DYNAMIC ENGINEERING

150 DuBois, Suite B/C

Santa Cruz, CA 95060
(831) 457-8891
https://www.dyneng.com
sales@dyneng.com
Est. 1988

(cc)PMC-BiSerial-VI-ORN1
Windows 10 WDF Driver Documentation

Developed with Windows Driver Foundation
Ver1.19

Revision 01p1 7/7/22
ccPMC: 10-2021-0401/2

ccPMC-BiSerial-VI-ORN1
WDF Device Drivers

Dynamic Engineering
150 DuBois, Suite B/C
Santa Cruz, CA 95060
(831) 457-8891

Table of Contents

Introduction

The ccPMC-BiSerial-VI-ORN1 driver was developed with the Windows Driver Foundation version 1.19 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF). The driver files are fully signed under the current requirements.

ccPMC-BiSerial-VI-ORN1 features a Spartan6 Xilinx FPGA to implement the PCI interface, FIFOs, and IO processing, control and status for 32 differential IO. Each IO can be RS-485 or LVDS (build option). There is a programmable PLL with four clock outputs. PLLA is defined as the SDLC receive reference. PLLB is the SDLC transmit reference when internal clock mode is in use. PLLC is used as a reference for the NRZL interfaces. The DDR is not in use on this design. The temperature and switch interfaces are supported with this driver.

UserAp is a stand-alone code set with a simple and powerful menu plus a series of tests that can be run on the installed hardware. Each of the tests execute calls to the driver, pass parameters and structures, and get results back. With the sequence of calls demonstrated, the functions of the hardware are utilized for loop-back testing. The software is used for manufacturing test at Dynamic Engineering. The test software can be ported to your application to provide a running start. The tests are simple and will quickly demonstrate the end-to-end operation of your application making calls to the driver and interacting with the hardware.

The menu allows the user to add tests, to run sequences of tests, to run until a failure occurs and stop or to continue, to program a set number of loops to execute and more. The user can add tests to the provided test suite to try out application ideas before committing to your system configuration. In many cases the test configuration will allow faster debugging in a more controlled environment before integrating with the rest of the system. The test menu uses the Type field to know which board type it is communicating with and prints that out at the top of the menu. The .inf file also has the definitions and the system will show the type in the device manager after installation of the driver.

When BiSerial-VI-ORN1 is recognized by the PCI bus configuration utility it will start the ccPMC-BiSerial-VI-ORN1 driver to allow communication with the device. IO Control calls (IOCTLs) are used to configure the board and read status. The driver is hierarchical with “base” and “chan” drivers to support each of the functions.

Note
This documentation will provide information about all calls made to the drivers, and how the drivers interact with the device for each of these calls. ***For more detailed information on the hardware implementation,*** refer to the ccPMC-BiSerial-VI-ORN1 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include ccPB6Orn1Base.cat, ccPB6Orn1Base.inf, ccPB6Orn1Base.sys, ccPB6Orn1Chan.cat, ccPB6Orn1Chan.inf, ccPB6Orn1Chan.sys, plus the public files: ccPB6Orn1BasePublic.h, ccPB6Orn1ChanPublic.h, and ccPB6Orn1Public.h.

The Base and Chan Public.h files are the C header files that defines the Application Program Interface (API) for the ccPMC-BiSerial-VI-ORN1 driver. These files are required at compile time by any application that wishes to interface with the drivers, but is not needed for driver installation. Included with the UserAp file set. Driver files also included in the UserAp.zip file set.

Windows 10 Installation

Copy the system files (6) to a CD, USB memory device, or local directory as preferred.

With the hardware installed, power-on the host computer.
- Open the ***Device Manager*** from the control panel.
- Under ***Other devices*** there should be an ***Other PCI Bridge Device****.
- Right-click on the ***Other PCI Bridge Device*** and select ***Update Driver Software***.

- Select ***Browse my computer for driver software***.
- Select ***Navigate to the folder or device. If at the root select the sub folders button.***
- Select ***Next***.
- Select ***Close*** to close the update window.

The system should now display the ORN1 adapter in the Device Manager.

Repeat to install the channel drivers.

***** If the ***Other PCI Bridge Device*** is not displayed, click on the ***Scan for hardware changes*** icon on the tool-bar.

Driver Startup

Once the driver has been installed it will start automatically when the system recognizes the hardware.

A handle can be opened to a specific board by using the CreateFile() function call and passing in the device name obtained from the system.

The interface to the device is identified using globally unique identifiers (GUID), which are defined in ccPB6Orn1BasePublic.h. See main.c in the ccPB6Orn1UserAp project for an example of how to acquire a handle to the base and ports.

The main file is designed to work with our test menu and includes user interaction steps to allow the user to select which board is being tested in a multiple board environment. The integrator can hardcode for single board systems or use an automatic loop to operate in multiple board systems without using user interaction. For multiple user systems it is suggested that the board number is associated with a switch setting so the calls can be associated with a particular board from a physical point of view.

IO Controls

The drivers use IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object, which controls a single board or I/O channel. IOCTLs are called using the Win function DeviceIoControl(), and passing in the handle to the device opened with CreateFile() (see above). IOCTLs generally have input parameters, output parameters, or both. Often a custom structure is used.

```
BOOL DeviceIoControl(
  HANDLE        hDevice,          // Handle opened with CreateFile()
  DWORD         dwIoControlCode, // Control code defined in API header
file
  LPVOID        lpInBuffer,       // Pointer to input parameter
  DWORD         nInBufferSize,    // Size of input parameter
  LPVOID        lpOutBuffer,      // Pointer to output parameter
  DWORD         nOutBufferSize,   // Size of output parameter
  LPDWORD       lpBytesReturned, // Pointer to return length parameter
  LPOVERLAPPED lpOverlapped,     // Optional pointer to overlapped
structure
);                                //   used for asynchronous I/O
```

The IOCTLs defined for the -ORN1 driver are described below:
Please note: some IOCTLs are defined but not used as DDR is not supported in this design.

IOCTL_ccPB6Orn1_BASE_GET_INFO

Function: Returns the device driver version, Xilinx flash revision, PLL device ID, user switch value, Type, and device instance number.
Input: None
Output: ccPB6Orn1_BASE_DRIVER_DEVICE_INFO structure
Notes: The switch value is the configuration of the 8-bit onboard dipswitch that has been selected by the user (see the board silk screen for bit position and polarity). Instance number is the zero-based device number. Revision Major and Revision Minor represent the current Flash revision Major.Minor. PLL Device ID is the I2C address discovered.

```
// Driver/Device information
typedef struct _ccPB6Orn1_BASE_DRIVER_DEVICE_INFO {
   UCHAR    DriverRev;
   UCHAR    DesignRev;
   UCHAR    DesignRevMin;
   UCHAR    DesignType;
   ULONG    InstanceNum;
   UCHAR    SwitchValue;
   UCHAR    PllDeviceId;
   BOOLEAN  BridgeCnfgd;
} ccPB6Orn1_BASE_DRIVER_DEVICE_INFO, *PccPB6Orn1_BASE_DRIVER_DEVICE_INFO;
```

IOCTL_ccPB6Orn1_LOAD_PLL_DATA

Function: Writes to the internal registers of the PLL.
Input: ccPB6Orn1_BASE_PLL_DATA structure
Output: None
Notes: The structure has only one field: Data – an array of 40 bytes containing the PLL register data to write. See below for the definition.

```
 // Structures for IOCTLs
#define PLL_MESSAGE1_SIZE        16
#define PLL_MESSAGE2_SIZE        24
#define PLL_MESSAGE_SIZE        (PLL_MESSAGE1_SIZE + PLL_MESSAGE2_SIZE)

typedef struct _ccPB6Orn1_BASE_PLL_DATA {
   UCHAR    Data[PLL_MESSAGE_SIZE];
} ccPB6Orn1_BASE_PLL_DATA, *PccPB6Orn1_BASE_PLL_DATA;
```

IOCTL_ccPB6Orn1_READ_PLL_DATA

Function: Reads and returns the contents of the internal registers of the PLL.
Input: None
Output: ccPB6Orn1_BASE_PLL_DATA structure
Notes: The PLL register data is returned in the structure in an array of 40 bytes. See definition of ccPB6Orn1_BASE_PLL_DATA above.

IOCTL_ccPB6Orn1_SET_BASE_CONFIG

Function: Writes the base configuration register on the Parallel-TTL-ORN1.
Input: PAR_TTL_ORN1_SET_CONFIG structure
Output: None
Notes: The Base Configuration register data is set with the PAR_TTL_ORN1_SET_CONFIG structure.

```
typedef struct _PAR_TTL_ORN1_BASE_SET_CONFIG {
        BOOLEAN    IoRst; // set to cause reset, return to cleared required to operate
        BOOLEAN    ForceIntEn; // set to force interrupt, clear to remove
        BOOLEAN    CosClkSel; // set to use PLL, cleared uses oscillator reference
        BOOLEAN    TestClkSel; // not set = std operation, set = push reference clock onto IO
} PAR_TTL_ORN1_BASE_SET_CONFIG, *PPAR_TTL_ORN1_BASE_SET_CONFIG;
```

IOCTL_ccPB6Orn1_BASE_GET_STATUS

Function: Returns the status register value
Input: None
Output: Value of status register (unsigned long integer)
Notes: Returns Base level status See HW manual for detail about the meaning of the bits.

IOCTL_ccPB6Orn1_BASE_RESET

Function: Returns the status register value
Input: None
Output: none
Notes: Causes a reset.

IOCTL_ccPB6Orn1_BASE_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to the Event object
Output: None
Notes: The user creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when a user interrupt is serviced by the driver. The user-defined interrupt service routine waits on this event, allowing it to respond to the interrupt. The DMA interrupts do not cause the event to be signaled unless they are explicitly enabled in the enable interrupts call.

IOCTL_ccPB6Orn1_BASE_ENABLE_INTERRUPT

Function: Enables the Master interrupt at the base level.
Input: none
Output: None
Notes: Required to be enabled to pass interrupts from the ports to the host. With the Master Interrupt Enable disabled the Port interrupts can be polled if desired.

IOCTL_ccPB6Orn1_BASE_DISABLE_INTERRUPT

Function: Disables the Master interrupt.
Input: none
Output: None
Notes: This call is used when interrupt processing is no longer desired.

IOCTL_ccPB6Orn1_BASE_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted as long as the master interrupt is enabled. This IOCTL is used for development, to test interrupt processing. Force Interrupt is automatically cleared by the ISR/DPC.

IOCTL_ccPB6Orn1_BASE_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: ccPB6Orn1_BASE_ISR_STAT structure

Notes: Returns the interrupt status that was read in the interrupt service routine of the last interrupt caused by one of the enabled interrupt conditions.

```
typedef struct _ccPB6Orn1_BASE_ISR_STAT {
        ULONG    Status;
        BOOLEAN  New;
} ccPB6Orn1_BASE_ISR_STAT, * PccPB6Orn1_BASE_ISR_STAT;
```

IOCTL_ccPB6Orn1_BASE_BRIDGE_RECONFIG

Function: Look for upstream bridge and reprogram
Input: None
Output: None
Notes: Creates a work item that looks for an upstream bridge. For example, if the XMC is used the bridge is on the XMC. If the PMC is used with PCIeBPMCX1 the bridge is on the carrier. Certain settings are modified to enhance DMA performance. To see if configuration was successful [BridgeConfigured] check that status. Since the work item operates in parallel allow for this call to complete. Example in the menu. Not required for this design as no DMA implemented. Menu prints status of Bridge programming for reference.

IOCTL_ccPB6Orn1_BASE_ENABLE_TSTCLK

Function: Enable Test Clock generation
Input: None
Output: None
Notes: Enables the test clock in place of the parallel port. If the mux selects the test clock it can be used to check the differential IO operation. See example in test menu.

IOCTL_ccPB6Orn1_BASE_DISABLE_TSTCLK

Function: Disable Test Clock generation
Input: None
Output: None
Notes: Disables the test clock. See example in test menu.

IOCTL_ccPB6Orn1_BASE_SET_DATA_OUT0

Function: Writes a single 32-bit data-word to the Data Register
Input: ULONG
Output: None
Notes: If the IO is selected in the data mux data will flow to the output based on the Direction Registers.

IOCTL_ccPB6Orn1_BASE_GET_DATA_OUT0

Function: Reads and returns a single 32-bit data word from the Data Register.
Input: None
Output: ULONG
Notes: This is the register read-back and will match the SET data.

IOCTL_ccPB6Orn1_BASE_SET_DIR0

Function: Writes a single 32-bit data-word to the Direction Register
Input: ULONG
Output: None
Notes: For each bit set to '1' the data from the register is enabled onto the bus to the external transceivers and the transceiver is enabled to transmit. When '0' the transceiver is configured to receive and the register data is isolated from the bus. See Read Direct call.

IOCTL_ccPB6Orn1_BASE_GET_DIR0

Function: Reads and returns a single 32-bit data word from the Data Enable Register.
Input: None
Output: ULONG
Notes:

IOCTL_ccPB6Orn1_BASE_SET_TERM0

Function: Writes a single 32-bit data-word to the Termination Register
Input: ULONG
Output: None
Notes: For each bit set to '1' the bit will be terminated. See UserAp and HW manual for more information.

IOCTL_ccPB6Orn1_BASE_GET_TERM0

Function: Reads and returns a single 32-bit data word from the Termination Register.
Input: None
Output: ULONG
Notes:

IOCTL_ccPB6Orn1_BASE_SET_MUX0

Function: Writes a single 32-bit data-word to the Mux Register
Input: ULONG
Output: None
Notes: For each bit set to '1' the programmed port operation will be used. For bits programmed to '0' the parallel port definition is used. For ports using more than 1 IO all bits need to be set. For example Port 0 uses IO 0,1,2,3. See UserAp for examples.

IOCTL_ccPB6Orn1_BASE_GET_MUX0

Function: Reads and returns a single 32-bit data word from the Mux Register.
Input: None
Output: ULONG
Notes:

IOCTL_ccPB6Orn1_BASE_READ_DIRECT0

Function: Reads and returns a single 32-bit data word from the IO port.
Input: None
Output: ULONG
Notes: Direct data is synchronized but not filtered in any way. Get the state of the IO (whether defined as output or input).

IOCTL_ccPB6Orn1_BASE_SET_TMP

Function: Write control word to Temperature interface
Input: ULONG
Output: none
Notes: The temperature interface is in hardware with the frequency, serialization etc. handled there. Control words are written to request data. The Get version of the call is used to poll for the updated data and retrieve the data. See the HW manual for the bit map. Public files have bit definitions, see UserAp for example of using the interface and converting the data.

IOCTL_ccPB6Orn1_BASE_GET_TMP

Function: Reads and returns a single 32-bit data word from the IO port.
Input: none
Output: ULONG
Notes:

Port Interface Common

IOCTL_ccPB6Orn1_CHAN_GET_INFO

Function: Returns the device driver version, user switch value, Type, and device instance number.
Input: None
Output: ccPB6Orn1_CHAN_DRIVER_DEVICE_INFO structure
Notes: The switch value is the configuration of the 8-bit onboard dipswitch that has been selected by the user (see the board silk screen for bit position and polarity). Instance number is the zero-based device number. Revision Major and Revision Minor represent the current Flash revision Major.Minor. PLL Device ID is the I2C address discovered.

```
// Driver/Device information
typedef struct _ccPB6Orn1_CHAN_DRIVER_DEVICE_INFO {
   UCHAR    DriverRev;
   UCHAR    ChannelNum;
   UCHAR    DesignRev;     // Design revision from base driver
   UCHAR    DesignRevMin;  // Design minor revision from base driver
   UCHAR    DesignType;    // Design type from base driver
   UCHAR    SwitchValue;   // Board user switch value from base driver
   ULONG    InstanceNum;   // Board instance from base driver
} ccPB6Orn1_CHAN_DRIVER_DEVICE_INFO, *PccPB6Orn1_CHAN_DRIVER_DEVICE_INFO;
```

IOCTL_ccPB6Orn1_CHAN_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to the Event object
Output: None
Notes: The user creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when a user interrupt is serviced by the driver. The user-defined interrupt service routine waits on this event, allowing it to respond to the interrupt. The DMA interrupts do not cause the event to be signaled unless they are explicitly enabled in the enable interrupts call.

IOCTL_ccPB6Orn1_CHAN_ENABLE_INTERRUPT

Function: Enables the Master interrupt at the port level.
Input: none
Output: None
Notes: Required to be enabled to pass interrupts from the ports to the Base level. With the Port Master Interrupt Enable disabled the Port status can be polled if desired.

IOCTL_ccPB6Orn1_CHAN_DISABLE_INTERRUPT

Function: Disables the Master interrupt for the port
Input: none
Output: None
Notes: This call is used when interrupt processing is no longer desired.

IOCTL_ccPB6Orn1_CHAN_FORCE_INTERRUPT

Function: Causes a system interrupt to occur from the port.
Input: None
Output: None
Notes: Causes an interrupt to be asserted as long as the master enable is enabled. This IOCTL is used for development, to test interrupt processing. Force Interrupt is automatically cleared by the ISR/DPC.

SDLC Port

IOCTL_ccPB6Orn1_CHAN_SDLC_WRITEFILE

Function: Write multiple data words to SDLC DPR
Input: TRANS_MULT
Output: none
Notes: Select Tx or Rx DPR,number of words to write, and array to load. See UserAp for reference.

IOCTL_ccPB6Orn1_CHAN_SDLC_READFILE

Function: Read multiple data words from SDLC DPR
Input: TRANS_MULT
Output: TRANS_MULT
Notes: Select Tx or Rx DPR,number of words to read. Array is returned. See UserAp for reference.

IOCTL_ccPB6Orn1_CHAN_SDLC_SET_CONTROL

Function: Write structure to SDLC control Register
Input: SDLC CHAN CNTL
Output: none
Notes: See HW manual for bit definitions. See UserAp for examples of use.

```
typedef struct _SDLC_CHAN_CNTL {
   BOOLEAN  TxEnable;
   BOOLEAN  RxEnable;
   BOOLEAN  TxExtClk;
   BOOLEAN  TxClearEnable;
   BOOLEAN  TxIntEnable;
   BOOLEAN  TxDnIntEnable;
   BOOLEAN  RxIntEnable;
   BOOLEAN  AbortIntEnable;
   BOOLEAN  TxIdleFrmEnd;
   BOOLEAN  TxFlgsShrZero;
   BOOLEAN  SendAbort;
   USHORT   RxStartAddress;
   BOOLEAN  LoadRxStartAddr;
   USHORT   TxStartAddress;
   BOOLEAN  LoadTxStartAddr;
   USHORT   TxEndAddress;
   BOOLEAN  LoadTxEndAddr;
} SDLC_CHAN_CNTL, * PSDLC_CHAN_CNTL;
```

IOCTL_ccPB6Orn1_CHAN_SDLC_GET_STATE

Function: Read from SDLC control register which also includes status information
Input:
Output: SDLC CHAN STATE
Notes: See HW manual for bit definitions. See UserAp for examples of use.

```
typedef struct _SDLC_CHAN_STATE {
   BOOLEAN  TxEnable;
   BOOLEAN  RxEnable;
   BOOLEAN  TxExtClk;
   BOOLEAN  TxSndngFrm;
   BOOLEAN  TxFrmDone;
   BOOLEAN  TxClearEnable;
   BOOLEAN  TxIntEnable;
   BOOLEAN  TxDnIntEnable;
   BOOLEAN  RxIntEnable;
   BOOLEAN  AbortIntEnable;
   BOOLEAN  TxFlgsShrZero;
   BOOLEAN  TxIdleFrmEnd;
   USHORT   RxEndAddress;
   BOOLEAN  AbortReceived;
   BOOLEAN  IdleDetected;
} SDLC_CHAN_STATE, * PSDLC_CHAN_STATE;
```

IOCTL_ccPB6Orn1_CHAN_SDLC_LOAD

Function: Write a LW to SDLC DPR
Input: SDLC_WRITE_WORD
Output: none
Notes:

```
typedef struct _SDLC_WRITE_WORD {
   DPR_BANK Bank;   // select Transmit or Receive memory bank
   ULONG    Offset; // Offset Relative to channel, bank start LW count
   ULONG    Data;   // Data to load to address
} SDLC_WRITE_WORD, * PSDLC_WRITE_WORD;
```

IOCTL_ccPB6Orn1_CHAN_SDLC_READ

Function: Read a LW from SDLC DPR
Input: SDLC_READ_WORD
Output: SDLC_READ_WORD
Notes:

```
typedef struct _SDLC_READ_WORD {
   DPR_BANK Bank;   // select Transmit or Receive memory bank
   ULONG    Offset; // Offset Relative to channel, bank start LW count
   ULONG    Data;   // Data read from address
} SDLC_READ_WORD, * PSDLC_READ_WORD;
```

IOCTL_ccPB6Orn1_CHAN_GET_SDLC_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: ccPB6Orn1_CHAN_ISR_STAT structure
Notes: Returns the interrupt status that was read in the interrupt service routine of the last interrupt caused by one of the enabled interrupt conditions.

```
typedef struct _ccPB6Orn1_CHAN_ISR_STAT {
  ULONG    Status;
  BOOLEAN  New;
} ccPB6Orn1_CHAN_ISR_STAT, *PccPB6Orn1_CHAN_ISR_STAT;
```

NRZL Port

IOCTL_ccPB6Orn1_CHAN_NRZL_WRM_FIFO

Function: Write multiple data words to NRZL TX Data FIFO
Input: FIFO_MULT
Output: none
Notes: Select Count to load and provide data in array. See UserAp for reference.

```
typedef struct _FIFO_MULT
{
   ULONG    Count;        // number of LWs to Read/Write
   ULONG    Data[NRZL_FIFO_ARRAY_SIZE];    // Data to transfer
} FIFO_MULT, * PFIFO_MULT;
```

IOCTL_ccPB6Orn1_CHAN_NRZL_RDM_FIFO

Function: Read multiple data words from NRZL RX Data FIFO
Input: FIFO_MULT
Output: FIFO_MULT
Notes: Select Count to load and receive data in array. See UserAp for reference.

```
typedef struct _FIFO_MULT
{
   ULONG    Count;        // number of LWs to Read/Write
   ULONG    Data[NRZL_FIFO_ARRAY_SIZE];    // Data to transfer
} FIFO_MULT, * PFIFO_MULT;
```

IOCTL_ccPB6Orn1_CHAN_NRZL_LOAD_TXDFIFO

Function: Write a single LW to NRZL TX Data FIFO
Input: LW
Output: none
Notes: See UserAp for reference.

IOCTL_ccPB6Orn1_CHAN_NRZL_READ_RXDFIFO

Function: Read a single LW from NRZL RX Data FIFO
Input: none
Output: LW
Notes: See UserAp for reference.

IOCTL_ccPB6Orn1_CHAN_NRZL_SET_CNTL

Function: Write to NRZL Common Control
Input: NRZL_CHAN_CNTL
Output: none
Notes: Enable FifoBiPass to perform loop-back between TX and RX Data FIFOs. Use Port Reset to reset the state-machines and FIFOs. See UserAp and HW manual for reference.

IOCTL_ccPB6Orn1_CHAN_NRZL_GET_CNTL

Function: Read from NRZL Common Control
Input: none
Output: NRZL_CHAN_CNTL
Notes: See UserAp for reference.

```
typedef struct _NRZL_CHAN_CNTL {
   BOOLEAN  PortReset;
   BOOLEAN  FifoBipass;
} NRZL_CHAN_CNTL, * PNRZL_CHAN_CNTL;
```

IOCTL_ccPB6Orn1_CHAN_NRZL_SET_TXRATE

Function: Write to NRZL Transmitter Frequency Control
Input: ULONG
Output: none
Notes: load divisor to use with PLLC reference clock. N+1 is used to select 2X the desired Tx rate. Set to 10 MHz to get 5 MHz output. See UserAp and HW manual for reference.

IOCTL_ccPB6Orn1_CHAN_NRZL_GET_TXRATE

Function: Read from NRZL Transmitter Frequency Control
Input: none
Output: ULONG
Notes: See UserAp for reference.

IOCTL_ccPB6Orn1_CHAN_NRZL_SET_TXCNTL

Function: Write to NRZL Transmitter Control
Input: NRZL_CHAN_TXCNTL
Output: none
Notes: See UserAp and HW manual for reference.

IOCTL_ccPB6Orn1_CHAN_NRZL_GET_TXCNTL

Function: Read from NRZL Transmitter Control
Input: none
Output: NRZL_CHAN_TXCNTL
Notes: See UserAp for reference.

```
typedef struct _NRZL_CHAN_TXCNTL {
   BOOLEAN  TxEnable;      // Enable Transmitter SM to operate
   BOOLEAN  TxMsbLsb;      // True for Msb, False for Lsb first operation
   BOOLEAN  TxDataInv;     // True to invert Data
   BOOLEAN  TxClkInv;      // True to invert clock [active low]
   BOOLEAN  TxIntEnable;   // True to enable Transmitter interrupt
} NRZL_CHAN_TXCNTL, * PNRZL_CHAN_TXCNTL;
```

IOCTL_ccPB6Orn1_CHAN_NRZL_SET_RXCNTL

Function: Write to NRZL Receiver Control
Input: NRZL_CHAN_RXCNTL
Output: none
Notes: See UserAp and HW manual for reference.

IOCTL_ccPB6Orn1_CHAN_NRZL_GET_RXCNTL

Function: Read from NRZL Receiver Control
Input: none
Output: NRZL_CHAN_RXCNTL
Notes: See UserAp for reference.

```
typedef struct _NRZL_CHAN_RXCNTL {
   BOOLEAN  RxEnable;      // Enable Receiver SM to operate
   BOOLEAN  RxMsbLsb;      // True for Msb, False for Lsb first operation
   BOOLEAN  RxDataInv;     // True to invert Data
   BOOLEAN  RxClkInv;      // True to invert clock [active low]
   BOOLEAN  RxIntEnable;   // True to enable Receiver interrupt
} NRZL_CHAN_RXCNTL, * PNRZL_CHAN_RXCNTL;
```

IOCTL_ccPB6Orn1_CHAN_NRZL_LOAD_TXPFIFO

Function: Write to NRZL Transmitter Packet FIFO
Input: ULONG
Output: none
Notes: Write descriptor to Tx Packet FIFO to communicate to Tx State-Machine how many bits to send. Data should be loaded into Data FIFO first to prevent under run.

IOCTL_ccPB6Orn1_CHAN_NRZL_READ_RXPFIFO

Function: Read from NRZL Receive Packet FIFO
Input: none
Output: ULONG
Notes: Retrieve Descriptor to know how many bits are stored. See UserAp for reference.

IOCTL_ccPB6Orn1_CHAN_NRZL_LOAD_TXGAP

Function: Write to NRZL Transmitter GAP control
Input: ULONG
Output: none
Notes: Set count of 2X transmitter rate clocks to count between Packets being sent. Used for multi-packet transfer control.

IOCTL_ccPB6Orn1_CHAN_NRZL_READ_TXGAP

Function: Read from NRZL TX GAP Control
Input: none
Output: ULONG
Notes: Retrieve current Gap timing parameter.

IOCTL_ccPB6Orn1_CHAN_NRZL_LOAD_RXGAP

Function: Write to NRZL Receiver GAP control
Input: ULONG
Output: none
Notes: Set count of PLLC rate clocks to count before determining the last bit received was the last bit of the transfer [packet] Set to 2x the expected period.

IOCTL_ccPB6Orn1_CHAN_NRZL_READ_RXGAP

Function: Read from NRZL RX GAP Control
Input: none
Output: ULONG
Notes: Retrieve current Gap timing parameter.

IOCTL_ccPB6Orn1_CHAN_NRZL_SET_FIFO_LEVELS

Function: Sets the transmitter almost empty and receiver almost full FIFO levels.
Input: NRZL_CHAN_FIFO_LEVELS structure
Output: None
Notes: The FIFO levels are used to determine at what data count the TX almost empty and RX almost full status bits are asserted. The counts are compared to the word counts of the transmit FIFO or receive FIFO.

```
typedef struct _NRZL_CHAN_FIFO_LEVELS {
    ULONG    AlmostFull;    // program the Rx Almost Full Status Level
    ULONG    AlmostEmpty;   // program the Tx Almost Empty Status Level
} NRZL_CHAN_FIFO_LEVELS, * PNRZL_CHAN_FIFO_LEVELS;
```

IOCTL_ccPB6Orn1_CHAN_NRZL_GET_FIFO_LEVELS

Function: Returns the transmitter almost empty and receiver almost full levels.
Input: None
Output: NRZL_CHAN_FIFO_LEVELS structure
Notes: Returns the current values for the transmit almost empty and receive almost full FIFO levels.

IOCTL_ccPB6Orn1_CHAN_NRZL_GET_FIFO_COUNTS

Function: Returns the number of data words in the transmit and receive FIFOs.
Input: None
Output: NRZL_CHAN_FIFO_COUNTS structure
Notes: Both Data and Packet FIFOs are returned. Counts are zero extended.

```
typedef struct _NRZL_CHAN_FIFO_COUNTS {
    ULONG    TxDataCount;   // Number of words in the Transmit data FIFO
    ULONG    TxPktCount;    // Number of words in the Transmit Packet FIFO
    ULONG    RxDataCount;   // Number of words in the Receive data pipeline
    ULONG    RxPktCount;    // Number of words in the Receive Packet FIFO
} NRZL_CHAN_FIFO_COUNTS, * PNRZL_CHAN_FIFO_COUNTS;
```

IOCTL_ccPB6Orn1_CHAN_GET_NRZL_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: ccPB6Orn1_CHAN_ISR_STAT structure
Notes: Returns the interrupt status that was read in the interrupt service routine of the last interrupt caused by one of the enabled interrupt conditions.

```
typedef struct _ccPB6Orn1_CHAN_ISR_STAT {
  ULONG    Status;
  BOOLEAN  New;
} ccPB6Orn1_CHAN_ISR_STAT, *PccPB6Orn1_CHAN_ISR_STAT;
```

IOCTL_ccPB6Orn1_CHAN_GET_NRZL_STATUS

Function: Read from NRZL Status Register.
Input: none
Output: ULONG
Notes: See NRZL status definitions in public file or HW manual.

IOCTL_ccPB6Orn1_CHAN_GET_NRZL_STATUSII

Function: Read from NRZL ISR Status Register.
Input: none
Output: ULONG
Notes: See NRZL STAT2 definitions in public file or HW manual.

Warranty and Repair

Please refer to the warranty page on our website for the current warranty offered and options.
http://www.dyneng.com/warranty.html

Service Policy

Before returning a product for repair, verify as well as possible that the driver is at fault. The driver has gone through extensive testing, and in most cases it will be "cockpit error" rather than an error with the driver. When you are sure or at least willing to pay to have someone help then call or e-mail and arrange to work with an engineer. We will work with you to determine the cause of the issue.

Support

The software described in this manual is provided at no cost to clients who have purchased the corresponding hardware. Minimal support is included along with the documentation. For help with integration into your project please contact sales@dyneng.com for a support contract. Several options are available. With a contract in place Dynamic Engineers can help with system debugging, special software development, or whatever you need to get going.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois Street, Suite B/C
Santa Cruz, CA 95060
831-457-8891
support@dyneng.com