Application Report
DLPA014C–July 2010–Revised June 2013

DLPC200 API Programmer's Guide

1 Purpose

This document is designed to be a how-to guide for software developers who are interested in writing their own custom applications using the DLP LightCommander™ API library.

For more information on the DLP LightCommander, see these links:

http://www.ti.com/DLPLightCommander
http://www.logicpd.com/products/development-kits/dlp-lightcommander-development-kit

2 Background

The DLP LightCommander Development Kit includes a highly versatile optical light engine comprised of the following:

- Projection lens
- Core optics module
- DLP5500 DMD board featuring 0.55” XGA DMD
- Illumination module featuring R,G,B,Infrared LEDs
- Controller board featuring the DLPC200 digital controller chip
- Industry-standard interfaces

Figure 1. DLP LightCommander™

The kit is designed to allow proof-of-concept validation of different light-processing applications, such as biometrics, 3D scanning, machine vision, 3D optical measurement, and security and surveillance.

The LightCommander Development Kit is highly programmable and includes graphical user interface (GUI) software written by Logic. This PC software GUI interfaces with dynamic link libraries (DLLs) that create DLPC200 configuration data and allow real-time communication/control of the kit (for further details, see “DLP LightCommander Control Software User Manual”). Support for the DLP® LightCommander™ Development Kit is provided by Logic PD. To create an account, register your kit, and access downloads, please visit www.logicpd.com.

Examples of DLPC200 configuration data include

- Image processing look-up-tables (LUTs)
- Register settings
- User-defined image data

Some of the communication and control operations supported by software include:

- Writing look-up-tables and (limited) registers
- Reading system status
- Controlling the LED driver
- Downloading image files to the DLPC200's image buffer memory
- Controlling the DLP display
- Selecting input data ports, internal test patterns, and VSYNC triggers
- Controlling the output sync signals (e.g. camera input trigger)
- Programming EDID

The remainder of this document will focus on the process customers would follow in order to write their own control software application.

3 System Architecture Overview

The figure below shows a high-level view of the system architecture.

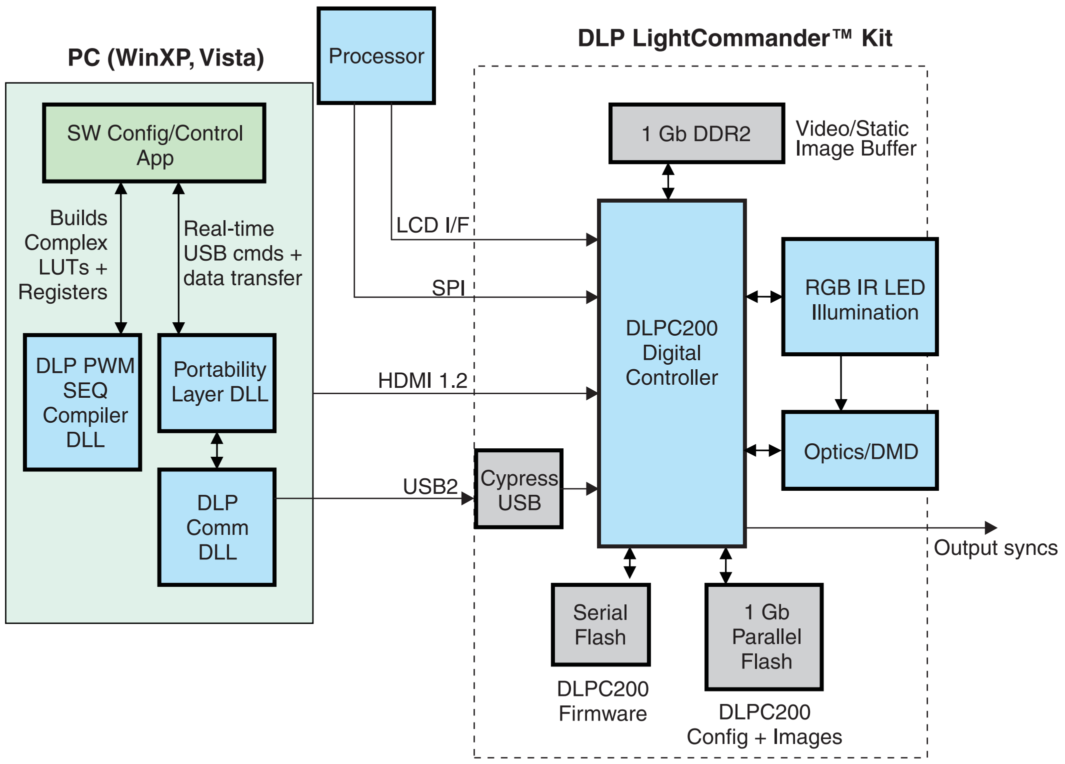


Figure 2. System Architecture

The DLPC200 is a configurable controller chip that requires various image processing Look-Up-Tables (LUTs) and register settings for proper operation. Because of the programmable nature of this chip, the DLP LightCommander Kit includes application software written to both control and configure the DLPC200 controller chip. This software has been written by LOGIC and is called "DLP LightCommander Control Software" (please see the documents on the LOGIC site for further details).

Figure 3 shows the major software components and their corresponding responsibilities. In this figure, a single software application has been written to perform both the control and configure operations, although it should be noted that this software architecture supports the development of custom control applications via interfacing with the PortabilityLayer DLL. The Portability DLL contains a library of functions that allow real-time interaction with the DLPC200 Controller Chip. The Portability DLL is not directly responsible for the configuration of the DLPC200 Controller Chip, but can indirectly program the chip by executing a configuration batchfile (more on this later).

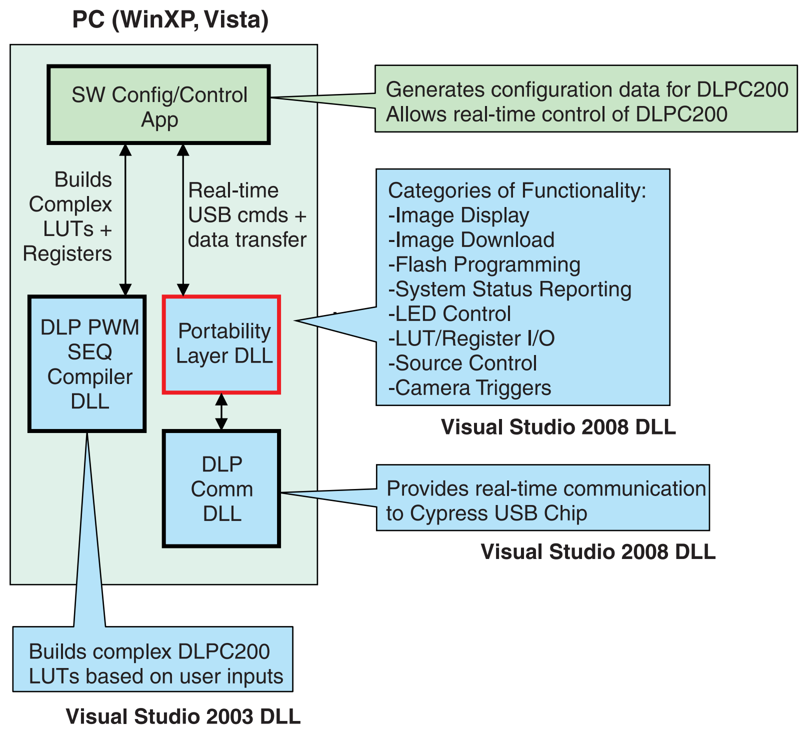


Figure 3. Software Architecture

The DLP USB Communication DLL is a wrapper around the CYUSB DLL provided by Cypress Semiconductor for their CY7C68013 USB peripheral controller chip. The PortabilityLayer DLL works in conjunction with the DLP Comm DLL to allow real-time USB communication to the Kit.

For the initial release of DLP LightCommander, Interfacing to the DLP PWM Sequence Compiler DLL is not supported.

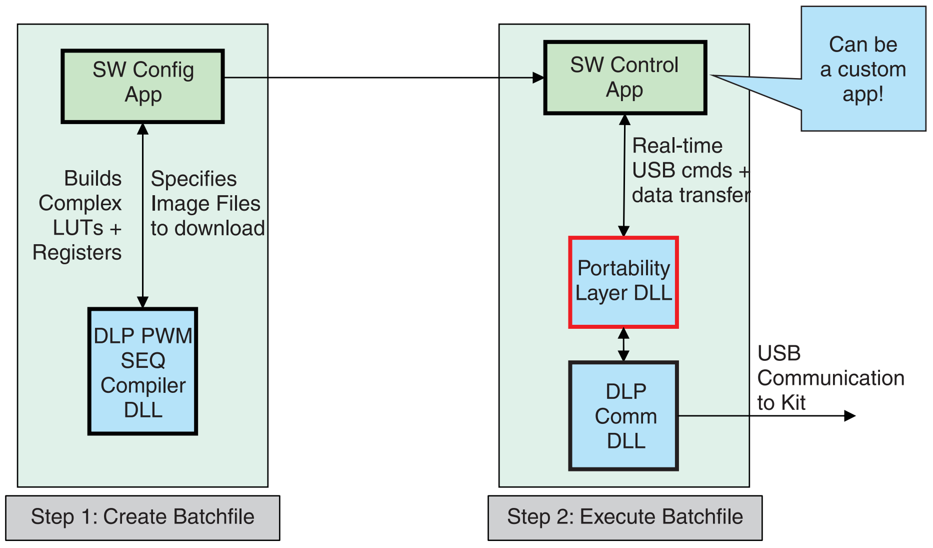
NOTE: The current version of the DLP Comm DLL 2.0 supports USB and SPI communication.

4 DLPC200 Configuration Batchfiles

As mentioned above, the DLPC200 is a highly configurable controller chip and thus requires configuration data that exists in the form of LUTs, register settings, and image data. The mechanism used to convey this configuration data is a configuration batchfile, which is a plaintext file containing commands to program DLPC200's LUTs, registers, and user-defined image data.

As shown in Figure 4, configuring the DLPC200 Controller Chip is accomplished via the following two steps:

1. Create a configuration batchfile using the software Configuration Application
2. Execute the configuration batchfile using the software Control Application. This operation is accomplished by calling the RunBatchfile API from the PortabilityLayer DLL (see the *DLP LightCommander API Reference Manual*, literature number DLPA024., for further details).



A Config and Control apps can be different software applications

Figure 4. Real-Time Configration and Control Flow

It should be noted that Figure 4 shows a conceptual view of the system, where the Config and Control Apps are shows as separate entities, although they can exist as a single application as was mentioned in the previous section.

Users wanting to customize the behavior of the control software are free to develop their own application-specific control apps by interfacing with the PortabilityLayer DLL as was mentioned in the previous section.

For more information on configuration batchfiles and the batchfile language specification, see the *DLP LightCommander API Reference Manual*, literature number DLPA024..

5 DLP LightCommander API Functions

The PortabilityLayer DLL contains a large collection of API functions to control the DLP LightCommander Kit. These functions can be classified based on the type of operation they perform. The different categories for the API are listed below:

- Display APIs
- Flash Program APIs
- Image Transfer APIs
- LED APIs
- MISC APIs
- LUT& Register Read/Write APIs
- Data Source APIs
- Video Test Pattern Generator APIs
- Sync APIs
- Status APIs
- Batchfile APIs
- Flash Compile APIs

Each of these categories will now be described to help provide an overview of the functionality provided by the API. For details on the actual functions contained within a category, please refer to the *DLP LightCommander API Reference Manual*, literature number DLPA024.

5.1 Display API

The Display APIs are used to control the display and appearance of the projected image. When displaying structured light patterns using the DLPC200's external image buffer memory, some of the Display APIs can be used to start/stop/step through the image data.

Other "Display" APIs can be used to park/unpark the DMD, flip the image horizontally/vertically, etc. The "Park DMD" operation can be used to put the display in "standby mode", where the illumination source is disabled and all of the DMD mirrors are forced to a known state.

5.2 Flash Program APIs

The Flash Program APIs are used for programming the serial or parallel flash devices. The serial flash contains DLPC200 firmware, while the parallel flash contains the user configuration settings (e.g. LUTs, registers, SL image data, etc.) for the DLP Digital Controller that are applied upon power up or when requested.

Currently, the DLP LightCommander Control software written by LOGIC produces a binary parallel flash file, which can then be downloaded.

5.3 Image Transfer APIs

The Image Transfer APIs are used to transfer binary or grayscale image data to the DLPC200's external frame buffer memory, which can then be used to display in structured light mode.

The downloaded image patterns are displayed using the display order defined by the Image Order LUT API. Using this API, the display order of the images can quickly be changed without having to re-download the set of image data.

For example, if 10 binary images were downloaded to the frame memory, their display order could be defined to be 0,1,2,…9. However, the Image Order LUT API could be called in order to display the images in the reverse order by sending: 9,8,7,…0.

See the *DLP LightCommander API Reference Manual*, literature number DLPA024., for details on the *WriteImageOrderLUT* API.

5.4 LED APIs

The LED APIs are used for controlling the LED illuminators contained inside the light engine. These APIs allow each of the four LEDs (e.g. R, G, B, IR) to be individually enabled or disabled. They also allow the light intensity of the four LEDs to be independently controlled in real-time.

MISC APIs

The Misc APIs are used for miscellaneous functions that don't naturally fit into any of the other categories. They can be used for API initialization, API logging, programming the EDID PROM, etc.

5.5 LUT and Register Read/Write APIs

The Register/LUT APIs are used for configuring the DLP Digital control chip. For the most part, the low-level Write-Register APIs are not intended to be called directly. More typically, they may be called while in the process of running a configuration batchfile. Configuration batchfiles are created by the LOGIC software application (see Batchfile APIs below for details on configuration batchfiles).

5.6 Data Source APIs

The Data Source APIs are used to configure the input data ports (e.g. pixel data) and external/internal VSYNC selection. Please refer to TI documentation for further details.

The Trigger APIs are used to define the characteristics of the external VSYNC triggers (from the perspective of the DLP Digital Controller). External VSYNC triggers are typically used in structured light mode to control the display's frame rate.

5.7 Video Test Pattern Generator APIs

The DLPC200 Controller has an internal Test Pattern Generator that is available when running in video mode.

The TPG APIs are used to define internal test patterns based on pattern type, color, and spatial frequency. The spatial frequency parameter is loosely equivalent to the number of repetitions of the pattern per screen. For example, if a horizontal ramp is selected, and spatial frequency = 2, then two horizontal ramps will be displayed on the screen.

5.8 Sync APIs

The Sync APIs are used for conditioning the output sync signals. Output syncs can be used to synchronize an external object like a camera (e.g. the camera can be “slaved” to the LightCommander Projector). Each sync is represented as a pulse and can be controlled in real-time to adjust its delay, (pulse) width, and polarity.

For debugging purposes, SYNC1 and SYNC2 can be used as “start of frame” markers when debugging with an oscilloscope (e.g. set o-scope to trigger off of SYNC1). For structured light mode, SYNC3 has a pulse width that matches the total illumination of the LEDs that are used.

The output syncs are accessed by connecting to the 3 BNC connectors on the outside of the kit (see Figure 5).

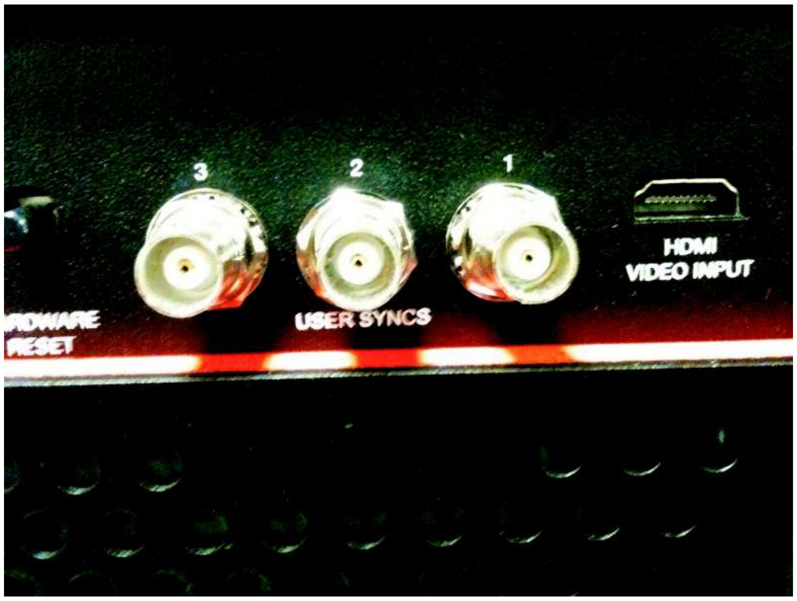


Figure 5. BNC Connectors for Output Syncs

5.9 Status APIs

The Status APIs are used for retrieving system status information. See the *DLP LightCommander API Reference Manual*, literature number DLPA024., for a full list of status APIs).

5.10 Batchfile APIs

The Batchfile APIs are used to process the commands found within a configuration batchfile, which is currently generated by LOGIC's application software and included with the DLP LightCommander Kit. A batchfile will contain directives to load the DLPC200's image processing LUTs, registers, image files, etc.

Please see the DLP LightCommander API Reference Manual for information on batchfiles and for the Batchfile Language Specification.

5.11 Flash Compile APIs

The Flash Compile APIs are used when converting a configuration batchfile (produced by the LOGIC application software) into binary data. This binary data can then be stored in the parallel (="user") flash for power-up configuration of the DLP Controller Chip. It should be noted, that normally a configuration batchfile sends commands to the DLP Controller Chip across USB or SPI, but in this context, these communication packets can be 'compiled' into binary data, then stored in parallel flash and used to configure the DLPC200 Controller on power-up.

These APIs are for advanced users and are intended to be used with the Batchfile APIs and application software that builds the binary parallel flash file.

6 Building Custom Apps using PortabilityLayer.dll

The following software components will be needed in order to build custom software applications for DLP LightCommander:

- PortabilityLayer.h – header file for PortabilityLayer.DLL
- PortabilityLayer.dll – API for interfacing with the DLPC200 Controller Chip
- PortabilityLayer.lib – import library for PortabilityLayer.DLL
- DLPcommDLL.dll – DLP Communications DLL

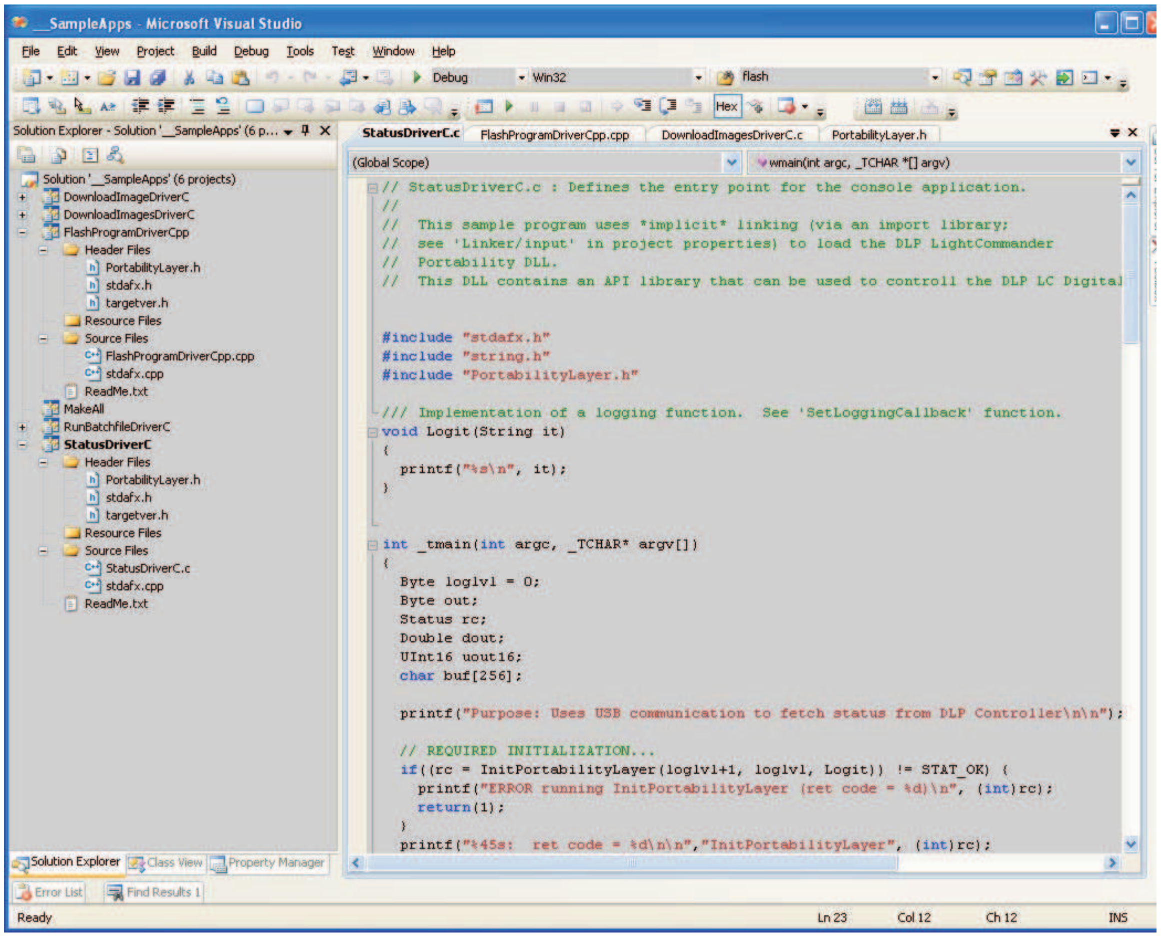
You can obtain these software components by choosing one of the following:

- Install the DLP LightCommander Control software and naviagate to the “SDK” directory (e.g. C:\Program Files\DLP LightCommander Control Software\SDK) to find the above components
- Download the Visual Studio 2008 sample solution. This download is available on www.ti.com. Search for "DLPR200" on www.ti.com to see the table of available support software. The download titled "DLPR200APP" with the name "Reference Application Source Code for DLPC200 API" has the files for the sample solution. Once the download is complete the files can be found in C:\Program Files\Texas Instruments-DLP\DLP LightCommander API Ref Apps.

7 Visual Studio Sample Projects

As mentioned above, sample Visual Studio 2008 projects are available from TI. At the present time, the sample applications exist as independent projects within the __SampleApps solution (see Figure 6). This solution currently consists of the following projects:

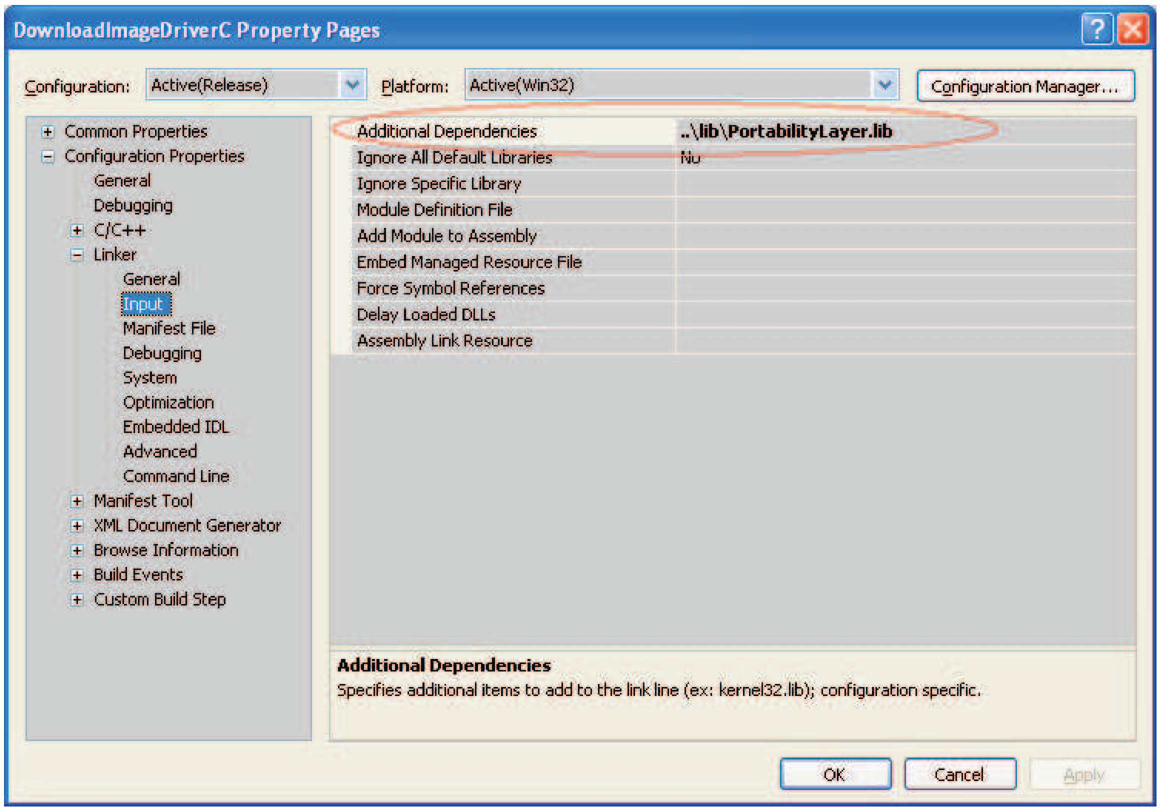
- DownloadImageDriverC: sample application written in ‘C’ that allows a single DBI image file to be downloaded to the DLPC200’s static image buffer memory
- DownloadImagesDriverC: sample application written in ‘C’ that allows multiple DBI image files to be downloaded to the DLPC200’s static image buffer memory
- FlashProgramDriverCpp: sample application written in C++ that allows programming of both the serial and parallel flash devices, while also supporting erasing of the parallel flash. It should be noted that this driver file using ‘C’ syntax, but is compiled using the C++ compiler, which helps demonstrate how either a C or C++ application can be compiled to use PortabilityLayer.dll.
- RunBatchfileDriverC: sample application written in ‘C’ that executes configuration batchfiles, which are used to configure the DLPC200 with image processing LUTs, register settings, and image data.
- StatusDriverDriverC: sample application written in ‘C’ that queries the DLPC200 for system status settings



```
// StatusDriverC.c : Defines the entry point for the console application.
//
//  This sample program uses *implicit* linking (via an import library;
//  see 'Linker/input' in project properties) to load the DLP LightCommander
//  Portability DLL.
//  This DLL contains an API library that can be used to controll the DLP LC Digital


#include "stdafx.h"
#include "string.h"
#include "PortabilityLayer.h"

/// Implementation of a logging function.  See 'SetLoggingCallback' function.
void Logit(String it)
{
  printf("%s\n", it);
}


int _tmain(int argc, _TCHAR* argv[])
{
  Byte loglvl = 0;
  Byte out;
  Status rc;
  Double dout;
  UInt16 uout16;
  char buf[256];

  printf("Purpose: Uses USB communication to fetch status from DLP Controller\n\n");

  // REQUIRED INITIALIZATION...
  if((rc = InitPortabilityLayer(loglvl+1, loglvl, Logit)) != STAT_OK) {
    printf("ERROR running InitPortabilityLayer (ret code = %d)\n", (int)rc);
    return(1);
  }
  printf("%45s:  ret code = %d\n\n","InitPortabilityLayer", (int)rc);
```

Figure 6. Sample Apps Using Microsoft Visual Studio 2008

An example of implicit linking using the provided import library (PortabilityLayer.lib) is shown below in Figure 7. This Visual Studio 2008 dialog is accessible by viewing the “Linker” section under “Configuration Properties”. The use of implicit linking allows the application software to use simple function calls, thus avoiding the use of explicit linking via calls to Microsoft’s LoadLibrary and GetProcAddress functions.



Configuration: Active(Release) Platform: Active(Win32)

Additional Dependencies	..\lib\PortabilityLayer.lib
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

Figure 7. Implicit Linking Using .lib file

8 Compiling API V2.0 Source Code for SPI Communication

The API has been updated for v2.0 to support communication using SPI. Prior versions of the API only supported USB communication (using Cypress Semiconductor's supplied USB SW libraries).

NOTE: API V2.0 requires the DLPC200 firmware HW V#1.0.363 and MCU V#2.1.2. Please go to the TI website and download/install the corresponding installer before using API V2.0.

9 API V2.0 Software Architecture

Figure 8 shows the major functional SW components for the API:

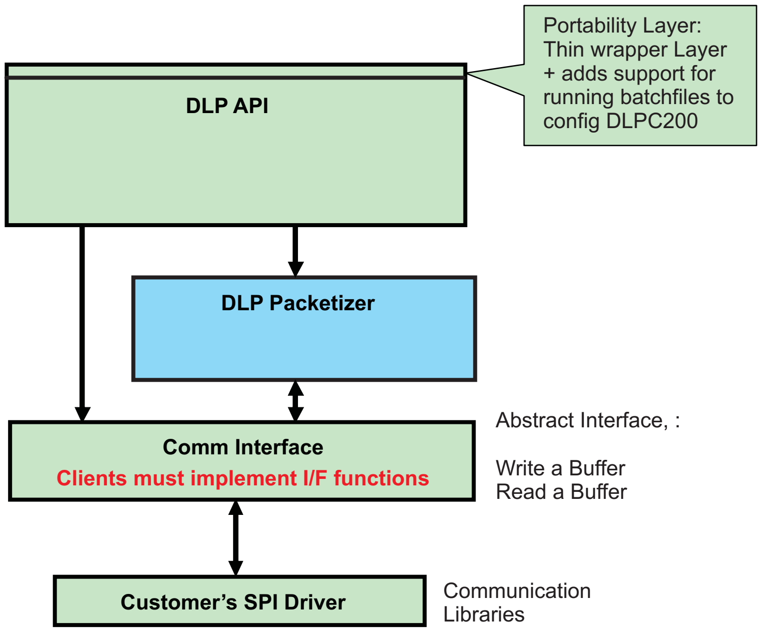


Figure 8.

Notice that the Communications Interface Layer, “Comm Interface”, requires customers to implement a few key interfaces to enable SPI communication. For USB communication, these functions are implemented, but for SPI communication they must be implemented by customers.

The two functions that are required to be implemented are:

CommIF_SendBufferAcrossCommChannel: writes a buffer with a variable number of bytes across the communication channel to the DLPC200.

CommIF_ReceiveBufferFromCommChannel: reads a buffer containing a variable number of bytes from the DLPC200.

These two functions are essentially abstract interfaces to send and receive a buffer of data. These functions are defined in CommInterface.c.

10 API V2.0 Source Files

A new installer has been created which contains the API source files. Please go to the DLP LightCommander website to download the API Source Installer. As of this writing, the website is located at:

Please note that a few of these files (e.g. omap35xx_mcspi.h, etc.) are required for compiling using Microsoft Visual Studio 2005 for a TI OMAP3530 / WinCE 6.0 platform. You can safely ignore these files if you are using a different platform.

11 Detailed Steps for Compiling API V2.0

1. Download the TI API source installer from the TI website. Extract the contents of the TI zip file (DLP_LC_Src_Files*.zip), placing the API source files in an appropriate location. It is recommended all of the API source files be saved in the same directory for simplicity.
2. For SPI Communication, make sure the #define BUILD_USB is not active. See CommInterface.h. Make sure to implement these functions:
   - CommIF_SendBufferAcrossCommChannel
   - CommIF_ReceiveBufferFromCommChannel

3. Follow the guidelines outlined in the SPI App note document, which are shown in the 2 CommInterface.c SPI functions written for WinCE: _SPI_SendBuffer, _ReceiveBuffer_SpiWinCE. This document and functions show the correct method for sending dummy bytes to end/begin the communication for Sending/Receiving data, respectively. The SPI app note document is available from the API Source installer.
4. Create custom SW driver application. For examples, see the V2.0 sample app DlpApiUsingSPI.c or any of the other previously provided sample apps. It should be noted that DlpApiUsingSPI.c was built using Microsoft's Visual Studio 2005 for a TI OMAP3530 / WinCE 6.0 platform.
5. In CommInterface.c, implement the function that waits until the DLPC200's MCU is available:

   ```
   void CheckProcessorBusySignal()
   {
   // TODO: Query Processor's "BUSY SIGNAL" and wait until it is not busy...
   Sleep(10); // for now, sleep 10 msec
   }
   ```
6. In CommInterface.c, replace any "Sleep" calls with calls to "CheckProcessorBusySingal".
7. For compiling, no extra preprocessor flags need to be defined. For linking using a Makefile, make sure to link with the "-lm -lc" libraries (e.g. math & standard C libraries).

See the following section for HW details on connecting the SW_BUSY signal to the DLPC200.

12 DLPC200 MCU BUSY SPI Signal

Normally, SPI uses a 4-wire scheme for communicating: MOSI, MISO, Slave Select, SPI CLK. For the DLP LightCommander, an additional signal is added, SLAVE_SPI_ACK that can be monitored by the master to check if the slave (DLPC200 MCU) is busy. The SLAVE_SPI_ACK signal is active high.

The schematic for the DLP LightCommander Controller Board is available from the LOGIC website: www.logicpd.com. The DLPC200 SLAVE_SPI_ACK signal can be found on the 40-pin J505 connector as pin number 36 (EXP_CON_SPI_nACK).

TI strongly recommends that this SPI_ACK signal be used when using SPI communication. See the DLPC200 datasheet for further information.

The maximum value for SPI_CLK is 5 MHz.

13 API V2.0 Testing Performed by TI

Full testing of the API was performed on a TI OMAP3530 / WinCE 6.0 platform. For production systems, customers should check the SW_BUSY signal before sending data across the communications channel. For testing, fixed delays were used without any issues or the SW_BUSY checks as described above. It should be noted that some of the APIs that transfer large amounts of data from the parallel flash to the external frame buffer memory take longer to execute, so using fixed delays for these types of APIs may not always work.

Disclaimer: The API source was also compiled on a RedHat Linux platform, **but not tested** due to hardware limitations.

14 References

A reference guide, *DLP LightCommander API Reference Manual*, literature number DLPA024. has been created to accompany this document.

Additional documentation from TI is forthcoming. Check the TI DLP LightCommander link for updates.

Product information, User Manuals, and HW documents will provide additional background information, and are available from LOGICs DLP LightCommander link.

Note that LOGIC's DLP LightCommander Control software has a functional system block diagram of the DLPC200 Digital Controller.

Revision History

Changes from A Revision (August 2010) to B Revision **Page**

- Changed document for new v2.0 API, which adds support for custom SPI communication. As of v2.0, the API is available as "C" source. 5

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Revision History

Changes from B Revision (July 2010) to C Revision **Page**

- Deleted For V2.0, the API exists as C source code, which allows it to be compiled for any platform/processor/OS, and can be updated or interfaced to any desired SPI driver SW. It should be noted that this source code can also be compiled for USB communication as well, but as of this writing, no sample projects are supplied for this option since they already exist using DLL components. 10

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

IMPORTANT NOTICE

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		