Software Interface
CCURPWM (WC-PWM-1012 Output)

PCIe 12-Channel Pulse Width Modulation Output Card (PWM)

Driver	ccurpwm (WC-PWM-1012)	v 23.0_1
OS	RedHawk	
Vendor	Concurrent Real-Time, Inc.	
Hardware	PCIe 12-Channel Pulse Width Modulation Output Card (CP-PWM-1012)	
Date	April 13, 2018	

This page intentionally left blank

Table of Contents

This page intentionally left blank

1. Introduction

This document provides the software interface to the ***ccurpwm*** driver which communicates with the Concurrent Real-Time PCI Express 12-Channel Pulse Width Modulation Output Card (CP-PWM-1012).

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* directories to assist the user in writing their applications.

1.1 Related Documents

- Pulse Width Output 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/ccurpwm0”, 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/ccurpwm<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_CCURPWM_ADD_IRQ
IOCTL_CCURPWM_DISABLE_PCI_INTERRUPTS
IOCTL_CCURPWM_ENABLE_PCI_INTERRUPTS
IOCTL_CCURPWM_GET_DRIVER_ERROR
IOCTL_CCURPWM_GET_DRIVER_INFO
IOCTL_CCURPWM_GET_PHYSICAL_MEMORY
IOCTL_CCURPWM_INIT_BOARD
IOCTL_CCURPWM_MAIN_CONTROL_REGISTERS
IOCTL_CCURPWM_MMAP_SELECT
IOCTL_CCURPWM_NO_COMMAND
IOCTL_CCURPWM_PCI_BRIDGE_REGISTERS
IOCTL_CCURPWM_PCI_CONFIG_REGISTERS
IOCTL_CCURPWM_REMOVE_IRQ
IOCTL_CCURPWM_RESET_BOARD
```

IOCTL_CCURPWM_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_CCURPWM_REMOVE_IRQ* call earlier to remove the interrupt handler.

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

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

IOCTL_CCURPWM_GET_DRIVER_ERROR: The argument supplied to this *ioctl* is a pointer to the *ccurpwm_user_error_t* structure. Information on the structure is located in the *ccurpwm_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 *CCURPWM_SUCCESS.*

IOCTL_CCURPWM_GET_DRIVER_INFO: The argument supplied to this *ioctl* is a pointer to the *ccurpwm_ ccurpwm_driver_info_t* structure. Information on the structure is located in the *ccurpwm_user.h* include file. This *ioctl* provides useful driver information.

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

IOCTL_CCURPWM_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_CCURPWM_RESET_BOARD* call.

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

IOCTL_CCURPWM_MMAP_SELECT: The argument to this *ioctl* is a pointer to the *ccurpwm_mmap_select_t* structure. Information on the structure is located in the *ccurpwm_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 *(CCURPWM_SELECT_LOCAL_MMAP)*, the configuration register space *(CCURPWM_SELECT_CONFIG_MMAP)* and a physical memory *(CCURPWM_SELECT_PHYS_MEM_MMAP)* that is created by the *mmap(2)* system call.

IOCTL_CCURPWM_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_CCURPWM_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 *ccurpwm_pci_bridge_register_t* structure. Raw 32-bit data values are read from the board and loaded into this structure.

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

IOCTL_CCURPWM_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_CCURPWM_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_CCURPWM_INIT_BOARD* call.

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_CCURPWM_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;
ccurpwm_local_ctrl_data_t  *local_ptr;
ccurpwm_mmap_select_t  mmap_select;
unsigned long mmap_local_size;

mmap_select.select = CCURPWM_SELECT_LOCAL_MMAP;
mmap_select.offset=0;
mmap_select.size=0;

ioctl(fp, IOCTL_CCURPWM_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 = (ccurpwm_local_ctrl_data_t *)munmap_local_ptr;
local_ptr = (ccurpwm_local_ctrl_data_t *)((char *)local_ptr +
                                              mmap_select.offset);
if(munmap_local_ptr != NULL)
    munmap((void *)munmap_local_ptr, mmap_local_size);
```

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.

```
Ccurpwm_Add_Irq()
Ccurpwm_Clear_Driver_Error()
Ccurpwm_Clear_Lib_Error()
Ccurpwm_Close()
Ccurpwm_Disable_Pci_Interrupts()
Ccurpwm_Enable_Pci_Interrupts()
Ccurpwm_Fast_Memcpy()
Ccurpem_Fast_Memcpy_Unlocked()
Ccurpwm_Get_Driver_Error()
Ccurpwm_Get_Info()
Ccurpwm_Get_Lib_Error()
Ccurpwm_Get_Mapped_Config_Ptr()
Ccurpwm_Get_Mapped_Local_Ptr()
Ccurpwm_Get_Physical_Memory()
Ccurpwm_Get_PWM()
Ccurpwm_Get_PWM_Individual()
Ccurpwm_Get_Value()
Ccurpwm_Initialize_Board()
Ccurpwm_MMap_Physical_Memory()
Ccurpwm_Munmap_Physical_Memory()
Ccurpwm_NanoDelay()
Ccurpwm_Open()
ccurpwm_PWM_Resync()
Ccurpwm_Read()
Ccurpwm_Remove_Irq()
Ccurpwm_Reset_Board()
ccurpwm_Set_PWM()
Ccurpwm_Set_PWM_Individual()
Ccurpwm_Set_Value()
Ccurpwm_Write()
```

2.2.1 Ccurpwm_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 Ccurpwm_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 Ccurpwm_Remove_Irq(), then this call adds it back.

   Input:       void *Handle                        (handle pointer)
   Output:      None
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_IOCTL_FAILED       (driver ioctl call failed)
 *****************************************************************************/
```

2.2.2 Ccurpwm_Clear_Driver_Error()

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

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

   Description: Clear any previously generated driver related error.

   Input:       void *Handle                        (handle pointer)
   Output:      None
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_IOCTL_FAILED       (driver ioctl call failed)
 ******************************************************************************/
```

2.2.3 Ccurpwm_Clear_Lib_Error()

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

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

   Description: Clear any previously generated library related error.

   Input:       void *Handle                        (handle pointer)
   Output:      None
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
 ******************************************************************************
```

2.2.4 Ccurpwm_Close()

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

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

   Description: Close a previously opened device.

   Input:       void *Handle                        (handle pointer)
   Output:      None
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
 ******************************************************************************/
```

2.2.5 Ccurpwm_Disable_Pci_Interrupts()

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

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

   Description: Disable interrupts being generated by the board.

   Input:       void *Handle                      (handle pointer)
   Output:      None
   Return:      CCURPWM_LIB_NO_ERROR            (successful)
                CCURPWM_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN            (device not open)
                CCURPWM_LIB_IOCTL_FAILED        (driver ioctl call failed)
 ******************************************************************************/
```

2.2.6 Ccurpwm_Enable_Pci_Interrupts()

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

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

   Description: Enable interrupts being generated by the board.

   Input:       void *Handle                      (handle pointer)
   Output:      None
   Return:      CCURPWM_LIB_NO_ERROR            (successful)
                CCURPWM_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN            (device not open)
                CCURPWM_LIB_IOCTL_FAILED        (driver ioctl call failed)
 ******************************************************************************/
```

2.2.7 Ccurpwm_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.

```
/************************************************************************
   Ccurpwm_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:      _ccurpwm_lib_error_number_t
                  - CCURPWM_LIB_NO_ERROR     (successful)
                  - CCURPWM_LIB_BAD_HANDLE   (no/bad handler supplied)
                  - CCURPWM_LIB_NOT_OPEN     (device not open)
 ***********************************************************************/
```

2.2.8 Ccurpwm_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.

```
/***********************************************************************
   Ccurpwm_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
 ************************************************************************/
```

2.2.9 Ccurpwm_Get_Driver_Error()

This call returns the last error generated by the driver.

```
/******************************************************************************
   int Ccurpwm_Get_Driver_Error(void *Handle, ccurpwm_user_error_t *ret_err)

   Description: Get the last error generated by the driver.

   Input:       void *Handle                     (handle pointer)
   Output:      ccurpwm_user_error_t *ret_err  (error struct pointer)
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_INVALID_ARG        (invalid argument)
                CCURPWM_LIB_IOCTL_FAILED       (driver ioctl call failed)
 ******************************************************************************/

#define CCURPWM_ERROR_NAME_SIZE    64
#define CCURPWM_ERROR_DESC_SIZE    128
typedef struct _ccurpwm_user_error_t {
    uint    error;                          /* error number */
    char    name[CCURPWM_ERROR_NAME_SIZE];   /* error name used in driver */
    char    desc[CCURPWM_ERROR_DESC_SIZE];   /* error description */
} ccurpwm_user_error_t;

enum    {
    CCURPWM_SUCCESS = 0,
    CCURPWM_INVALID_PARAMETER,
    CCURPWM_TIMEOUT,
    CCURPWM_OPERATION_CANCELLED,
    CCURPWM_RESOURCE_ALLOCATION_ERROR,
    CCURPWM_INVALID_REQUEST,
    CCURPWM_FAULT_ERROR,
    CCURPWM_BUSY,
    CCURPWM_ADDRESS_IN_USE,
    CCURPWM_DMA_TIMEOUT,
};
```

2.2.10 Ccurpwm_Get_Info()

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

```
/*****************************************************************************
   int Ccurpwm_Get_Info(void *Handle,  ccurpwm_driver_info_t *info)

   Description: Get device information from driver.

   Input:       void *Handle                   (handle pointer)
   Output:      ccurpwm_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:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_INVALID_ARG        (invalid argument)
                CCURPWM_LIB_IOCTL_FAILED       (driver ioctl call failed)
 *****************************************************************************/

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

#define CCURPWM_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                    Ccurpwm_Max_Region;/*kernel DEVICE_COUNT_RESOURCE*/

    ccurpwm_dev_region_t mem_region[CCURPWM_MAX_REGION];
} ccurpwm_driver_info_t;
```

2.2.11 Ccurpwm_Get_Lib_Error()

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

```
/******************************************************************************
   int Ccurpwm_Get_Lib_Error(void *Handle, ccurpwm_lib_error_t *lib_error)

   Description: Get last error generated by the library.

   Input:       void *Handle                     (handle pointer)
   Output:      ccurpwm_lib_error_t *lib_error (error struct pointer)
                -- uint error                    (error number)
                -- char name[CCURPWM_LIB_ERROR_NAME_SIZE] (error name)
                -- char desc[CCURPWM_LIB_ERROR_DESC_SIZE] (error description)
                -- int  line_number              (error line number in lib)
                -- char function[CCURPWM_LIB_ERROR_FUNC_SIZE]
                                                 (library function in error)
   Return:      CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                Last Library Error
 ******************************************************************************/

typedef struct _ccurpwm_lib_error_t {
    uint    error;                              /* lib error number */
    char    name[CCURPWM_LIB_ERROR_NAME_SIZE];  /* error name used in lib */
    char    desc[CCURPWM_LIB_ERROR_DESC_SIZE];  /* error description */
    int     line_number;                        /* line number in library */
    char    function[CCURPWM_LIB_ERROR_FUNC_SIZE];
                                                /* library function */
} ccurpwm_lib_error_t;
```

2.2.12 Ccurpwm_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 *ccurpwm_user.h* include file that is supplied with the driver.

```
/******************************************************************************
   int Ccurpwm_Get_Mapped_Config_Ptr(void *Handle,
                                      ccurpwm_config_local_data_t **config_ptr)

   Description: Get mapped configuration pointer.

   Input:       void *Handle                     (handle pointer)
   Output:      ccurpwm_config_local_data_t **config_ptr (config struct ptr)
                -- structure in ccurpwm_user.h
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_INVALID_ARG        (invalid argument)
                CCURPWM_LIB_NO_CONFIG_REGION   (config region not present)
 ******************************************************************************/
```

2.2.13 Ccurpwm_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 *ccurpwm_user.h* include file that is supplied with the driver.

```
/******************************************************************************
   int Ccurpwm_Get_Mapped_Local_Ptr(void *Handle,
                                  ccurpwm_local_ctrl_data_t **local_ptr)

   Description: Get mapped local pointer.

   Input:       void *Handle                     (handle pointer)
   Output:      ccurpwm_local_ctrl_data_t **local_ptr (local struct ptr)
                -- structure in ccurpwm_user.h
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_INVALID_ARG        (invalid argument)
                CCURPWM_LIB_NO_LOCAL_REGION    (local region not present)
 ******************************************************************************/
```

2.2.14 Ccurpwm_Get_Physical_Memory()

This call returns to the user the physical memory pointer and size that was previously allocated by the *Ccurpwm_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 Ccurpwm_Get_Physical_Memory(void *Handle,
                                 ccurpwm_phys_mem_t *phys_mem)

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

   Input:       void *Handle                     (handle pointer)
   Output:      ccurpwm_phys_mem_t *phys_mem   (mem struct pointer)
                -- void *phys_mem
                -- u_int phys_mem_size
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_INVALID_ARG        (invalid argument)
                CCURPWM_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 */
} ccurpwm_phys_mem_t;
```

2.2.15 Ccurpwm_Get_PWM()

This call returns to the user information about a specified wave. The user can specify either CCURPWM_WAVE_A or CCURPWM_WAVE_B.

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

   int Ccurpwm_Get_PWM(void *Handle, CCURPWM_WAVE wave, ccurpwm_wave_t *value)

   Description: Return the wave settings of the specified wave.

   Input:       void              *Handle       (handle pointer)
                CCURPWM_WAVE      wave          (which wave)
   Output:      ccurpwm_wave_t    *value;       (pointer to value)
   Return:      CCURPWM_LIB_NO_ERROR            (successful)
                CCURPWM_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN            (device not open)
                CCURPWM_LIB_INVALID_ARG         (invalid argument)

 *****************************************************************************/

typedef enum {
    CCURPWM_WAVE_A=1,
    CCURPWM_WAVE_B,
} CCURPWM_WAVE;



typedef struct
{
    u_int32_t   pwm_sine_frequency;     /* sine frequency */
    u_int32_t   pwm_phase_1;            /* phase 1  - 0 to 360 degrees */
    u_int32_t   pwm_phase_2;            /* phase 2  - 0 to 360 degrees */
    u_int32_t   pwm_phase_3;            /* phase 3  - 0 to 360 degrees */
    u_int32_t   pwm_deadband;           /* deadband */
    u_int32_t   pwm_PWM_frequency;      /* PWM frequency */
}  _ccurpwm_raw_wave_t;

typedef struct
{
    double      pwm_sine_frequency;     /* sine frequency */
    double      pwm_phase_1;            /* phase 1  - 0 to 360 degrees */
    double      pwm_phase_2;            /* phase 2  - 0 to 360 degrees */
    double      pwm_phase_3;            /* phase 3  - 0 to 360 degrees */
    u_int32_t   pwm_deadband;           /* deadband */
    double      pwm_PWM_frequency;      /* PWM frequency */
    _ccurpwm_raw_wave_t  raw;           /* raw data structure */
}  ccurpwm_wave_t;
```

2.2.16 Ccurpwm_Get_PWM_Individual()

This call allows the user to get the individual frequency and duty cycle.

```
/*****************************************************************************
   int Ccurpwm_Get_PWM_Individual(void *Handle, u_int32_t select,
                                   ccurpwm_individual_t *value)

   Description: Return the individual settings of the specified entry.

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

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

```
                 u_int32_t          select        (which individual)
   Output:       ccurpwm_individual_t *value;     (pointer to value)
   Return:       CCURPWM_LIB_NO_ERROR             (successful)
                 CCURPWM_LIB_BAD_HANDLE           (no/bad handler supplied)
                 CCURPWM_LIB_NOT_OPEN             (device not open)
                 CCURPWM_LIB_INVALID_ARG          (invalid argument)
 ******************************************************************************/
```

Select ranges from 0 to (PWM_MAX_PWM_FREQ_REGS-1) individual channels.

```
typedef struct
{
    u_int32_t   pwm_PWM_frequency;      /* PWM frequency */
    u_int32_t   pwm_duty;               /* duty cycle - 0 - 100% */
} _ccurpwm_raw_individual_t;

typedef struct
{
    double   pwm_PWM_frequency;         /* PWM frequency */
    double   pwm_duty;                  /* duty cycle - 0 - 100% */
    _ccurpwm_raw_individual_t raw;      /* raw data structure */
} ccurpwm_individual_t;
```

2.2.17 Ccurpwm_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 Ccurpwm_Get_Value(void *Handle, CCURPWM_CONTROL cmd, void *value)

   Description: Return the value of the specified board register.

   Input:       void              *Handle      (handle pointer)
                CCURPWM_CONTROL   cmd          (register definition)
   Output:      void              *value;      (pointer to value)
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_INVALID_ARG        (invalid argument)
                CCURPWM_LIB_NO_LOCAL_REGION    (local region not present)
 ******************************************************************************/

typedef enum {
    CCURPWM_STATUS,
    CCURPWM_REVISION,
    CCURPWM_RESYNC,
    CCURPWM_MODE,

    CCURPWM_A_SINE_FREQUENCY,
    CCURPWM_A_PHASE_1,
    CCURPWM_A_PHASE_2,
    CCURPWM_A_PHASE_3,
    CCURPWM_A_DEADBAND,
    CCURPWM_A_PWM_FREQUENCY,

    CCURPWM_B_SINE_FREQUENCY,
    CCURPWM_B_PHASE_1,
    CCURPWM_B_PHASE_2,
    CCURPWM_B_PHASE_3,
    CCURPWM_B_DEADBAND,
    CCURPWM_B_PWM_FREQUENCY,

    CCURPWM_INDIV0_PWM_FREQUENCY,
    CCURPWM_INDIV0_DUTY,
    CCURPWM_INDIV1_PWM_FREQUENCY,
    CCURPWM_INDIV1_DUTY,
    CCURPWM_INDIV2_PWM_FREQUENCY,
    CCURPWM_INDIV2_DUTY,
    CCURPWM_INDIV3_PWM_FREQUENCY,
    CCURPWM_INDIV3_DUTY,
    CCURPWM_INDIV4_PWM_FREQUENCY,
    CCURPWM_INDIV4_DUTY,
    CCURPWM_INDIV5_PWM_FREQUENCY,
    CCURPWM_INDIV5_DUTY,
    CCURPWM_INDIV6_PWM_FREQUENCY,
    CCURPWM_INDIV6_DUTY,
    CCURPWM_INDIV7_PWM_FREQUENCY,
    CCURPWM_INDIV7_DUTY,
    CCURPWM_INDIV8_PWM_FREQUENCY,
    CCURPWM_INDIV8_DUTY,
    CCURPWM_INDIV9_PWM_FREQUENCY,
    CCURPWM_INDIV9_DUTY,
    CCURPWM_INDIV10_PWM_FREQUENCY,
    CCURPWM_INDIV10_DUTY,
    CCURPWM_INDIV11_PWM_FREQUENCY,
    CCURPWM_INDIV11_DUTY,
} CCURPWM_CONTROL;
```

2.2.18 Ccurpwm_Initialize_Board()

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

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

   Description: Initialize the board.

   Input:       void *Handle                    (handle pointer)
   Output:      None
   Return:      CCURPWM_LIB_NO_ERROR          (successful)
                CCURPWM_LIB_BAD_HANDLE        (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN          (device not open)
                CCURPWM_LIB_IOCTL_FAILED      (driver ioctl call failed)
                CCURPWM_LIB_NO_LOCAL_REGION   (local region not present)
 ******************************************************************************/
```

2.2.19 Ccurpwm_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 *Ccurpwm_Munmap_Physical_Memory()* API call to remove the previously allocated physical memory.

```
/******************************************************************************
   int Ccurpwm_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:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_INVALID_ARG        (invalid argument)
                CCURPWM_LIB_MMAP_SELECT_FAILED (mmap selection failed)
                CCURPWM_LIB_MMAP_FAILED        (mmap failed)
 ******************************************************************************/
```

2.2.20 Ccurpwm_Munmap_Physical_Memory()

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

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

   Description: Unmap a previously mapped physical DMA memory.

   Input:       void *Handle                     (handle pointer)
   Output:      None
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_MUNMAP_FAILED      (failed to un-map memory)
                CCURPWM_LIB_NOT_MAPPED         (memory not mapped)
 ******************************************************************************/
```

2.2.21 Ccurpwm_NanoDelay()

This call simply delays (loops) for user specified nanoseconds.

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

   Description: Delay )loop for user specified nanoseconds.

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

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

2.2.22 Ccurpwm_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/ccurpwm<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 Ccurpwm_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:      CCURPWM_LIB_NO_ERROR              (successful)
                CCURPWM_LIB_INVALID_ARG           (invalid argument)
                CCURPWM_LIB_ALREADY_OPEN          (device already opened)
                CCURPWM_LIB_OPEN_FAILED           (device open failed)
                CCURPWM_LIB_ALREADY_MAPPED        (memory already mmapped)
                CCURPWM_LIB_MMAP_SELECT_FAILED (mmap selection failed)
                CCURPWM_LIB_MMAP_FAILED           (mmap failed)
 ******************************************************************************/
```

2.2.23 Ccurpwm_PWM_Resync()

This call issues a Resync command to the PWM.

```
/******************************************************************************
   Ccurpwm_PWM_Resync()

   Description: Issue resync command to the PWM

   Input:       void              *Handle        (handle pointer)
   Return:      CCURPWM_LIB_NO_ERROR             (successful)
                CCURPWM_LIB_BAD_HANDLE           (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN             (device not open)
                CCURPWM_LIB_INVALID_ARG          (invalid argument)
 ******************************************************************************/
```

2.2.24 Ccurpwm_Read()

This call is not supported for this card.

```
/******************************************************************************
   int Ccurpwm_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:      CCURPWM_LIB_NO_ERROR              (successful)
                CCURPWM_LIB_BAD_HANDLE            (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN              (device not open)
                CCURPWM_LIB_IO_ERROR              (read failed)
                CCURPWM_LIB_FIFO_OVERFLOW         (FIFO overflow)
 ******************************************************************************/
```

2.2.25 Ccurpwm_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 Ccurpwm_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:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_IOCTL_FAILED       (driver ioctl call failed)
 ******************************************************************************/
```

2.2.26 Ccurpwm_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 *Ccurpwm_Initialize_Board()* call.

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

   Description: Reset the board.

   Input:       void *Handle                     (handle pointer)
   Output:      None
   Return:      CCURPWM_LIB_NO_ERROR           (successful)
                CCURPWM_LIB_BAD_HANDLE         (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN           (device not open)
                CCURPWM_LIB_IOCTL_FAILED       (driver ioctl call failed)
                CCURPWM_LIB_NO_LOCAL_REGION    (local region not present)
 ******************************************************************************/
```

2.2.27 Ccurpwm_Set_PWM()

This call sets information for the specified wave.

```
/******************************************************************************
   int Ccurpwm_Set_PWM(void *Handle, CCURPWM_WAVE wave, ccurpwm_wave_t *value)

   Description: Set the wave parameters for the specified wave.

   Input:       void              *Handle       (handle pointer)
                CCURPWM_WAVE      wave          (which wave)
                ccurpwm_wave_t    *value;       (pointer to value)
   Return:      CCURPWM_LIB_NO_ERROR            (successful)
                CCURPWM_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN            (device not open)
                CCURPWM_LIB_INVALID_ARG         (invalid argument)
 ******************************************************************************/

typedef enum {
    CCURPWM_WAVE_A=1,
    CCURPWM_WAVE_B,
} CCURPWM_WAVE;

typedef struct
{
    u_int32_t   pwm_sine_frequency;     /* sine frequency */
    u_int32_t   pwm_phase_1;            /* phase 1  - 0 to 360 degrees */
    u_int32_t   pwm_phase_2;            /* phase 2  - 0 to 360 degrees */
    u_int32_t   pwm_phase_3;            /* phase 3  - 0 to 360 degrees */
    u_int32_t   pwm_deadband;           /* deadband */
    u_int32_t   pwm_PWM_frequency;      /* PWM frequency */
}  _ccurpwm_raw_wave_t;

typedef struct
{
    double      pwm_sine_frequency;     /* sine frequency */
    double      pwm_phase_1;            /* phase 1  - 0 to 360 degrees */
    double      pwm_phase_2;            /* phase 2  - 0 to 360 degrees */
    double      pwm_phase_3;            /* phase 3  - 0 to 360 degrees */
    u_int32_t   pwm_deadband;           /* deadband */
    double      pwm_PWM_frequency;      /* PWM frequency */
    _ccurpwm_raw_wave_t  raw;           /* raw data structure */
}  ccurpwm_wave_t;
```

2.2.28 Ccurpwm_Set_PWM_Individual()

This call allows the user to set the individual frequency and duty cycle.

```
/******************************************************************************
   int Ccurpwm_Set_PWM_Individual(void *Handle, u_int32_t select,
                                  ccurpwm_individual_t *value)

   Description: Set the individual settings for the specified entry.

   Input:       void              *Handle       (handle pointer)
                u_int32_t          select       (which individual)
                ccurpwm_individual_t *value;    (pointer to value)
   Return:      CCURPWM_LIB_NO_ERROR            (successful)
                CCURPWM_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN            (device not open)
                CCURPWM_LIB_INVALID_ARG         (invalid argument)
 ******************************************************************************/
```

Select ranges from 0 to (PWM_MAX_PWM_FREQ_REGS-1) individual channels.

```
typedef struct
{
    u_int32_t   pwm_PWM_frequency;      /* PWM frequency */
    u_int32_t   pwm_duty;               /* duty cycle - 0 - 100% */
} _ccurpwm_raw_individual_t;

typedef struct
{
    double   pwm_PWM_frequency;         /* PWM frequency */
    double   pwm_duty;                  /* duty cycle - 0 - 100% */
    _ccurpwm_raw_individual_t raw;      /* raw data structure */
} ccurpwm_individual_t;
```

2.2.29 Ccurpwm_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 Ccurpwm_Set_Value(void *Handle, CCURPWM_CONTROL cmd, int value)

   Description: Set the value of the specified board register.

   Input:       void *Handle                    (handle pointer)
                CCURPWM_CONTROL cmd             (register definition)
                int value                       (value to be set)
   Output:      None
   Return:      CCURPWM_LIB_NO_ERROR            (successful)
                CCURPWM_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN            (device not open)
                CCURPWM_LIB_INVALID_ARG         (invalid argument)
 ******************************************************************************/

typedef enum {
    CCURPWM_STATUS,
    CCURPWM_REVISION,
    CCURPWM_RESYNC,
    CCURPWM_MODE,

    CCURPWM_A_SINE_FREQUENCY,
    CCURPWM_A_PHASE_1,
    CCURPWM_A_PHASE_2,
    CCURPWM_A_PHASE_3,
    CCURPWM_A_DEADBAND,
    CCURPWM_A_PWM_FREQUENCY,

    CCURPWM_B_SINE_FREQUENCY,
    CCURPWM_B_PHASE_1,
    CCURPWM_B_PHASE_2,
    CCURPWM_B_PHASE_3,
    CCURPWM_B_DEADBAND,
    CCURPWM_B_PWM_FREQUENCY,

    CCURPWM_INDIV0_PWM_FREQUENCY,
    CCURPWM_INDIV0_DUTY,
    CCURPWM_INDIV1_PWM_FREQUENCY,
    CCURPWM_INDIV1_DUTY,
    CCURPWM_INDIV2_PWM_FREQUENCY,
    CCURPWM_INDIV2_DUTY,
    CCURPWM_INDIV3_PWM_FREQUENCY,
    CCURPWM_INDIV3_DUTY,
    CCURPWM_INDIV4_PWM_FREQUENCY,
    CCURPWM_INDIV4_DUTY,
    CCURPWM_INDIV5_PWM_FREQUENCY,
    CCURPWM_INDIV5_DUTY,
    CCURPWM_INDIV6_PWM_FREQUENCY,
    CCURPWM_INDIV6_DUTY,
    CCURPWM_INDIV7_PWM_FREQUENCY,
    CCURPWM_INDIV7_DUTY,
    CCURPWM_INDIV8_PWM_FREQUENCY,
    CCURPWM_INDIV8_DUTY,
    CCURPWM_INDIV9_PWM_FREQUENCY,
    CCURPWM_INDIV9_DUTY,
    CCURPWM_INDIV10_PWM_FREQUENCY,
    CCURPWM_INDIV10_DUTY,
    CCURPWM_INDIV11_PWM_FREQUENCY,
    CCURPWM_INDIV11_DUTY,
} CCURPWM_CONTROL;
```

2.2.30 Ccurpwm_Write()

This call is not supported for this card.

```
/*******************************************************************************
   int Ccurpwm_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:      CCURPWM_LIB_NO_ERROR            (successful)
                CCURPWM_LIB_BAD_HANDLE          (no/bad handler supplied)
                CCURPWM_LIB_NOT_OPEN            (device not open)
                CCURPWM_LIB_IO_ERROR            (write failed)
                CCURPWM_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 ccurpwm_dump

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

Usage: ccurpwm_dump -b<device number>

Example display:

```
Device Name    : /dev/ccurpwm0
LOCAL Register 0x7ffff7ff5000 Offset=0x0
CONFIG Register 0x7ffff7ff4000 Offset=0x0

======= LOCAL BOARD REGISTERS =========
LBR: @0x0000 --> 0x00010000
LBR: @0x000c --> 0x00000000
...
LBR: @0x07d0 --> 0x00000000
LBR: @0x07dc --> 0x00000000
...
LBR: @0x13f0 --> 0x00000000
LBR: @0x13fc --> 0x00000000

======= LOCAL CONFIG REGISTERS =========
LCR: @0x0000 --> 0xffff8000
LCR: @0x0004 --> 0x00000001
...
LCR: @0x00f4 --> 0x00000000
LCR: @0x00f8 --> 0x00000043

======= PCI CONFIG REG ADDR MAPPING =========
PCR: @0x0000 --> 0x92721542
PCR: @0x0004 --> 0x02b00117
...
PCR: @0x004c --> 0x00000003
PCR: @0x0050 --> 0x00000000

======= PCI BRIDGE REGISTERS =========
PBR: @0x0000 --> 0x811110b5
PBR: @0x0004 --> 0x00100117
...
PBR: @0x010c --> 0x00000000
PBR: @0x0110 --> 0x00000000

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

3.1.2 ccurpwm_reg

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

Usage: ccurpwm_reg -b<device number>

Example display:

```
Device Name: /dev/ccurpwm0
LOCAL Register 0xb7ff8000 Offset=0x0

#### LOCAL REGS #### (length=32768)
+LCL+        0   00010000  00020000  00000000  00000000 *................*
+LCL+     0x10   00000000  00000000  00000000  00000000 *................*
...
+LCL+   0x7fe0   00000000  00000000  00000000  00000000 *................*
+LCL+   0x7ff0   00000000  00000000  00000000  00000000 *................*


CONFIG Register 0xb7ff7c00 Offset=0xc00
```

```
#### CONFIG REGS #### (length=512)
+CFG+        0   ffff8000  00000001  00200000  00000400 *......... ......*
+CFG+     0x10   00000000  00000011  f20301db  00000000 *................*
...
+CFG+    0x1e0   00000000  00000000  00000000  00000000 *................*
+CFG+    0x1f0   00000000  00000000  00000000  00000000 *................*

======= LOCAL REGISTERS =========
    pwm_status               =0x00010000        @0x00000000
    pwm_revision             =0x00020000        @0x00000004
    pwm_resync               =0x00000000        @0x00001000
    pwm_mode                 =0x00000000        @0x00001004
    pwm_a_sine_frequency     =0x00000000        @0x00001100
    pwm_a_phase_1            =0x00000000        @0x00001104
    pwm_a_phase_2            =0x00000000        @0x00001108
    pwm_a_phase_3            =0x00000000        @0x0000110c
    pwm_a_deadband           =0x00000000        @0x00001110
    pwm_a_PWM_frequency      =0x00000000        @0x00001114
    pwm_b_sine_frequency     =0x00000000        @0x00001130
    pwm_b_phase_1            =0x00000000        @0x00001134
    pwm_b_phase_2            =0x00000000        @0x00001138
    pwm_b_phase_3            =0x00000000        @0x0000113c
    pwm_b_deadband           =0x00000000        @0x00001140
    pwm_b_PWM_frequency      =0x00000000        @0x00001144
    pwm_indiv0.pwm_PWM_frequency=0x00000000     @0x00001220
    pwm_indiv0.pwm_duty      =0x00000000        @0x00001224
    pwm_indiv1.pwm_PWM_frequency=0x00000000     @0x00001228
    pwm_indiv1.pwm_duty      =0x00000000        @0x0000122c
    pwm_indiv2.pwm_PWM_frequency=0x00000000     @0x00001230
    pwm_indiv2.pwm_duty      =0x00000000        @0x00001234
    pwm_indiv3.pwm_PWM_frequency=0x00000000     @0x00001238
    pwm_indiv3.pwm_duty      =0x00000000        @0x0000123c
    pwm_indiv4.pwm_PWM_frequency=0x00000000     @0x00001240
    pwm_indiv4.pwm_duty      =0x00000000        @0x00001244
    pwm_indiv5.pwm_PWM_frequency=0x00000000     @0x00001248
    pwm_indiv5.pwm_duty      =0x00000000        @0x0000124c
    pwm_indiv6.pwm_PWM_frequency=0x00000000     @0x00001250
    pwm_indiv6.pwm_duty      =0x00000000        @0x00001254
    pwm_indiv7.pwm_PWM_frequency=0x00000000     @0x00001258
    pwm_indiv7.pwm_duty      =0x00000000        @0x0000125c
    pwm_indiv8.pwm_PWM_frequency=0x00000000     @0x00001260
    pwm_indiv8.pwm_duty      =0x00000000        @0x00001264
    pwm_indiv9.pwm_PWM_frequency=0x00000000     @0x00001268
    pwm_indiv9.pwm_duty      =0x00000000        @0x0000126c
    pwm_indiv10.pwm_PWM_frequency=0x00000000        @0x00001270
    pwm_indiv10.pwm_duty     =0x00000000        @0x00001274
    pwm_indiv11.pwm_PWM_frequency=0x00000000        @0x00001278
    pwm_indiv11.pwm_duty     =0x00000000        @0x0000127c

======= 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                =0x00000003       @0x00000080
dmapadr0                =0x00000000       @0x00000084
dmaladr0                =0x00000000       @0x00000088
dmasiz0                 =0x00000000       @0x0000008c
dmadpr0                 =0x00000000       @0x00000090
dmamode1                =0x00000003       @0x00000094
dmapadr1                =0x00000000       @0x00000098
dmaladr1                =0x00000000       @0x0000009c
dmasiz1                 =0x00000000       @0x000000a0
dmadpr1                 =0x00000000       @0x000000a4
dmacsr0                 =0x00001010       @0x000000a8
dmacsr1                 =0x00200000       @0x000000ac
las1rr                  =0x00000000       @0x000000f0
las1ba                  =0x00000000       @0x000000f4
lbrd1                   =0x00000043       @0x000000f8
```

3.1.3 ccurpwm_tst

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

Usage: ccurpwm_tst -b<device number>

Example display:

```
Device Name: /dev/ccurpwm0
Initialize_Board: Firmware Rev. 0x20000 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)->
```

3.1.4 ccurpwm_rdreg

This is a simple program that reads registers by address.

Usage: ccurpwm_rdreg -b<device number> -o<offset> -s<size>

Example display:

```
Device Name    : /dev/ccurpwm0

#### LOCAL REGS #### (length=4)
+LCL+       0   00010000                  *....          *
```

3.1.5 ccurpwm_wreg

This is a simple program that writes registers by address.

Usage: ccurpwm_wreg -b<device number> -o<offset> -s<size>

Example display:

```
Device Name    : /dev/ccurpwm0
Writing 0x00000000 to offset 0x0000 for 4 bytes

#### LOCAL REGS #### (length=4)
+LCL+       0   00010000                  *....          *
```

3.2 Application Program Interface (API) Access Example Test

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

3.2.1 ccurpwm_tst_lib

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

```
Usage: ccurpwm_tst_lib <device number>
```

Example display:

```
  01 = Add Irq                         02 = Clear Driver Error
  03 = Clear Library Error             04 = Disable Pci Interrupts
  05 = Display BOARD Registers         06 = Enable Pci Interrupts
  07 = Get Information                 08 = Get Driver Error
  09 = Get Library Error               10 = Get Mapped Config Pointer
  11 = Get Mapped Local Pointer        12 = Get Physical Memory
  13 = Get PWM                         14 = Get PWM Individual
  15 = Get Value                       16 = Initialize Board
  17 = MMap Physical Memory            18 = Munmap Physical Memory
  19 = PWM Resync                      20 = Read Operation
  21 = Remove Irq                      22 = Reset Board
  23 = Set PWM                         24 = Set PWM Individual
  25 = Set Value                       26 = Test Registers
  27 = Write Operation

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

This page intentionally left blank