
LCR: @0x004c --> 0x00000000
LCR: @0x0050 --> 0x00000000
LCR: @0x0054 --> 0x00000000
LCR: @0x0058 --> 0x00000000
LCR: @0x005c --> 0x00000000
LCR: @0x0060 --> 0x00000000
LCR: @0x0064 --> 0x00000000
LCR: @0x0068 --> 0x0f000483
LCR: @0x006c --> 0x100f767e
LCR: @0x0070 --> 0x905610b5
LCR: @0x0074 --> 0x000000ba
LCR: @0x0078 --> 0x00000000
LCR: @0x007c --> 0x00000000
LCR: @0x0080 --> 0x00000043
LCR: @0x0084 --> 0x17e53000
LCR: @0x0088 --> 0x00001400
LCR: @0x008c --> 0x000000f0
LCR: @0x0090 --> 0x0000000a
LCR: @0x0094 --> 0x00000003
LCR: @0x0098 --> 0x00000000
LCR: @0x009c --> 0x00000000
LCR: @0x00a0 --> 0x00000000
LCR: @0x00a4 --> 0x00000000
LCR: @0x00a8 --> 0x00001011
LCR: @0x00ac --> 0x00200000
LCR: @0x00b0 --> 0x00000000
LCR: @0x00b4 --> 0x00000000
LCR: @0x00b8 --> 0x00000000
LCR: @0x00bc --> 0x00000000
LCR: @0x00c0 --> 0x00000002
LCR: @0x00c4 --> 0x00000000
LCR: @0x00c8 --> 0x00000000
LCR: @0x00cc --> 0x00000000
LCR: @0x00d0 --> 0x00000000
LCR: @0x00d4 --> 0x00000000
LCR: @0x00d8 --> 0x00000000
LCR: @0x00dc --> 0x00000000
LCR: @0x00e0 --> 0x00000000
LCR: @0x00e4 --> 0x00000000
LCR: @0x00e8 --> 0x00000050
LCR: @0x00ec --> 0x00000000
LCR: @0x00f0 --> 0x00000000
LCR: @0x00f4 --> 0x00000000
LCR: @0x00f8 --> 0x00000043
```

```
======= PCI CONFIG REG ADDR MAPPING =========
PCR: @0x0000 --> 0x92721542
PCR: @0x0004 --> 0x02b00017
PCR: @0x0008 --> 0x08800001
PCR: @0x000c --> 0x00006008
PCR: @0x0010 --> 0xbd508000
PCR: @0x0014 --> 0x00000000
PCR: @0x0018 --> 0xbd500000
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 --> 0x00100017
PBR: @0x0008 --> 0x06040021
PBR: @0x000c --> 0x00010010
PBR: @0x0010 --> 0xbd20000c
PBR: @0x0014 --> 0x00000000
PBR: @0x0018 --> 0x00070706
PBR: @0x001c --> 0x220000f0
PBR: @0x0020 --> 0xbd50bd50
PBR: @0x0024 --> 0x0000fff0
PBR: @0x0028 --> 0x00000000
PBR: @0x002c --> 0x00000000
PBR: @0x0030 --> 0x00000000
PBR: @0x0034 --> 0x00000040
PBR: @0x0038 --> 0x00000000
PBR: @0x003c --> 0x0000010b
PBR: @0x0040 --> 0x5a025001
PBR: @0x0044 --> 0x00000000
PBR: @0x0048 --> 0x000e2012
PBR: @0x004c --> 0x00000000
PBR: @0x0050 --> 0x00806005
PBR: @0x0054 --> 0x00000000
PBR: @0x0058 --> 0x00000000
PBR: @0x005c --> 0x00000000
PBR: @0x0060 --> 0x00710010
PBR: @0x0064 --> 0x00640000
PBR: @0x0068 --> 0x00002000
PBR: @0x006c --> 0x00024c11
PBR: @0x0070 --> 0x00110000
PBR: @0x0074 --> 0x00000c80
PBR: @0x0078 --> 0x00400000
PBR: @0x007c --> 0x00000000
PBR: @0x0080 --> 0x00000000
PBR: @0x0084 --> 0x00000000
PBR: @0x0088 --> 0x00000033
PBR: @0x008c --> 0x00000000
PBR: @0x0090 --> 0x00000000
PBR: @0x0094 --> 0x00000000
```

```
PBR: @0x0098 --> 0x00000000
PBR: @0x009c --> 0x00000000
PBR: @0x00a0 --> 0x00000000
PBR: @0x00a4 --> 0x00000000
PBR: @0x00a8 --> 0x00000000
PBR: @0x00ac --> 0x00000000
PBR: @0x00b0 --> 0x00000000
PBR: @0x00b4 --> 0x00000000
PBR: @0x00b8 --> 0x00000000
PBR: @0x00bc --> 0x00000000
PBR: @0x00c0 --> 0x00000000
PBR: @0x00c4 --> 0x00000000
PBR: @0x00c8 --> 0x00000000
PBR: @0x00cc --> 0x00000000
PBR: @0x00d0 --> 0x00000000
PBR: @0x00d4 --> 0x00000000
PBR: @0x00d8 --> 0x00000000
PBR: @0x00dc --> 0x00000000
PBR: @0x00e0 --> 0x00000000
PBR: @0x00e4 --> 0x00000000
PBR: @0x00e8 --> 0x00000000
PBR: @0x00ec --> 0x00000000
PBR: @0x00f0 --> 0x00000000
PBR: @0x00f4 --> 0x00000000
PBR: @0x00f8 --> 0x00000000
PBR: @0x00fc --> 0x00000000
PBR: @0x0100 --> 0x00010004
PBR: @0x0104 --> 0x00000000
PBR: @0x0108 --> 0x00000000
PBR: @0x010c --> 0x00000000
PBR: @0x0110 --> 0x00000000
PBR: @0x0114 --> 0x00000000
PBR: @0x0118 --> 0x00000000

======= MAIN CONTROL REGISTERS =========
MCR: @0x0000 --> 0x00000033
MCR: @0x0004 --> 0x8000ff00
MCR: @0x0008 --> 0x00000000
MCR: @0x000c --> 0x03008090
MCR: @0x0010 --> 0x80000000
MCR: @0x0014 --> 0x00000000
MCR: @0x0018 --> 0x00000000
MCR: @0x001c --> 0x00000000
MCR: @0x0020 --> 0x0000101f
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 --> 0x00010600
MCR: @0x0054 --> 0x00000000
MCR: @0x0058 --> 0x080a2c2a
MCR: @0x005c --> 0x0000029a
MCR: @0x0060 --> 0x00000019
MCR: @0x0064 --> 0x00000000
```

3.1.2 ccurpwmin_rdreg

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

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

Example display:

```
Read at offset 0x0000: 0x00010000
```

3.1.3 ccurpwmin_reg

This is a simple program that dumps the local and configuration registers.

Usage: ccurpwmin_reg <device number>

Example display:

```
Device Name    : /dev/ccurpwmin0
LOCAL Register 0x7ffff7ff0000 Offset=0x0
CONFIG Register 0x7ffff7fef000 Offset=0x0

#### CONFIG REGS #### (length=512)
+CFG+        0   ffff8000  00000001  00200000  00000400 *........ ......*
+CFG+     0x10   00000000  00000011  f20301db  00000000 *................*
+CFG+     0x20   00000000  00000000  00001009  00000000 *................*
+CFG+     0x30   00000000  00000008  00000000  00000000 *................*
+CFG+     0x40   00000000  00000000  00000000  00000000 *................*
+CFG+     0x50   00000000  00000000  00000000  00000000 *................*
+CFG+     0x60   00000000  00000000  0f000403  100f767e *..............v~*
+CFG+     0x70   905610b5  000000ba  00000000  00000000 *.V..............*
+CFG+     0x80   00000043  17e53000  00001400  000000f0 *...C..0.........*
+CFG+     0x90   0000000a  00000003  00000000  00000000 *................*
+CFG+     0xa0   00000000  00000000  00001011  00200000 *.............  ..*
+CFG+     0xb0   00000000  00000000  00000000  00000000 *................*
+CFG+     0xc0   00000002  00000000  00000000  00000000 *................*
+CFG+     0xd0   00000000  00000000  00000000  00000000 *................*
+CFG+     0xe0   00000000  00000000  00000050  00000000 *...........P....*
+CFG+     0xf0   00000000  00000000  00000043  00000000 *...........C....*
+CFG+    0x100   00000000  17e530e8  00000000  00000000 *......0.........*
+CFG+    0x110   00000000  00000000  00000000  00000000 *................*
+CFG+    0x120   00000000  00000000  00000000  00000000 *................*
+CFG+    0x130   00000000  00000000  00000000  00000000 *................*
+CFG+    0x140   00000000  00000000  00000000  00000000 *................*
+CFG+    0x150   00000000  00000000  00000000  00000000 *................*
+CFG+    0x160   00000000  00000000  00000000  00000000 *................*
+CFG+    0x170   00000000  00000000  00000000  00000000 *................*
+CFG+    0x180   00000000  00000000  00000000  00000000 *................*
+CFG+    0x190   00000000  00000000  00000000  00000000 *................*
+CFG+    0x1a0   00000000  00000000  00000000  00000000 *................*
+CFG+    0x1b0   00000000  00000000  00000000  00000000 *................*
+CFG+    0x1c0   00000000  00000000  00000000  00000000 *................*
+CFG+    0x1d0   00000000  00000000  00000000  00000000 *................*
+CFG+    0x1e0   00000000  00000000  00000000  00000000 *................*
+CFG+    0x1f0   00000000  00000000  00000000  00000000 *................*


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

```
pwm_status                                          =0x00010000 @0x00000000
pwm_revision                                        =0x00020002 @0x00000004
pwm_reset                                           =0x00000000 @0x00001000
pwm_reset_pulsecount                                =0xfffff000 @0x00001100
pwm_freeze_output                                   =0xfffff000 @0x00001104
pwm_flush_fifo                                      =0xfffff000 @0x00001108
pwm_indiv0.pwm_period_high_clock_count              =0xdeadffff @0x00001400
pwm_indiv0.pwm_period_width_clock_count             =0xdeadffff @0x00001404
pwm_indiv0.pwm_number_rising_edges                  =0x00000000 @0x00001408
pwm_indiv0.pwm_period_sum                           =0x00000000 @0x0000140c
pwm_indiv0.pwm_period_average_count_rcvd            =0x00000000 @0x00001410
pwm_indiv_control0.pwm_period_average_count_set =0x00000001 @0x00001200
pwm_indiv_control0.pwm_noise_filter_count           =0x00000014 @0x00001204
pwm_indiv1.pwm_period_high_clock_count              =0xdeadffff @0x00001414
pwm_indiv1.pwm_period_width_clock_count             =0xdeadffff @0x00001418
pwm_indiv1.pwm_number_rising_edges                  =0x00000000 @0x0000141c
pwm_indiv1.pwm_period_sum                           =0x00000000 @0x00001420
pwm_indiv1.pwm_period_average_count_rcvd            =0x00000000 @0x00001424
pwm_indiv_control1.pwm_period_average_count_set =0x00000001 @0x00001208
pwm_indiv_control1.pwm_noise_filter_count           =0x00000014 @0x0000120c
pwm_indiv2.pwm_period_high_clock_count              =0xdeadffff @0x00001428
pwm_indiv2.pwm_period_width_clock_count             =0xdeadffff @0x0000142c
pwm_indiv2.pwm_number_rising_edges                  =0x00000000 @0x00001430
pwm_indiv2.pwm_period_sum                           =0x00000000 @0x00001434
pwm_indiv2.pwm_period_average_count_rcvd            =0x00000000 @0x00001438
pwm_indiv_control2.pwm_period_average_count_set =0x00000001 @0x00001210
pwm_indiv_control2.pwm_noise_filter_count           =0x00000014 @0x00001214
pwm_indiv3.pwm_period_high_clock_count              =0xdeadffff @0x0000143c
pwm_indiv3.pwm_period_width_clock_count             =0xdeadffff @0x00001440
pwm_indiv3.pwm_number_rising_edges                  =0x00000000 @0x00001444
pwm_indiv3.pwm_period_sum                           =0x00000000 @0x00001448
pwm_indiv3.pwm_period_average_count_rcvd            =0x00000000 @0x0000144c
pwm_indiv_control3.pwm_period_average_count_set =0x00000001 @0x00001218
pwm_indiv_control3.pwm_noise_filter_count           =0x00000014 @0x0000121c
pwm_indiv4.pwm_period_high_clock_count              =0xdeadffff @0x00001450
pwm_indiv4.pwm_period_width_clock_count             =0xdeadffff @0x00001454
pwm_indiv4.pwm_number_rising_edges                  =0x00000000 @0x00001458
pwm_indiv4.pwm_period_sum                           =0x00000000 @0x0000145c
pwm_indiv4.pwm_period_average_count_rcvd            =0x00000000 @0x00001460
pwm_indiv_control4.pwm_period_average_count_set =0x00000001 @0x00001220
pwm_indiv_control4.pwm_noise_filter_count           =0x00000014 @0x00001224
pwm_indiv5.pwm_period_high_clock_count              =0xdeadffff @0x00001464
pwm_indiv5.pwm_period_width_clock_count             =0xdeadffff @0x00001468
pwm_indiv5.pwm_number_rising_edges                  =0x00000000 @0x0000146c
pwm_indiv5.pwm_period_sum                           =0x00000000 @0x00001470
pwm_indiv5.pwm_period_average_count_rcvd            =0x00000000 @0x00001474
pwm_indiv_control5.pwm_period_average_count_set =0x00000001 @0x00001228
pwm_indiv_control5.pwm_noise_filter_count           =0x00000014 @0x0000122c
pwm_indiv6.pwm_period_high_clock_count              =0xdeadffff @0x00001478
pwm_indiv6.pwm_period_width_clock_count             =0xdeadffff @0x0000147c
pwm_indiv6.pwm_number_rising_edges                  =0x00000000 @0x00001480
pwm_indiv6.pwm_period_sum                           =0x00000000 @0x00001484
pwm_indiv6.pwm_period_average_count_rcvd            =0x00000000 @0x00001488
pwm_indiv_control6.pwm_period_average_count_set =0x00000001 @0x00001230
pwm_indiv_control6.pwm_noise_filter_count           =0x00000014 @0x00001234
pwm_indiv7.pwm_period_high_clock_count              =0xdeadffff @0x0000148c
pwm_indiv7.pwm_period_width_clock_count             =0xdeadffff @0x00001490
pwm_indiv7.pwm_number_rising_edges                  =0x00000000 @0x00001494
pwm_indiv7.pwm_period_sum                           =0x00000000 @0x00001498
pwm_indiv7.pwm_period_average_count_rcvd            =0x00000000 @0x0000149c
pwm_indiv_control7.pwm_period_average_count_set =0x00000001 @0x00001238
pwm_indiv_control7.pwm_noise_filter_count           =0x00000014 @0x0000123c
pwm_indiv8.pwm_period_high_clock_count              =0xdeadffff @0x000014a0
```

```
    pwm_indiv8.pwm_period_width_clock_count         =0xdeadffff @0x000014a4
    pwm_indiv8.pwm_number_rising_edges              =0x00000000 @0x000014a8
    pwm_indiv8.pwm_period_sum                       =0x00000000 @0x000014ac
    pwm_indiv8.pwm_period_average_count_rcvd        =0x00000000 @0x000014b0
    pwm_indiv_control8.pwm_period_average_count_set =0x00000001 @0x00001240
    pwm_indiv_control8.pwm_noise_filter_count       =0x00000014 @0x00001244
    pwm_indiv9.pwm_period_high_clock_count          =0xdeadffff @0x000014b4
    pwm_indiv9.pwm_period_width_clock_count         =0xdeadffff @0x000014b8
    pwm_indiv9.pwm_number_rising_edges              =0x00000000 @0x000014bc
    pwm_indiv9.pwm_period_sum                       =0x00000000 @0x000014c0
    pwm_indiv9.pwm_period_average_count_rcvd        =0x00000000 @0x000014c4
    pwm_indiv_control9.pwm_period_average_count_set =0x00000001 @0x00001248
    pwm_indiv_control9.pwm_noise_filter_count       =0x00000014 @0x0000124c
    pwm_indiv10.pwm_period_high_clock_count         =0xdeadffff @0x000014c8
    pwm_indiv10.pwm_period_width_clock_count        =0xdeadffff @0x000014cc
    pwm_indiv10.pwm_number_rising_edges             =0x00000000 @0x000014d0
    pwm_indiv10.pwm_period_sum                      =0x00000000 @0x000014d4
    pwm_indiv10.pwm_period_average_count_rcvd       =0x00000000 @0x000014d8
    pwm_indiv_control10.pwm_period_average_count_set=0x00000001 @0x00001250
    pwm_indiv_control10.pwm_noise_filter_count      =0x00000014 @0x00001254
    pwm_indiv11.pwm_period_high_clock_count         =0xdeadffff @0x000014dc
    pwm_indiv11.pwm_period_width_clock_count        =0xdeadffff @0x000014e0
    pwm_indiv11.pwm_number_rising_edges             =0x00000000 @0x000014e4
    pwm_indiv11.pwm_period_sum                      =0x00000000 @0x000014e8
    pwm_indiv11.pwm_period_average_count_rcvd       =0x00000000 @0x000014ec
    pwm_indiv_control11.pwm_period_average_count_set=0x00000001 @0x00001258
    pwm_indiv_control11.pwm_noise_filter_count      =0x00000014 @0x0000125c

        spi_ram[0..63]
@0x3800 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x3820 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x3840 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x3860 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x3880 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x38a0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x38c0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
@0x38e0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

======= CONFIG REGISTERS =========
    las0rr                                          =0xffff8000 @0x00000000
    las0ba                                          =0x00000001 @0x00000004
    marbr                                           =0x00200000 @0x00000008
    bigend                                          =0x00000400 @0x0000000c
    eromrr                                          =0x00000000 @0x00000010
    eromba                                          =0x00000011 @0x00000014
    lbrd0                                           =0xf20301db @0x00000018
    dmrr                                            =0x00000000 @0x0000001c
    dmlbam                                          =0x00000000 @0x00000020
    dmlbai                                          =0x00000000 @0x00000024
    dmpbam                                          =0x00001009 @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                                          =0x0f000483 @0x00000068
cntrl                                           =0x100f767e @0x0000006c
pcihidr                                         =0x905610b5 @0x00000070
pcihrev                                         =0x000000ba @0x00000074
dmamode0                                        =0x00000043 @0x00000080
dmapadr0                                        =0x17e53000 @0x00000084
dmaladr0                                        =0x00001400 @0x00000088
dmasiz0                                         =0x000000f0 @0x0000008c
dmadpr0                                         =0x0000000a @0x00000090
dmamode1                                        =0x00000003 @0x00000094
dmapadr1                                        =0x00000000 @0x00000098
dmaladr1                                        =0x00000000 @0x0000009c
dmasiz1                                         =0x00000000 @0x000000a0
dmadpr1                                         =0x00000000 @0x000000a4
dmacsr0                                         =0x00001011 @0x000000a8
dmacsr1                                         =0x00200000 @0x000000ac
las1rr                                          =0x00000000 @0x000000f0
las1ba                                          =0x00000000 @0x000000f4
lbrd1                                           =0x00000043 @0x000000f8
```

3.1.4 ccurpwmin_tst

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

Usage: ccurpwmin_tst <device number>

Example display:

```
Initialize_Board: Firmware Rev. 0x10002 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)->
```

3.1.5 ccurpwmin_wreg

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

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

Example display:

```
Writing 0x00000000 to offset 0x0000
Read at offset 0x0000: 0x00010000
```

3.2 Application Program Interface (API) Access Example Tests

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

3.2.1 ccurpwmin_disp

Useful program to display all the analog input channels using various read modes and test the driver/board. This program uses the *curses* library.

```
Usage: ./ccurpwmin_disp [-b board] [-c average_count] [-d delay]
[-E expected_frequency] [-F debug_file] [-i ignore_mask] [-l loop_count]
[-mD|-mp|-mP] [-n noise_filter] [-r] [-t tolerance]
 -b <board>              (default = 0)
 -c <average count>      (default = 30)
 -d <delay> microsecs    (default = 1000000)
 -E <expected_frequency> (default = no expected frequency test)
 -F <debug_file>         (write to debug file)
 -F @<debug_file>        (No display, however write to debug file)
 -F @                    (No display and no writing to debug file)
 -i <ignore_mask>        (Channels NOT to Test (default is test all))
 -l <loop_channel>       (Loop Count (default = Infinite))
 -mD                     (Driver DMA read mode)
 -mp                     (User PIO read mode)
 -mP                     (Driver PIO read mode)
 -n <noise filter count> (default = 20)
 -r                      (Perform Register Test)
 -t <tolerance>          (Frequency Tolerance (default is 5.0%))
```

Example display:

```
           Board Num [-b]:  0
               Delay [-d]:  1000000 (usec)
           Read Mode [-m]:  DRIVER_DMA_CHANNEL
                  Version:  25.1.0
                    Build:  Thu May  6 11:03:25 EDT 2025
                   Module:  ccurpwmin
               Board Type:  0 (PLX-CCURPWMIN)
                Vendor ID:  0x1542
                Device ID:  0x9272
                 Board ID:  0x9056
                 Firmware:  0x20002
               Interrupts:  0
                 Region 0:  Addr=0xfb708000  Size=512   (0x200)
                 Region 2:  Addr=0xfb700000  Size=32768 (0x8000)
     Ignore Channel Mask:  0x0
 Period Average Count Set:  30 [CH0]
   Noise Filter Count Set:  20 [CH0]
               cycleTime:  1000068.1 usec
                  ioTime:  13.1 usec
              Loop Count:  10 [Fail 0, Pass 120]
      Expected Frequency:  20000Hz
               Tolerance:  5.0%
```

Chan	Period (us)	Freq(Hz)	Duty%	WidthCount	HighCount	NumRiseEdge	PeriodAve	AveCount
[0]	0.00	0.00	0.00	all_low	all_low	0	0.00	0
[1]	0.00	0.00	0.00	all_low	all_low	0	0.00	0
[2]	0.00	0.00	0.00	all_low	all_low	0	0.00	0
[3]	0.00	0.00	0.00	all_low	all_low	0	0.00	0
[4]	0.00	0.00	0.00	all_low	all_low	0	0.00	0
[5]	49.98	20006.06	41.25	3299	1361	20000	50.00	30
[6]	0.00	0.00	100.00	ALL_HIGH	ALL_HIGH	0	0.00	0
[7]	0.00	0.00	100.00	ALL_HIGH	ALL_HIGH	0	0.00	0
[8]	0.00	0.00	100.00	ALL_HIGH	ALL_HIGH	0	0.00	0
[9]	0.00	0.00	100.00	ALL_HIGH	ALL_HIGH	0	0.00	0
[10]	0.00	0.00	100.00	ALL_HIGH	ALL_HIGH	0	0.00	0
[11]	0.00	0.00	100.00	ALL_HIGH	ALL_HIGH	0	0.00	0

3.2.2 ccurpwmin_tst_lib

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

```
Usage: ccurpwmin_tst_lib <device number>
```

Example display:

```
  01 = Add Irq                        02 = Calculate Duty Cycle
  03 = Calculate Pulse Frequency      04 = Clear Driver Error
  05 = Clear Library Error            06 = Disable Pci Interrupts
  07 = Display BOARD Registers        08 = Enable Pci Interrupts
  09 = Flush FIFO                     10 = Freeze Output
  11 = Get Information                12 = Get Driver Error
  13 = Get Driver Read Mode           14 = Get Library Error
  15 = Get Mapped Config Pointer      16 = Get Mapped Local Pointer
  17 = Get Noise Filter Count         18 = Get Period Average Count
  19 = Get Physical Memory            20 = Get PWM
  21 = Get Value                      22 = Initialize Board
  23 = MMap Physical Memory           24 = Munmap Physical Memory
  25 = Reset PulseCount               26 = Read Operation
  27 = Remove Irq                     28 = Reset Board
  29 = Select Driver Read Mode        30 = Set Noise Filter Count
  31 = Set Period Average Count       32 = Set Value
  33 = Unfreeze Output                34 = Test Registers
  35 = Write Operation

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

This page intentionally left blank