TEXAS INSTRUMENTS

Reference Manual
TI DN 2510446
July 2009

DLP Discovery™ 4100 Controller Board API Programmer's Guide

This manual describes the use of the Application Programming Interface (API) for the DLP Discovery Controller Board. The Controller Board combines hardware, software, firmware, and documentation to form a stand-alone platform for use in developing and testing applications designed for using the Texas Instruments DLP Discovery.

IMPORTANT NOTICE

BEFORE USING TECHNICAL INFORMATION, THE USER SHOULD CAREFULLY READ THE FOLLOWING TERMS.

The term "Technical Information" includes reference designs, drawings, specifications, and other information relating to TI DLP® products or applications, contained herein or provided separately in any format or via any medium.

TI is providing Technical Information for the convenience of purchasers of DLP® products ("Users"), and will not accept any responsibility or liability arising from providing the Technical Information or its use. Any use or reliance on Technical Information is strictly the responsibility of the User.

1. **No Warranty.** ***THE TECHNICAL INFORMATION IS PROVIDED "AS IS".*** TI MAKES NO WARRANTIES OR REPRESENTATIONS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING LACK OF VIRUSES, ACCURACY, OR COMPLETENESS. TI DISCLAIMS ANY WARRANTY OF TITLE, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT, QUIET POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE TECHNICAL INFORMATION OR THE USE OF THOSE MATERIALS.
2. **Warranty for Products Not Affected.** The foregoing exclusion and disclaimer of warranty does not affect or diminish any warranty rights with regard to DLP® products. Such rights are governed exclusively by the terms of a written and signed purchase agreement with TI.
3. **Limitations and Exclusion of Damages.** IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT OF THE TECHNICAL INFORMATION OR THE USE OF THE TECHNICAL INFORMATION.
4. **No Engineering Services.** User is fully responsible for all design decisions and engineering with regard to its products, including decisions relating to application of DLP® products. By providing Technical Information TI does not intend to offer or provide engineering services or advice concerning User's design. If User desires engineering services, then User should rely on its retained employees and consultants and/or procure engineering services from a licensed professional engineer ("LPE").
5. **Compliance with Export Control Laws.** Unless prior authorization is obtained from the U.S. Department of Commerce, User may not export, re-export, or release, directly or indirectly, any Technical Information, or export, directly or indirectly, any direct product of such Technical Information to any destination or country to which the export, re-export or release of the Technical Information or direct product is prohibited by the Export Administration Regulations of the U.S. Department of Commerce ("EAR").

Notational Conventions

This document uses the following conventions.

The Graphical User Interface is also referred to as GUI.

Program listings, program examples, and interactive displays are shown as a `special typeface` similar to a typewriter's. Some examples use a **`bold version`** of the special typeface for emphasis; interactive displays use a **`bold version`** of the special typeface to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.).

Here is a sample program listing:

```
0011 0005 0001 .field 1, 2
0012 0005 0003 .field 3, 4
0013 0005 0006 .field 6, 3
0014 0006 .even
```

In syntax descriptions, the instruction, command, or directive is in a **bold typeface** font and parameters are in an *italic* typeface. Portions of the syntax that are **bold** should be entered as shown; portions of syntax that are in italics describe the type of information that should be entered. Syntax that is entered on a command line is centered. Syntax that is used in a text file is left justified.

Square brackets ([and]) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets. Unless the square brackets are in a bold typeface, do not enter the brackets themselves.

Table of Contents

List of Illustrations

1.0 Overview

This manual provides:

A general description of the Discovery 4100 Board API

An overview of the ActiveX™ interface commands

Programming Examples

2.0 Terms and Definitions

API	Application Programming Interface
DDC4100	Discovery 4100 Controller FPGA
DMD	Digital Micromirror Device
GUI	Graphical User Interface
USB	Universal Serial Bus
SHORT	16 bit value
LONG	32 bit value

3.0 API Overview

Direct Dimensions, Inc.

The Discovery 4100 Controller Board API provides a complete function library to support custom programming of Discovery 4100 control applications. Most popular Windows development platforms are supported. The API provides control of the DDC4100 Controller Board hardware via the USB by interfacing with an ActiveX™ control. The ActiveX™ control is designed to transmit data, send commands, and read the status of the controller board. The API provides the capability to control all functions of the DDC4100 controller and to control image load and display to the Discovery 4100 board.

3.1 DMD Image Control

Images are controlled and displayed in blocks on the Digital Micromirror Device (DMD). The DMD is logically divided into blocks : 16 blocks of 48 rows of mirrors with 1024 mirrors in each row for XGA DMDs and 15 blocks of 72 rows of mirrors with 1920 mirrors in each row for 1080p DMDs. Blocks can be loaded and displayed individually, or as an entire image (all blocks). Refer the Discovery DDC4100 datasheet, TI DN 2509511, for more information.

4.0 Discovery 4100 Controller Board USB API Interface

ActiveX™ is a Microsoft defined set of technologies that enables software components to interact with one another regardless of the language in which they were created. ActiveX™ is built on the Component Object Model (COM). ActiveX™ controls are distributed as .ocx files and are predated by .vbx and dll files. Refer to http://msdn.microsoft.com/ and search for ActiveX™ for additional information on ActiveX™ controls.

The Discovery 4100 ActiveX™ control provides a convenient mechanism for communication between customer developed software applications and the Discovery 4100 driver software. The control is distributed in file "DDC4100l.ocx" and provides an interface for configuration, control and display to the Discovery 4100. When this control is used to build applications in C, C++ and Visual Basic control of the Discovery board is easily accomplished using the methods documented in this chapter. Refer to the sample code available on the TI KnowledgeBase for an example of using the ActiveX™ control. The following sections briefly describe the ActiveX™ control features.

4.1 Configuration/Status Methods

4.1.1 Void AboutBox( )

This subroutine displays an about box showing the ActiveX™ control version and copyright information.

4.1.2 Short AllowMessages(short value)

Controls display of error messages by ActiveX™ control. TRUE = nonzero= enable messages, FALSE = 0 = disable messages. Default is TRUE.

4.1.1 Short DownloadAppsFPGACode(LPCTSTR *FileName*)

This method will load the program (.bin format file) specified by *FileName* into the AppsFPGA of the Discovery 4100. Returned values are 1 if successful or 0 if unsuccessful.

4.1.2 Long GetActivexRev( )

Returns control revision. Upper 16 bits contain major revision, lower 16 bits contain minor revision.

4.1.3 Short GetNumDevices( )

This function will return the number of Discovery USB devices connected to the system. They will automatically be numbered starting at zero. For example, if you have four devices plugged in, GetNumDev( )will return a 4. To access these devices you will pass the ConnectDevice function a 1, 2, 3, or a 4 for that respective device.

4.1.4 Short ConnectDevice(short *DeviceNumber*, LPCTSTR *FileName)*

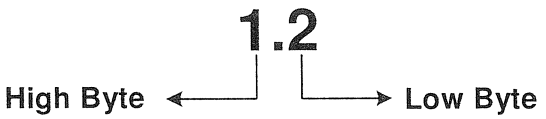
This must be called to initiate the board. Device number is used if multiple D4100 boards are connected to the PC, in most cases only one device will be connected and a 1 will be passed. If the board has not been previously initialized the application APPSFPGA code (.bin format file) passed as FileName will be loaded into the hardware.

4.1.5 Long GetDriverRev( )

Returns driver revision. Upper 16 bits contain major revision, lower 16 bits contain minor revision.

4.1.6 Short GetFirmwareRev( )

This command returns the firmware revision. The low byte of the returned value contains all the digits after the decimal point and the high byte contains all the digits before the decimal point.
Example:



4.1.7 Long GetDLLRev( )

Returns the version information for this DLL. The low byte of the returned value contains all the digits after the decimal point and the high byte contains all the digits before the decimal point. (See the example in 4.1.6)

4.1.8 unsigned int GetFPGARev( )

Returns the version of the firmware running on the D4100 applications FPGA. The low byte of the returned value contains all the digits after the decimal point and the high byte contains all the digits before the decimal point. (See the example in 4.1.6)

4.1.9 Short GetSpeedMode( )

The GetSpeedMode method is used to determine the speed at which the Discovery 4100 is operating. A return value of 1 indicates that the device is operating at high speed (USB 2.0), a value of 0 means it is operating at full speed (USB 1.0).

4.1.10 BOOL IsDeviceAttached( )

Returns a TRUE (1) if a Discovery 4100 Controller board has been attached using the GetDevice method. Otherwise a FALSE (0) value is returned.

4.2 Low Level Control Methods 4.2.1 – 4.2.29

Provide basic DDC4100 control. Any DMD function can be performed with the low level control methods. Low level control methods do not utilize the ActiveX control image buffer, they directly send data and commands over USB.

4.2.1 short LoadControl( )

Will load control registers into DMD. this can be used for all non data transaction such as, resets and clears. Returns a 1 is successful.

4.2.2 int LoadData(UCHAR* RowData, long length)

This function is used to load row data into the DMD. Data must be in a UCHAR array of size length. No more than 500 rows can be loaded at a time(96Kbit for 1080p, 51.2Kbit for XGA). To load an entire DMD call this function multiple times. Returns a 1 is successful.

4.2.3 Short SetBlkMd (short *value*)

Sets the BLKMD register of the D4100. Returned values are 1 if successful or 0 if unsuccessful.

4.2.4 Short GetBlkMd ( )

Gets the BLKMD values from the D4100 register.

4.2.5 Short SetBlkAd(short *value*)

Sets the BLKAD register of the D4100. Returned values are 1 if successful or 0 if unsuccessful.

4.2.6 Short GetBlkAd ( )

Gets the BLKMD values from the D4100 register.

4.2.7 Short SetRST2BLKZ(short *value*)

Sets the Reset Two Blocks flag on the D4100. Active = 0, inactive = 1 (default). Returned values are 1 if successful or 0 if unsuccessful.

4.2.8 Short GetRST2BLKZ ( )

Gets the status of the Reset Two Block flag from the D4100

4.2.9 Short SetRowMd(short *value*)

Sets the ROWMD register of the D4100. Returned values are 1 if successful or 0 if unsuccessful.

4.2.10 Short GetRowMd( )

Gets the ROWMD values from the D4100 register.

4.2.11 Short SetRowAddr(short *value*)

Sets the ROWAD register of the D4100. Returned values are 1 if successful or 0 if unsuccessful.

4.2.12 Short GetRowAddr( )

Sets the ROWAD register of the D4100.

4.2.13 short SetSTEPVCC(short *value*)

Sets the Step VCC flag on the D4100. Active = 1, inactive = 0 (default). Returned values are 1 if successful or 0 if unsuccessful.

4.2.14 short GetSTEPVCC( )

Gets the status of the Step VCC flag from the D4100

4.2.15 short SetCOMPDATA(short *value*)

Sets the Complement Data flag on the D4100. Active = 1, inactive = 0 (default). Returned values are 1 if successful or 0 if unsuccessful.

4.2.16 short GetCOMPDATA( )

Gets the status of the Complement Data flag from the D4100.

4.2.17 short SetNSFLIP(short *value*)

Sets the North South Flip flag on the D4100. Active = 1, inactive = 0 (default). Returned values are 1 if successful or 0 if unsuccessful.

4.2.18 short GetNSFLIP( )

Gets the status of the North South Flip flag from the D4100.

4.2.19 short SetWDT(short *value*)

Sets the Step Watch Dog Timer flag on the D4100. Active = 1 (default)., inactive = 0 . Returned values are 1 if successful or 0 if unsuccessful.

4.2.20 short GetWDT( )

Gets the status of the Watch Dog Timer flag from the D4100.

4.2.21 short SetPWRFLOAT(short *value*)

Sets the Power Float flag on the D4100. Active = 1, inactive = 0 (default). Returned values are 1 if successful or 0 if unsuccessful.

4.2.22 short GetPWRFLOAT( )

Gets the status of the Power Float flag from the D4100.

4.2.23 short SetEXTRESETENBL(short *value*)

This will enable or disable GPIOA.0 as an external reset input. Enabled = 1, disabled = 0 (default).

GPIOA.0 is a 2.5V CMOS input. Refer to TI DN 2509510 for signal location. When enabled, all software control of the DMD resets is disabled. Reset operation as defined by RST2BLKZ, BLK_MD, and BLK_ADDR will be initiated on rising edge of external reset input. Returned values are 1 if successful or 0 if unsuccessful.

4.2.24 short GetEXTRESETENBL( )

Get the current External Reset Enable value. Enabled = 1, disabled = 0 (default).

4.2.25 short SetGPIO(short *value*)

Set the output values of GPIOA.2-4 to provide programmable digital outputs. Bits 4,3,2 of value control the output state. Bits 7,6,5,1,0 of value are not used. GPIOA.2-4 are 2.5V CMOS outputs. Refer to TI DN 2509510 for signal location.

4.2.26 short GetGPIO( )

Returns the values of GPIOA.5-7 as digital inputs. Bits 7,6,5 of returned value are the input data. Bits 4,3,2,1,0 of returned value are not used. 2.5V CMOS inputs. Refer to TI DN 2509510 for signal location.

4.2.27 short GetDMDTYPE( )

Returns DMD type as listed below:

DMD TYPE :	
.95 1080p Type A	0
.7 XGA Type A	1
.55 XGA Type A	2
.55 XGA Type X	3

4.2.28 short GetRESETCOMPLETE(long *waittime*)

Will enable global external reset (see 4.2.23) then loop *waittime* in millisecond for an external reset trigger event to happen. If external reset occurs in the *waittime* windows the this function will return a 1. Set *waittime* to 0 to wait indefinably.

4.2.29 short SetConversionThreshold(short threshold)

Will set the image conversion threshold to a value between 0 and 255.

4.2.30 short GetDDCVERSION( )

Returns DDC Version.2-0 in return value bits 2,1,0.

4.3 DMD Display Operation Methods

DMD images may be sourced from an image file or constructed directly by software. Refer to Figure 1 for an illustration of image display methods.

Image files should be 1024x768 pixels for XGA DMDs, or 1920 x 1080 for 1080p DMDs. An image file must be reformatted to a format which can be sent to the DMD. This re-formatting first thresholds each pixel using a value of RGB(240,240,240) to convert from a grayscale image to a binary image with 1 bit per pixel. The pixel values are stored in a binary format for DMD loading. Images may be pre-converted and stored to disk file .bin format using ConvertImage or stored to memory images buffer using LoadImageFileToBuffer. The pre-converted images may be quickly displayed on the DMD.

The converted image data is transferred to an Active Control Internal Memory buffer. The data is transferred from this buffer to the DMD via the USB driver.

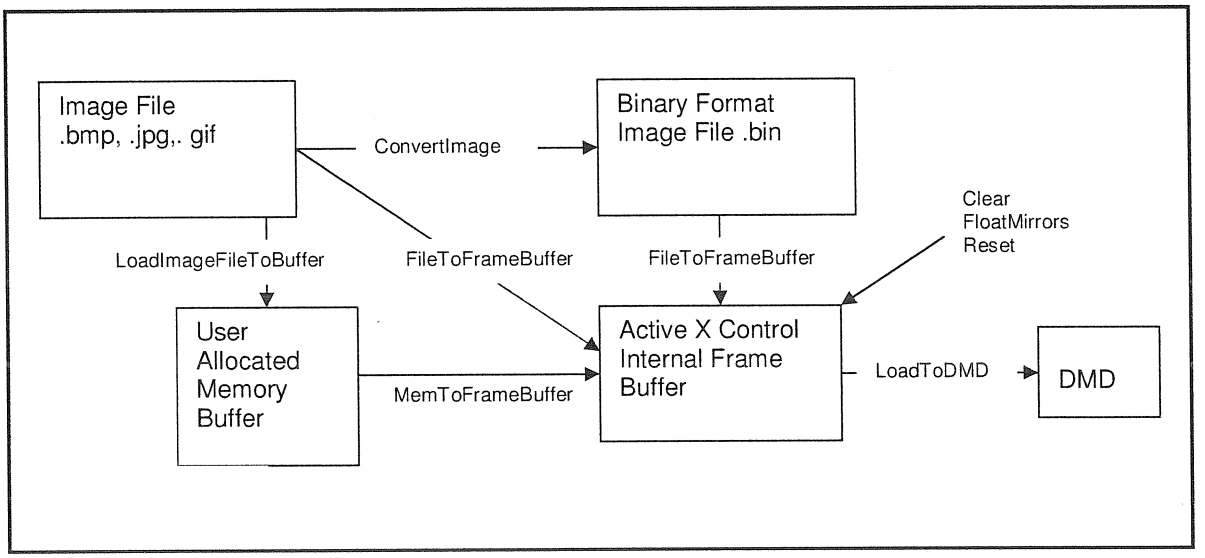


Figure 1. Graphical User Interface Layout

4.3.1 Short Clear(short *BlockNum, short DoReset*)

This method is used to clear the contents of an entire block on the device. Specify the block to be cleared in BlockNum. Block numbers range from 1 to 16, a block number greater than 16 will signal a Global Clear. Reset is executed on the specified blocks if DoReset is nonzero. Returned values are 1 if successful or 0 if unsuccessful.

4.3.2 Short ConvertImage (LPCTSTR *SrcFile*, LPCTSTR *DestFile*, short *MirrorImage*)

This method is used to convert a standard bmp, jpg, or gif image file, indicated by the file *SrcFile*, into a binary image file (.bin) which may be sent directly to the display. The binary image is stored into the file

indicated by *DestFile*. A mirror image may be created by setting *MirrorImage* to a non-zero value. Returned values are 1 if successful or 0 if unsuccessful.

A .bin file contains 1 bit per DMD pixel arranged starting with pixel (0,0). Arrange in row, column format. For example with and XGA DMD the first 1024 bit in the file contain the data for the first row, second 1024 bit is row 2.

4.3.3 Short FloatMirrors( )

Places the DMD in a safe state with the mirrors in the "floated" or flat condition, with no bias applied to the DMD. Returned values are 1 for success or 0 if unsuccessful.

4.3.4 Short FileToFrameBuffer(LPCTSTR *ImageFile*, short *MirrorImage*)

This method is used to process an image into a binary format which may be sent directly to the display and load the binary image into the ActiveX™ control image buffer. Setting *MirrorImage* to a non-zero value will instruct the method to construct a mirrored image. The image file must be in the form of a Bitmap file (bmp), a JPEG file (*.jpg), or a GIF file (*.gif), or a binary file (*.bin). The converted image is stored in the ActiveX™ control's image buffer. Returned values are 1 if successful or 0 if unsuccessful.

4.3.5 Short LoadToDMD (short *BlockNum, short DoReset*)

Use this method to load a block of data from the ActiveX™ control image buffer to the DMD. Specify the block to be loaded in *BlockNum*. Block numbers range from 1 to 16, a block number greater than 16 will load the entire DMD. Image data is loaded from the ActiveX™ control's image buffer. Reset for selected blocks is executed if DoReset is nonzero. Returned values are 1 if successful or 0 if unsuccessful.

4.3.6 Short MemToFrameBuffer (unsigned short* Image*BufferPtr*)

LoadFrameBuffer is used to load the ActiveX™ control image buffer with a binary image. The binary image may be read directly from a file generated by the ConvertImage method or may be program generated. Returned values are 1 if successful or 0 if unsuccessful.

4.3.7 Short LoadImageFileToBuffer(LPCTSTR *FileName*, unsigned short* *ImageBufferPtr*, short *MirrorImage*)

LoadImageFileToBuffer is used to load memory buffer *ImageBuffer* with the desired image. The image is converted to a binary format. The image is read a from a standard bmp, jpg, or gif image file, indicated by *FileName* . Returned values are 1 if successful or 0 if unsuccessful.

4.3.8 Short Reset(short *BlockNum*)

This method executes a reset operation for the block specified by BlockNum. Block numbers range from 1 to 16, a block number greater than 16 will signal a Global Reset. Returned values are 1 if successful or 0 if unsuccessful.

4.4 ActiveX™ Control Usage Examples

Control sequences for common Discovery operations are presented below in programming language independent examples :

4.4.1 Open USB Device

The USB device must be attached prior to performing any DMD operations. After connecting, the device will remain open until the calling program ends.

DDC4100Ctrl.**ConnectDevice(1,C:\usb_main.bin)** // Open channel to device

4.4.2 Display Single Image on DMD

A single image can be loaded from a standard image file and displayed on the DMD using the following methods :

DDC4100Ctrl.**FileToFrameBuffer** (*ImageFile*, *MirrorImage*) // Load ActiveX™ control's frame buffer from specified standard bmp, jpg, gif or .bin image file.
DDC4100Ctrl.**LoadResetFrame** ( ) // Write Image to DMD and reset

4.4.3 Display Multiple Images on DMD

To maximize display speed images should be converted and loaded to memory buffer(s) prior to starting display -

Multiple images are first converted and loaded to memory buffers by repeating this method as needed:
DDC DDC4100Ctrl Ctrl.**LoadImageFileToBuffer**(*FileName*, *ImageBuffer*, *MirrorImage*) // Convert image from specified standard bmp, jpg, gif image file to a format which can be directly written to the DMD. Store converted file in memory buffer *ImageBuffer.*

An image is displayed from memory by passing a pointer to the memory buffer :
DDC4100Ctrl. **MemToFrameBuffer** (*VarPBuffer*) // load ActiveX™ control's frame buffer from memory pointer *VarPBuffer*
DDC4100Ctrl.**LoadResetFrame** ( ) // Write Image to DMD and reset

4.4.4 Clear Block on DMD

DDC_ DDC4100Ctrl.**Clear**(*BlockNum,0*) // Clear block without Reset
DDC DDC4100Ctrl Ctrl.**Clear**(*BlockNum,1*) // Clear block and Reset
DDC DDC4100Ctrl Ctrl.**Reset**(*BlockNum*) // Reset block

4.4.5 Clear Entire DMD Display

DDC DDC4100CtrlCtrl.**Clear**(*17,0*) // Global clear without Reset
DDC DDC4100CtrlCtrl.**Clear**(*17,1*) // Global clear and Reset
DDC DDC4100CtrlCtrl.**Reset**(17) // Global reset

5.0 Discovery 4100 Controller Board USB DLL API Interface

Dynamic Link Libraries are Microsoft's take on shared libraries. They contain functions that are accessed on run time, and have a .DLL extension. The Discovery 4100 DLL library contains the necessary low level functions as an alternative to the ActiveX control.

5.1.1 short GetNumDev( )

This function will return the number of Discovery USB devices connected to the system. They will automatically be numbered starting at zero. For example, if you have four devices plugged in, GetNumDev( ) will return a 4. To access these devices you will pass the other functions a 0, 1, 2, or a 3 for that respective device.

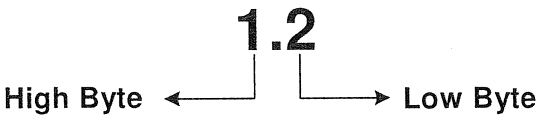
5.1.2 int GetDescriptor(int*, short DeviceNum);

This function accesses the USB device descriptor information. It will return the number of transferred bytes if successful, a -1 if the USB device failed to open, and a -2 if the device descriptor request fails.

Integer Array information:
Array[0] = bLength
Array[1] = bDescriptorType
Array[2] = bcdUSB
Array[3] = bDeviceClass
Array[4] = bDeviceSubClass
Array[5] = bDeviceProtocol
Array[6] = bMaxPacketSize0
Array[7] = idVendor
Array[8] = idProduct
Array[9] = bcdDevice
Array[10] = iManufacturer
Array[11] = iProduct
Array[12] = iSerialNumber
Array[13] = bNumConfigurations

5.1.3 unsigned int GetFirmwareRev(short DeviceNumber)

This command returns the firmware revision on the Cypress FX2 device. The low byte of the returned value contains all the digits after the decimal point and the high byte contains all the digits before the decimal point.
Example:



5.1.4 unsigned int GetDriverRev(short DeviceNumber)

Returns driver revision. Upper 16 bits contain major revision, lower 16 bits contain minor revision.

5.1.5 unsigned int GetDLLRev( )

Returns the version information for this DLL. The low byte of the returned value contains all the digits after the decimal point and the high byte contains all the digits before the decimal point.

5.1.6 unsigned int GetFPGARev(short DeviceNumber)

Returns the version of the firmware running on the D4100 applications FPGA. The low byte of the returned value contains all the digits after the decimal point and the high byte contains all the digits before the decimal point.

5.1.7 short int GetUsbSpeed(short DeviceNumber)

Used to determine if the device is plugged into a high-speed (USB 2.0) or low speed (USB 1.1) USB port. Will return a 0 if the port is low-speed, a 1 if the device is high-speed.
Negative values indicate errors:
-1 USB Device failed to open
-2 Device Descriptor request failed
-3 USB speed value differs from expected

5.1.8 int program_FPGA(uchar* write_buffer, long write_size, short int DeviceNumber)

This function is used to program the FPGA. The FPGA configuration file must first be loaded into a UCHAR (1 byte per element) array. The write_size is in bytes. Program control will remain with the DLL until the board has finished programming the FPGA.

5.1.9 short LoadControl(short DeviceNumber)

Will load control registers into DMD. this can be used for all non data transaction such as, resets and clears. Returns a 1 is successful.

5.1.10 int LoadData(uchar* RowData, unsigned int length, short DMDType, short DeviceNumber)

This function is used to load row data into the DMD. Data must be in a UCHAR array of size length. The short DMDType should be the Type returned by short GetDMDTYPE(short DeviceNumber) [see 5.1.36]. No more than 500 rows can be loaded at a time(96Kbit for 1080p, 51.2Kbit for XGA). To load an entire DMD call this function multiple times. Returns a 1 is successful.

5.1.11 short ClearFifos(short DeviceNumber)

Resets hardware receiving FIFO buffers. This should be used to put the device into a know state before sending DMD image data. Returns 1 if successful.

5.1.12 short SetBlkMd(short value, short DeviceNumber)

Sets the BLKAD register of the D4100. Returns a 1 if successful and a negative value on failure.

5.1.13 short GetBlkMd(short DeviceNumber)

Gets the BLKMD values from the D4100 register.

5.1.14 short SetBlkAd(short value, short DeviceNumber)

Sets the BLKAD register of the D4100. Returns a 1 if successful and a negative value on failure.

5.1.15 short GetBlkAd(short DeviceNumber)

Gets the BLKMD values from the D4100 register.

5.1.16 short SetRST2BLKZ(short value, short DeviceNumber)

Sets the Reset Two Blocks flag on the D4100. Active = 0, inactive = 1 (default). Returned values are 1 if successful or 0 if unsuccessful.

5.1.17 short GetRST2BLKZ(short DeviceNumber)

Gets the status of the Reset Two Block flag from the D4100

5.1.18 short SetRowMd(short value, short DeviceNumber)

Sets the ROWMD register of the D4100. Returns a 1 if successful and a negative value on failure.

5.1.19 short GetRowMd(short DeviceNumber)

Gets the ROWMD values from the D4100 register.

5.1.20 short SetRowAddr(short value, short DeviceNumber)

Sets the ROWAD register of the D4100. Returned values are 1 if successful or 0 if unsuccessful.

5.1.21 short GetRowAddr(short DeviceNumber)

Sets the ROWAD register of the D4100.

5.1.22 short SetSTEPVCC(short value, short DeviceNumber)

Sets the Step VCC flag on the D4100. Active = 1, inactive = 0 (default). Returned values are 1 if successful or 0 if unsuccessful.

5.1.23 short GetSTEPVCC(short DeviceNumber)

Gets the status of the Step VCC flag from the D4100.

5.1.24 short SetCOMPDATA(short value, short DeviceNumber)

Sets the Complement Data flag on the D4100. Active = 1, inactive = 0 (default). Returned values are 1 if successful or 0 if unsuccessful.

5.1.25 short GetCOMPDATA(short DeviceNumber)

Gets the status of the Complement Data flag from the D4100.

5.1.26 short SetNSFLIP(short value, short DeviceNumber)

Sets the North South Flip flag on the D4100. Active = 1, inactive = 0 (default). Returned values are 1 if successful or 0 if unsuccessful.

5.1.27 short GetNSFLIP(short DeviceNumber)

Gets the status of the North South Flip flag from the D4100.

5.1.28 short SetWDT(short value, short DeviceNumber)

Sets the Step Watch Dog Timer flag on the D4100. Active = 1 (default)., inactive = 0 . Returned values are 1 if successful or 0 if unsuccessful.

5.1.29 short GetWDT(short DeviceNumber)

Gets the status of the Watch Dog Timer flag from the D4100.

5.1.30 short SetPWRFLOAT(short value, short DeviceNumber)

Sets the Power Float flag on the D4100. Active = 1, inactive = 0 (default). Returned values are 1 if successful or 0 if unsuccessful.

5.1.31 short GetPWRFLOAT(short DeviceNumber)

Gets the status of the Power Float flag from the D4100.

5.1.32 short SetEXTRESETENBL(short value, short DeviceNumber)

This will enable or disable GPIOA.0 as an external reset input. Enabled = 1, disabled = 0 (default).

GPIOA.0 is a 2.5V CMOS input. Refer to TI DN 2509510 for signal location. When enabled, all software control of the DMD resets is disabled. Reset operation as defined by RST2BLKZ, BLK_MD, and BLK_ADDR will be initiated on rising edge of external reset input. Returned values are 1 if successful or 0 if unsuccessful.

5.1.33 short GetEXTRESETENBL(short DeviceNumber)

Get the current External Reset Enable value. Enabled = 1, disabled = 0 (default).

5.1.34 short SetGPIO(short value, short DeviceNumber)

Set the output values of GPIOA.2-4 to provide programmable digital outputs. Bits 4,3,2 of value control the output state. Bits 7,6,5,1,0 of value are not used. GPIOA.2-4 are 2.5V CMOS outputs. Refer to TI DN 2509510 for signal location.

5.1.35 short GetGPIO(short DeviceNumber)

Returns the values of GPIOA.5-7 as digital inputs. Bits 7,6,5 of returned value are the input data. Bits 4,3,2,1,0 of returned value are not used. 2.5V CMOS inputs. Refer to TI DN 2509510 for signal location.

5.1.36 short GetDMDTYPE(short DeviceNumber)

Returns DMD type as listed below:

DMD TYPE :	
.95 1080p Type A	0
.7 XGA Type A	1
.55 XGA Type A	2
.55 XGA Type X	3

5.1.37 short GetDDCVERSION(short DeviceNumber)

Returns DDC Version 2-0 in return value bits 2,1,0.

5.1.38 short GetRESETCOMPLETE(int waittime, short int DeviceNumber)

Will enable external reset (see 5.1.32) then loop waittime in milliseconds for an external reset trigger event to happen. If external reset occurs in the waittime windows, Returns a 1. If waittime is 0, loop will run indefinably until a reset happens.

5.1.39 short GetGPIORESETCOMPLETE(short DeviceNumber)

The will create a 1 us pulse on GPIOA.1. This can be used to trigger external hardware from the PC. Returns a 1 if successful.