Digilent Port Communications Programmers Reference Manual

Revision: 06/03/05

www.digilentinc.com
215 E Main SuiteD | Pullman, WA 99163
(509) 334 6306 Voice and Fax

Introduction

The DPCUTIL Dynamic Link Library (DLL) provides an Applications Programming Interface (API), allowing Digilent system boards to communicate with applications software running under Microsoft Windows on a host computer.

When using DPCUTIL DLL a Digilent Communication Module is requred to create a communications channel between the host PC and a system board. Digilent currently provides communications modules supporting Ethernet, USB 2.0, and Serial RS-232 communication protocols.

The DPCUTIL API has functions for controlling and communicating with the scan chain of JTAG devices connected to a Digilent board. The DPCUTIL API can also send and recieve data to and from the user logic configured into the gate array on a connected Digilent board.

The JTAG scan chain manipulation functions are primarily for configuration of the programmable logic devices in the scan chain. The JTAG scan chain manipulation functions can also be used to access to the boundary scan registers in the device scan chain for loading of test vectors, the read back of test results, and other manipulations of the JTAG scan chain supported by the attached devices.

The DPCUTIL data transfer functions require that the gate array configuration contain a parallel port interface compatible with the Digilent Parallel Interface Module specification and reference design, available on the Digilent web site, at www.digilentinc.com. This interface allows the user to define a set of addressable registers that can be accessed by the DPCUTIL data transfer API functions. The data transfer API functions allow for writing or reading a single register, writing or reading sets of registers, or reading or writing a stream of data into or out of a single register.

The DPCUTIL DLL was created and compiled using the Microsoft C++ compiler in Visual Studio 6. The API is defined as a set of C callable functions, and can be used with programs written in either C or C++. It is also possible to write programs using Microsoft Visual BASIC to access the DPCUTIL API functions, but Digilent does not provide technical support for this use.

Overview

The Components

In order to use any of the DPCUTIL API functions, a program source module must include the following header files in this order:
#include <windows.h>
#include “dpcdefs.h”
#include “dpcutil.h”

These header files should be placed in a directory visible to the compiler. The files should include a path for the development environment that causes the compiler to search the appropriate directory at compilation time.

The program must be linked with the dpcutil.lib library. This establishes the dynamic link references between the application program and the DPCUTIL dynamic link library. This library file should be placed in a directory visible to the linker and the linker library search path set in the development environment to cause the linker to search the appropriate directory at program link time.

The DPCUTIL Dynamic Link Library (dpcutil.dll) must be in a directory that is searched by the operating system at program run time. The DLL file should either be placed in the same directory as the application program, or in a directory that is listed on the operating system PATH environment variable. When a program is executed, the operating system will look for dynamic link libraries in the directory where the program executable resides or in any directory listed on the system PATH environment variable.

DPCUTIL API functions

Most DPCUTIL API calls are formed into transactions. These transactions are put into a queue and processed on a first-in-first-out basis (FIFO). A function can either return upon completion (blocking) or return immediately and be processed on another thread (non-blocking). A transaction entered into the queue is assigned a TRID (Transaction ID), a value used to distinguish between transactions.

DPCUTIL API functions that require an established connection with a communications device must be passed a HIF (interface handle) to specify the connection to use. This handle is acquired by calling *DpcOpenJtag* or *DpcOpenData*.

Most DPCUTIL API functions require a pointer to an error code of type ERC. This variable will hold the error code for a completed transaction.

Most JTAG and Data Transfer functions have a pointer of type TRID as a parameter. If this parameter is set to NULL, the function will block and not return until the transaction has completed. Otherwise, if a non-null TRID pointer is sent, the function will return immediately and the transaction will be processed on a different thread. Since none of the data sent to DPCUTIL is copied, all data sent to a non-blocking API call must remain intact and unchanged until the transaction is complete. Calling the *DpcWaitForTransaction* function and sending it the TRID of a particular transaction will allow an application to wait for the completion of the transaction. Sending a TRID of NULL to *DpcWaitForTransaction* will force a wait on all transactions in the queue.

Initializing DPCUTIL

Before any of the DPCUTIL API functions can be used, the *DpcInit* function must be called. If it returns false, an error occurred while attaching and initializing the DLL. The application must not call any other DPCUTIL API functions if *DpcInit* returns false.

The following is a simple list of all DPCUTIL API functions (with return types) available for public use:

BOOL DpcInit	*BOOL DpcPutTmsTdiBits*
VOID DpcTerm	*BOOL DpcGetTdoBits*
BOOL DpcGetDpcVersion	*BOOL DpcOpenData*
BOOL DpcStartNotify	*BOOL DpcCloseData*
BOOL DpcEndNotify	*BOOL DpcPutReg*
BOOL DpcQueryConfigStatus	*BOOL DpcGetReg*
BOOL DpcPendingTransactions	*BOOL DpcPutRegSet*
BOOL DpcAbortConfigTransaction	*BOOL DpcGetRegSet*
BOOL DpcClearConfigStatus	*BOOL DpcPutRegRepeat*
BOOL DpcWaitForTransaction	*BOOL DpcGetRegRepeat*
ERC DpcGetFirstError	*VOID DvmgStartConfigureDevices*
BOOL DpcOpenJtag	*int DvmgGetDevCount*
BOOL DpcCloseJtag	*BOOL DvmgGetDevName*
BOOL DpcEnableJtag	*BOOL DvmgGetDevType*
BOOL DpcDisableJtag	*int DvmgGetDefaultDev*
BOOL DpcSetTmsTdiTck	*int DvmgGetHDVC*
BOOL DpcPutTdiBits	

The Device Table

All communication modules accessed through DPCUTIL are kept in a table called the *device table*. All details needed to connect to a device are stored in this table. Each device in the table is assigned a name and DPCUTIL uses this name (and not the index) to access the device. The device table can only be modified through a dialog box included in the DPCUTIL DLL. Calling the *DvmgStartConfigureDevices* function will open this dialog box.

To get the total number of devices in the device table, call the *DvmgGetDevCount* function. To get the name of a device with a given index, call the *DvmgGetDevName* function. A device table always has a default device in it. The index of this device can be obtained by calling *DvmgGetDefaultDev*. If there are no devices in the device table, *DvmgGetDefaultDev* will return –1.

Using the DPCUTIL Data Transfer functions

The data transfer functions in DPCUTIL rely on the logic loaded into the FPGA. This logic must reserve byte-sized registers used for reading and/or writing. Through these registers, the DPCUTIL data transfer functions will communicate with the FPGA through a connected communications module. As mentioned before, the data transfer functions allow for:

1. a register be written to or read from
2. many registers can be written to or read from as a single transaction.
3. a stream of bytes to be sequentially written to or read from a single register

In this way, an application can communicate with an FPGA through DPCUTIL. For a more detailed explanation of the interface between the FPGA and communications module, see *Digilent Parallel Interface Module Reference Manual*.

Before using any data transfer functions, the application must connect to a communication device using *DpcOpenData*. The first parameter is a pointer to an interface handle (hif). If the function returns successfully, this handle will be used to connect to the device in all proceeding data transfer calls. The device is specified by its assigned name in the device table and passed as the second parameter in the *DpcOpenData* function.

After this API function is called, any of the data transfer functions can be used. When finished, close the device using *DpcCloseData*.

About JTAG

Most logic memory devices are programmable. Many chip manufacturers accomplish this by conforming these devices to a standard specified in IEEE 1149.1. This is the Joint Test Action Group, or JTAG. A device is said to be JTAG compliant if it contains a JTAG TAP controller and the following pins: TDI, TDO, TMS, and TCK.

TDI inputs data into the JTAG TAP controller, and TDO provides outputs. TMS is used to set the JTAG TAP controller to a specified state, and TCK is used to clock bits into and out of the JTAG TAP controller. After being set to the proper state by TMS, bits are shifted into the TAP controller on TDI while bits are shifted out on TDO. Any FPGA, CPLD, or PROM that is JTAG compliant can be erased, programmed, and verified using this standard.

The TDI and TDO pins of several JTAG devices can be connected together to form a chain. This is called a JTAG scan chain. In order to program a device in the JTAG scan chain, all other devices are first set to BYPASS, meaning that they are ignored and not to be configured. Then a series of bits are shifted into the scan chain through TDI to configure the device.

For more information on JTAG functionality and programming, read the IEEE 1149.1

specification and Xilinx app notes on device programming.

Using the DPCUTIL JTAG functions

Before using any JTAG functions, *DpcOpenJtag* must be called to connect to a communication device. The first parameter is a HANDLE pointer. If the function returns successfully, this handle will be used to connect to the device in all proceeding JTAG calls. The device is specified by name and passed as the second parameter in the *DpcOpenJtag* function.

DpcEnableJtag must be called directly after *DpcOpenJtag*. This enables the driving of JTAG signals on the communication device.

After these two API functions are called, any of the JTAG functions can be used. When finished, disable the JTAG interface and close the device using *DpcDisableJtag and DpcCloseJtag*.

Multiple Instances of DPCUTIL (for Win32 applications)

More than one application can use the DPCUTIL.DLL at once. This presents the possibility of the device table being changed leaving other instances with outdated information about it. To remedy this, an application should register itself with the *DpcStartNotify* function. Whenever a modification is made to the device table, a registered application will be notified of the change via a message sent to the provided window handle. The application can then reload all needed information about the device table. The following is a description of the message parameters to the window procedure:

HWND	user specified window handle
MSG	WM_COMMAND
WPARAM	
low order word	user specified identifier.
high order word	0
LPARAM	0

To stop notification messages, an application should call the *DpcEndNotify* function.

Description of Data TypesTRID

16 bit data type that holds the ID of a transaction (used for non-blocking calls)

<u>ERC</u>
32 bit (signed) data type. Holds error code for a finished transaction

<u>TRT</u>
32 bit (signed) data type. Holds code for transaction type

<u>STS</u>
32 bit (signed) data type. Holds code for transaction status

<u>DVCT</u>
32 bit (signed) data type. Holds code for communications interface type.

<u>TRS</u>
Structure that contains the following information about a transaction:

```
Typedef struct tagTRS {
TRT    trt;     /* transaction type */
TRID   trid;    /* transaction ID */
STS    sts;     /* status of transaction */
ERC    erc;     /* error code for transaction */
}TRS;
```

Description of API calls

API Startup/Cleanup calls

<u>BOOL DpcInit(ERC * perc)</u>

Parameters
perc - pointer to store error code

Return Values
Returns true if DLL instance is properly initialized

Description
This function performs startup initialization of the DLL. It must be called before any of the other API calls can be used.

<u>void DpcTerm()</u>

Parameters
none

Return Values
none

Description
This function must be called to clean up resources when the application is done using the DLL.

API Transaction and Utility calls

<u>BOOL DpcStartNotify (HWND hwnd, WORD idNotify, ERC *perc)</u>

Parameters
hwnd - handle of window that is to be sent notification messages
idNotify - message identifier to be sent upon device table change
perc - pointer to store error code

Return Values
Returns true if successful. Returns false otherwise.

Description
Used to register a window handle for being notified of device table changes. When a change (deletion, addition, or modification) occurs in the device table, all registered windows are sent their specified messages.

<u>BOOL DpcEndNotify (HWND hwnd, ERC *perc)</u>

Parameters
hwnd - handle of window that is to no longer be sent notification messages
perc - pointer to store error code

Return Values
Returns true if successful. Returns false otherwise.

Description
De-registers a specified window handle from being notified of device table changes

<u>BOOL DpcPendingTransactions(HANDLE hif, int * pctran, ERC *perc)</u>

Parameters
hif - handle to JTAG interface
pctran - pointer to store number of pending transactions
perc - pointer to store error code

Return Values
Returns true if any non-blocking transactions are pending.

Description
Used to check if non-blocking transactions are still pending. If the function returns true, the number of non-blocking, pending transactions is returned by reference in pctran.

<u>BOOL DpcQueryConfigStatus(HANDLE hif, TRID trid, TRS * ptrs, ERC * perc)</u>

Parameters
hif - handle to JTAG interface
trid - transaction ID to query. If 0, then the status of the oldest transaction is queried.
ptrs - pointer to store information about transaction
perc - pointer to store error code

Return Values
Returns true if transaction ID is found. Returns false otherwise.

Description
Used to check a specified transaction's status. If the transaction is complete, it will be removed from the status queue and subsequent calls to DpcQueryConfigStatus with the same trid will fail. If the transaction is not complete, its information will be returned by reference in a TRS structure.

<u>BOOL DpcAbortConfigTransaction(HANDLE hif, TRID trid, ERC * perc)</u>

Parameters
hif - handle to JTAG interface
trid - transaction ID to abort. If 0, then abort ALL transactions
perc - pointer to store error code

Return Values
Returns true if successful. Returns false otherwise.

Description
Aborts a specified transaction or all transactions. After a transaction is aborted, it will remain in the status queue until read out with DpcQueryConfigStatus or DpcClearConfigStatus. Not all transactions will abort immediately, so DpcQuery ConfigStatus should be called afterward to make sure that the transaction has been terminated.

<u>BOOL DpcClearConfigStatus(HANDLE hif, TRID trid, ERC * perc)</u>

Parameters
hif - handle to JTAG interface
trid - transaction ID to clear from status queue. If 0, then clear all completed transactions.
perc - pointer to store error code

Return Values
Returns true if successful. Returns false otherwise.

Description
Clears completed transactions from the status queue. If the trid of a specific transaction is sent, then that transaction will be cleared if completed. If the trid is set as 0, then all completed transactions will be cleared.

<u>BOOL DpcWaitForTransaction(HANDLE hif, TRID trid, ERC * perc)</u>

Parameters
hif - handle to JTAG interface
trid - transaction ID to wait on. If 0, then wait for all transactions to complete.
perc - pointer to store error code

Return Values
Returns true if successful. Returns false otherwise.

Description
Wait indefinitely on a transaction if its trid is specified. Wait for all transactions to complete if trid is 0. This indefinite block can be broken if the transaction being waited on is aborted.

<u>ERC DpcGetFirstError(HANDLE hif)</u>

Parameters
hif - handle to JTAG interface

Return Values
Returns the first error code encountered in the status queue.

Description
Searches through the status queue for the first transaction with an error code set. Returns the error code *ercNoError* if no error codes are found.

BOOL DpcGetDpcVersion (char * szVersion, ERC *perc)

Parameters

szVersion - string that will store the current version of dpcutil.dll

perc - pointer to store error code

Return Values

Returns true if the version string is successfully retrieved. Returns false otherwise.

Description

Stores the version string of dpcutil.dll into the given szVersion pointer.

API JTAG manipulation calls

BOOL DpcOpenJtag(HANDLE * phif, char * szdvc, ERC * perc, TRID * ptrid)

Parameters

phif - pointer to store opened interface handle

szdvc - name of device to open

perc - pointer to store error code

ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Opens the JTAG interface for access. No other JTAG configuration transactions can be used until a communications device has been opened and enabled (enabling a JTAG interface is performed by calling the *DpcEnableJtag* API function. The communications device is specified by the name assigned to it in the device table and this name is placed in szdvc. The handle to the JTAG interface is returned by reference in phif.

BOOL DpcCloseJtag(HANDLE hif, ERC * perc)

Parameters

hif - handle to JTAG interface

perc - pointer to store error code

Return Values

Returns true if successful. Returns false otherwise.

Description

Releases the JTAG interface specified by hif and closes the communications module.

BOOL DpcEnableJtag(HANDLE hif, ERC * perc, TRID * ptrid)

Parameters

hif - handle to JTAG interface

perc - pointer to store error code

ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Enables the driving of JTAG signals. Must be called after FOpenJtag is called. After the JTAG signals have been enabled on the communications module, the JTAG manipulation functions in DPCUTIL can be used.

BOOL DpcDisableJtag(HANDLE hif, ERC * perc, TRID * ptrid)

Parameters

hif - handle to JTAG interface

perc - pointer to store error code
ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values
Returns true if successful. Returns false otherwise.

Description
Disables driving of JTAG signals. Must be called before FCloseJtag is called.

<u>BOOL DpcSetTmsTdiTck(HANDLE hif, BOOL fTms, BOOL fTdi, BOOL fTck, ERC * perc, TRID * ptrid)</u>

Parameters
hif - handle to JTAG interface
fTms - value of TMS pin (true = 1, false = 0)
fTdi - value of TDI pin (true = 1, false = 0)
fTck - value of TCK pin (true = 1, false = 0)
perc - pointer to store error code
ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values
Returns true if successful. Returns false otherwise.

Description
Sets the JTAG lines TMS, TDI, and TCK to a specified state.

<u>BOOL DpcPutTdiBits(HANDLE hif, int cbit, BYTE * rgbSnd, BOOL bitTms, BOOL fReturnTdo, BYTE * rgbRcv, ERC * perc, TRID * ptrid)</u>

Parameters
hif - handle to JTAG interface
cbit - number of TDI bits to clock into the JTAG TAP controller
rgbSnd - buffer that holds TDI values. Bits are shifted in sequentially starting at the first element in the BYTE array, from LSB to MSB.
bitTms - value the TMS pin will be held at while the rgbSnd bits are shifted in (true = 1, false = 0)
fReturnTdo - specifies if bits from TDO should be returned
rgbRcv - (optional) holds TDO bits. Only used if fReturnTdo is true.
perc - pointer to store error code

ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values
Returns true if successful. Returns false otherwise.

Description
Shifts a specified number of bits into TDI and (optionally) returns bits shifted out TDO. RgbSend holds TDI bits. Each bit is shifted in sequentially, starting at the first element in the array, from least significant bit to most significant bit Below is an example of how the TDI bits are placed in each byte of rgbSend.

rgbSend[0]

TDI 8	TDI 7	TDI 6	TDI 5	TDI 4	TDI 3	TDI 2	TDI 1

rgbSend[1]

TDI 16	TDI 15	TDI 14	TDI 13	TDI 12	TDI 11	TDI 10	TDI 9

If fReturnTdo is set to true, all bits shifted out are stored in rgbRcv Tdo. Each bit is shifted out sequentially starting at the first element in the rgbRcv, from lowest significant bit to most significant bit. TMS is held at the value specified by bitTms while bits are being shifted into TDI.

<u>BOOL DpcPutTmsTdiBits(HANDLE hif, int cbit, BYTE * rgbSnd, BOOL fReturnTdo, BYTE * rgbRcv, ERC * perc, TRID * ptrid)</u>

Parameters

hif - handle to JTAG interface

cbit - number of TMS and TDI bit pairs sent.

rgbSend[0]

TMS 4	TDI 4	TMS 3	TDI 3	TMS 2	TDI 2	TMS 1	TDI 1

rgbSend[1]

MS 8	TDI 8	TMS 7	TDI 7	TMS 6	TDI 6	TMS 5	TDI 5

rgbSnd - buffer that holds TMS and TDI bit pairs.

fReturnTdo - specifies if bits from TDO should be returned

rgbRcv - (optional) holds shifted out TDO bits. Only used if fReturnTdo is true.

perc - pointer to store error code

ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Shifts a specified number of bits into TMS and TDI and (optionally) returns bits shifted out of TDO. RgbSend holds TMS and TDI bit pairs. Each bit pair is shifted in sequentially; starting at the first element in the array, from least significant bit pair to most significant bit pair. In each pair, the TMS value is the MSB and the TDI value is the LSB. Below is an example of how the TMS/TDI bit pairs are placed in each byte of rgbSend.

If fReturnTdo is set to true, all bits shifted out are stored in rgbRcv Tdo. Each bit is shifted out sequentially starting at the first element in the rgbRcv, from lowest significant bit to most significant bit.

<u>BOOL DpcGetTdoBits(HANDLE hif, int cbits, BOOL bitTdi, BOOL bitTms, BYTE *rgbRcv, ERC *perc, TRID *ptrid)</u>

Parameters

hif - handle to JTAG interface

cbit - number of TMS and TDI bits to push onto the JTAG TAP controller

bitTdi - value TDI pin will be held at (true = 1, false = 0)

bitTms - value TMS pin will be held at (true = 1, false = 0)

rgbRcv - holds shifted out TDO bits.

perc - pointer to store error code

ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Shifts out and returns a specified number of bits from TDO. Each bit is shifted out sequentially starting at the first element in the rgbRcv, from lowest significant bit to most significant bit. Below is an example of how the TDO bits are placed in each byte of rgbSend.

rgbRcv [0]	TDO 8	TDO 7	TDO 6	TDO 5	TDO 4	TDO 3	TDO 2	TDO 1
rgbRcv [1]	TDO 16	TDO 15	TDO 14	TDO 13	TDO 12	TDO 11	TDO 10	TDO 9

TDI and TMS are held at the values specified in bitTdi and bitTms while bits are shifted out of TDO.

API Data Transfer calls

<u>BOOL DpcOpenData(HANDLE * phif, char * szdvc, ERC * perc, TRID * ptrid)</u>

Parameters

phif - pointer to store handle to Data interface
szdvc - name of device to open
perc - pointer to store error code
ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Opens the data transfer interface for access. No other data transfer functions can be used until a communications device has been opened. The communications device is specified by the name assigned to it in the device table and this name is placed in szdvc. The handle to the data transfer interface is returned by reference in phif.

<u>BOOL DpcCloseData(HANDLE hif, ERC * perc)</u>

Parameters

hif - handle to Data interface
perc - pointer to store error code

Return Values

Returns true if successful. Returns false otherwise.

Description

Releases the data transfer interface specified by hif and closes the communications module.

<u>BOOL DpcPutReg(HANDLE hif, BYTE bAddr, BYTE bData, ERC * perc, TRID * ptrid)</u>

Parameters

hif -handle to Data interface
bAddr -address of register to send data byte
bData -data byte to send to address
perc- pointer to store error code
ptrid- pointer to store transaction ID.
If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Sends a single data byte to a register specified by its address.

BOOL DpcGetReg(HANDLE hif, BYTE bAddr, BYTE * pbData, ERC * perc, TRID * ptrid)

Parameters

hif - handle to Data interface
bAddr - address of register to read Data byte
pbData - pointer to store data byte read
perc - pointer to store error code
ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Gets a single data byte from a register specified by its address.

BOOL DpcPutRegSet(HANDLE hif, BYTE * rgbAddr, BYTE * rgbData, int cbData, ERC * perc,
TRID * ptrid)

Parameters

Hif - handle to Data interface
rgbAddr - addresses of registers to write Data bytes
rgbData - buffer with data bytes to be sent
cbData - number of bytes to be sent
perc - pointer to store error code
ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Sends many data bytes to many specified addresses. RgbAddr is a buffer containing addresses of registers. RgbData is a buffer containing data that will be sent to the corresponding addresses. Each element in the rgbData buffer is written to the corresponding address in the rgbAddr buffer. For example, the data byte in rgbData[0] is written to the register address in rgbAddr[0], the data byte in rgbData[1] is written to the register address in rgbAddr[1], etc.

BOOL DpcGetRegSet(HANDLE hif, BYTE * rgbAddr, BYTE * rgbData, int cbData, ERC * perc,
TRID * ptrid)

Parameters

hif - handle to Data interface
rgbAddr - addresses of registers to read Data bytes
rgbData - buffer to store data bytes read
cbData - number of bytes to be sent
perc - pointer to store error code
ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Gets many data bytes from many specified addresses. RgbAddr is a buffer containing addresses of registers. RgbData is a buffer that will contain data recieved from the registers specified by the addresses in rgbAddr. Each element in the rgbData buffer is read from the corresponding address in the rgbAddr buffer. For example, the data byte in the register specified by the address in rgbAddr[0] is written to rgbData[0], the data byte in the register specified by the address in rgbAddr[1] is written to rgbData[1], etc.

BOOL DpcPutRegRepeat(HANDLE hif, BYTE bAddr, BYTE * rgbData, int cbData, ERC * perc,
TRID * ptrid)

Parameters

hif - handle to Data interface
bAddr - address of register to send stream of data bytes
rgbData - buffer with data bytes to be sent to address
cbData - number of bytes to be sent
perc - pointer to store error code
ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Sends a stream of data bytes to a single, specified register address. BAddr is a register address that will be sent many bytes one at a time in sequential order. RgbData is a buffer containing the data that will be sent to bAddr. The will be sent as quickly as the Digilent Communications Interface Module will allow. The number of bytes to be sent to bAddr is specified in cbData.

BOOL DpcGetRegRepeat (HANDLE hif, BYTE bAddr, BYTE * rgbData, int cbData, ERC * perc,
TRID * ptrid)

Parameters

hif - handle to Data interface
bAddr - address of register to send stream of data bytes
rgbData - buffer to store data bytes from address
cbData - number of bytes to be received from address
perc - pointer to store error code
ptrid - pointer to store transaction ID. If ptrid is NULL, this function will be blocking.

Return Values

Returns true if successful. Returns false otherwise.

Description

Gets a stream of data bytes from a single, specified register address. RgbData is a buffer that will contain data read out of the register at address bAddr. The data is read as quickly as the Digilent Communications Interface Module will allow. The number of bytes to be read out of bAddr is specified in cbData.

API Device Manager calls

void DvmgStartConfigureDevices(HWND hwnd, ERC *perc)

Parameters

hwnd - handle to parent window
perc - pointer to store error code

Return Values

none

Description

Opens the Device Table dialog box. This dialog box is used to add, remove, or modify communication modules in the device table.

int DvmgGetDevCount(ERC *perc)

Parameters

perc - pointer to store error code

Return Values

Returns number of devices in the device table

Description
Gets the total number of devices in the device table.

<u>BOOL DvmgGetDevName(int idvc, char * szdvcTemp, ERC*perc)</u>

Parameters
idvc - index of device to query
szdvcTemp - string to store device name
perc - pointer to store error code

Return Values
Returns true if successful. Returns false otherwise.

Description
Gets the name of a device from its index into the device table

<u>BOOL DvmgGetDevType(int idvc, DVCT * dvtp, ERC* perc)</u>

Parameters
idvc - index of device to query
dvtp - pointer to store device type
perc - pointer to store error code

Return Values
Returns true if successful. Returns false otherwise.

Description
Gets the type of a particular device, given its index in the device table. The device type is returned by reference in dvtp.

<u>int DvmgGetDefaultDev(ERC *perc)</u>

Parameters
perc - pointer to store error code

Return Values
Returns index of default device in table

Description
Gets the index of the default device in the device table

<u>int DvmgGetHDVC (char* szdvc, ERC *perc)</u>

Parameters
Szdvc - name of device
perc - pointer to store error code

Return Values
Returns index of default device in table

Description
Gets the index of a device given its name. If no device is found, the returned index is –1, and an error code is set in perc.

Type/Error Codes

ERC

The following are error codes of type ERC and used to specify transaction errors in DPCUTIL.

Error Code	**Value**	**Description**
ercNoError	0	No error occurred in transaction
ercInvParam	3004	Invalid parameter sent in API call
ercInvCmd	3005	Internal error. Please report occurrence as a bug
ercUnknown	3006	Internal error. Please report occurrence as a bug
ercNoMem	3009	Not enough memory to carry out transaction
ercNotInit	3102	Communication device not initialized
ercCantConnect	3103	Can't connect to communication module
ercAlreadyConnect	3104	Already connected to communication device
ercSendError	3105	Error occurred while sending data to communication device
ercRcvError	3106	Error occurred while receiving data from communication device
ercAbort	3107	Error occurred while trying to abort transaction(s)
ercOutOfOrder	3109	Completion out of order
ercExtraData	3110	Too much data received from communication device
ercMissingData	3111	Nothing to send or data/address mismatched pairs
ercTridNotFound	3201	Unable to find matching TRID in transaction queue
ercNotComplete	3202	Transaction being cleared is not complete
ercNotConnected	3203	Not connected to communication device
ercWrongMode	3204	Connected in wrong mode (JTAG or data transfer)
ercWrongVersion	3205	Internal error. Please report occurrence as a bug
ercDvctableDne	3301	Device table doesn't exist (an empty one has been created)
ercDvctableCorrupt	3302	All or part of the device table is corrupted
ercDvcDne	3303	Device does not exist in device table
ercDpcutilInitFail	3304	DpcInit API call failed
ercDvcTableOpen	3306	Communications devices dialog box already open.
ercRegError	3307	Error occurred while accessing the registry

STS

The following are status codes of type STS and used to specify the status of a transaction

stsNew	1	Transaction has not been processed
stsComplete	2	Transaction is complete

DVCT

The following are ID codes of type DVCT and used to specify types of communications devices

dvctEthernet	0	Ethernet device type
dvctUSB	1	USB device type
dvctSerial	2	Serial device type