DLP LightCommander API

アプリケーション・レポート

Literature Number: JAJA224
February 2011

DLP LightCommander API

1 DLP LightCommander API リファレンス・マニュアル

1.1 概要

DLP LightCommander API（DLPLC API）は、DLP Light Commanderキットおよびチップセットを制御するための各種ソフトウェア（SW）機能を提供します。チップセットにはDLPC200コントローラが含まれ、各種の画像処理機能および制御機能を実行します。

クライアントでは、DLPLC APIを使用して、画像データのダウンロード、ステータスの読み出し、レジスタやルックアップ・テーブル（LUT）の設定など、DLPC200のさまざまな機能を制御できます。また、DLPLC APIは、Cypress CY7C68013 USB周辺コントローラ・チップを使用して、USB 2.0通信をサポートします。

設定バッチファイル内のコマンドを実行することにより、LUTおよびレジスタのリアルタイム・ロードが行われます。これは、RunBatchFile APIを実装するSWアプリケーションによって実行されます。このAPIでは、バッチファイル・コマンドからAPI関数への内部マッピングにより、対応するコマンドが検出されたときにAPI関数を呼び出すことができます。詳細については、「バッチファイル言語仕様V1.0」Batchfile Language Specification V1.0 を参照してください。

各種の詳細情報については、以下のリンクからオンライン・リソースおよび関連データ・シートを参照してください。

- API関数 — DLP LightCommander API関数」を参照 DLP LightCommander API Functions
- APIデータ型 — DLP LightCommander APIのデータ型、変数、定数」を参照 DLP LightCommander API Data Types, Variables, and Constants
- バッチファイル構文 — バッチファイル言語仕様V1.0」を参照 Batchfile Language Specification V1.0
- バージョン番号 — APIのバージョン番号およびリリース・ノート」を参照 API Version Number and Release Notes
- オンライン・リソース:
  - http://www.ti.com/DLPLightCommander
  - http://www.logicpd.com/products/development-kits/dlp-lightcommander-development-kit

2 モジュール索引

- DLP LightCommander API関数
  - 表示API フラッシュ・コンパイルAPI
  - フラッシュ・プログラムAPI
  - 画像転送API
  - LED API
  - 各種API
  - レジスタおよびLUT読み取り/書き込みAPI
  - ビデオ・テスト・パターン生成API データ・ソースAPI
  - 同期API
  - ステータスAPI
  - バッチファイルAPI

3 ファイル索引

文書化されているファイルの一覧を示します。

- PortabilityLayer.h

4 モジュール説明

4.1 DLP LightCommander API関数

モジュール

- 表示API
- フラッシュ・コンパイルAPI
- フラッシュ・プログラムAPI
- 画像転送API
- LED API
- 各種API
- レジスタおよびLUT読み取り/書き込みAPI
- データ・ソースAPI
- ビデオ・テスト・パターン生成API
- 同期API
- ステータスAPI
- バッチファイルAPI

4.2 表示APIs

関数

- UINT8 DLP_Display_DisplayPatternManualStep ()
- UINT8 DLP_Display_DisplayPatternManualForceFirstPattern ()
- UINT8 DLP_Display_DisplayPatternAutoStepForSinglePass ()
- UINT8 DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses ()
- UINT8 DLP_Display_DisplayStop ()
- UINT8 DLP_Display_ParkDMD ()
- UINT8 DLP_Display_UnparkDMD ()
- UINT8 DLP_Display_SetDegammaEnable (UINT8 enableBit)
- UINT8 DLP_Display_HorizontalFlip (UINT8 enableBit)

- UINT8 DLP_Display_VerticalFlip (UINT8 enableBit)

表示APIは、画像データの表示および外観の制御に使用されます。外部フレーム・メモリから構造化ライト(SL)パターンを表示する場合、いくつかの表示APIを使用して、各SL画像の開始、停止、およびステップ表示を行えます。

4.2.1 UINT8 DLP_Display_DisplayPatternManualStep ()

ImageOrder LUTで定義される次の構造化ライト画像を繰り返し表示します。

ビット平面パターンが外部メモリにロードされ、DLP制御チップの設定が完了した後で、この関数を呼び出すことにより、DMDおよびDLP制御チップに対して、ImageOrder LUTに定義されたパターンを表示するよう指示できます。

表示できる画像パターンが複数ある場合は、この関数を(繰り返し)呼び出すことで、要求に応じて各パターンを表示できます。内部でプログラミングされたVSYNCを使用する場合は、1つのパターンが連続して表示され、(人間による)目視検査が可能になります。

戻り値

成功時は0

4.2.2 UINT8 DLP_Display_DisplayPatternManualForceFirstPattern ()

ImageOrder LUTで定義される最初のSL画像を繰り返し表示します。

ビット平面パターンが外部メモリにロードされ、DLP制御チップの設定が完了した後で、この関数を呼び出すことにより、DMDおよびDLP制御チップに対して、ImageOrder LUTに定義された最初のパターンを表示するよう指示できます。

この関数をDLP_Display_DisplayPatternManualStepとともに使用することにより、DLP_Display_DisplayPatternManualStep ImageOrder LUTの先頭から開始して、対応する各イメージを順番に表示できます。

戻り値

成功時は0

4.2.3 UINT8 DLP_Display_DisplayPatternAutoStepForSinglePass ()

開発中:この関数は現時点では完全に実装されていません。

ビット平面パターンが外部メモリにロードされ、DLP制御チップの設定が完了した後で、この関数を呼び出すことにより、DMDおよびDLP制御チップに対して、パターンを表示するよう指示できます。

表示できる画像パターンが複数ある場合は、この関数を(1回)呼び出すことで、一連のパターンを表示できます。表示するパターンが10個ある場合、この関数を呼び出すと、10個すべてのパターンが表示された後で、表示がディスエーブルになります。

戻り値

成功時は0

4.2.4 UINT8 DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses ()

ImageOrder LUTで定義されるすべての構造化ライト画像(1フレームごとに1画像)を連続して表示します。

ビット平面パターンが外部メモリにロードされ、DLP制御チップの設定が完了した後で、この関数を呼び出すことにより、DMDおよびDLP制御チップに対して、パターンを表示するよう指示できます。

表示できる画像パターンが複数ある場合は、この関数を(1回)呼び出すことで、一連のパターンを繰り返し表示できます。表示するパターンが10個ある場合、この関数を呼び出すと、10個すべてのパターンが表示された後で、表示を繰り返します。

戻り値

成功時は0

4.2.5 UINT8 DLP_Display_DisplayStop ()

Tこの関数は、ビット平面パターンの表示をディスエーブルにします。

戻り値

成功時は0

4.2.6 UINT8 DLP_Display_ParkDMD ()

パークDMDコマンドを発行します。

戻り値

成功時は0

4.2.7 UINT8 DLP_Display_UnparkDMD ()

パーク解除DMDコマンドを発行します。

戻り値

成功時は0

4.2.8 UINT8 DLP_Display_SetDegammaEnable (UINT8 enableBit)

デガンマ機能をイネーブル/ディスエーブルにします。

デガンマ機能は一般にビデオ・データに関連付けられ、(適用されたガンマ伝達関数を除去することで)光出力を線形化するために使用されます。

デガンマ機能をイネーブルにする場合は、あらかじめ対応するLUTが非ゼロ値でプログラミングされている必要があります。デガンマLUTは、LOGICアプリケーションSWの出力です。

パラメータ

enableBit デガンマをイネーブルにするには0以外、ディスエーブルにするには0を設定

戻り値

成功時は0

4.2.9 UINT8 DLP_Display_HorizontalFlip (UINT8 enableBit)

表示の左右反転をイネーブル/ディスエーブルにします。

パラメータ

enableBit イネーブルにするには0以外、ディスエーブルにするには0を設定

戻り値

成功時は0

4.2.10 UINT8 DLP_Display_VerticalFlip (UINT8 enableBit)

表示の上下反転をイネーブル/ディスエーブルにします。

パラメータ

enableBit イネーブルにするには0以外、ディスエーブルにするには0を設定

戻り値

成功時は0

4.3 フラッシュ・コンパイルAPI

関数

- void DLP_FlashCompile_SetCommPacketCallback (void(*pf)(UINT8 *packetBuffer, UINT16 nBufBytes))
- void DLP_FlashCompile_FlushCommPacketBuffer ()
- void DLP_FlashCompile_SetCompileMode (UINT8 *enableBit*)
- UINT8 DLP_FlashCompile_GetCompileMode ()

フラッシュ・コンパイルAPIは、(LOGICアプリケーションSWによって生成された)バッチファイルをバイナリ・データに変換する際に使用されます。このバイナリ・データは、DLPコントローラ・チップの起動時設定用に、パラレル(="ユーザ")フラッシュに格納されます。通常、設定バッチファイルは DLPコントローラ・チップにUSBまたはSPI経由でコマンドを送信しますが、このコンテキストでは、これらの通信パケットがバイナリ・データに"コンパイル"されてから、パラレル・フラッシュに格納され、起動時のコントローラの設定に使用されます。

これらのAPIは、RunBatchFile APIとともに使用します。4.17節を参照してください。 4.17 .

4.3.1 void DLP_FlashCompile_SetCommPacketCallback (void(*)(UINT8 *packetBuffer, UINT16 nBufBytes) pf)

フラッシュ・コンパイル・コールバック関数を初期化します。

コールバック関数を設定する初期化関数です。このコールバック関数は、通信パケット・バッファを、フラッシュ出力の格納に使用するバッファへとコピーするために使用されます。

例: void CopyCommPacketBufferToFlashBuffer(UINT8* packetBuffer, UINT16 nBufBytes { ... }

4.3.2 void DLP_FlashCompile_FlushCommPacketBuffer ()

通信データを呼び出し元のクライアントへ強制的に返送します。

DLP_FlashCompile_AppendWriteRegisterFieldCmdWithMaskの1回または複数回の呼び出しによって部分的に埋められた通信パケット・バッファの内容を、クライアントに返送します。

4.3.3 void DLP_FlashCompile_SetCompileMode (UINT8 enableBit)

"フラッシュ・コンパイル"モードをイネーブル/ディスエーブルにします。

"フラッシュ・コンパイル"モードをイネーブルにした場合、USB/SPI通信がディスエーブルになり、API呼び出しを行うと DLP_FlashCompile_SetCommPacketCallbackによって定義されたコールバック関数が呼び出され、データのバイナリ通信パケットを作成できます。このバイナリ・データを入力として、フラッシュ設定ファイルを構築できます。

パラメータ

enableBit イネーブルにするには0以外、ディスエーブルにするには0を設定

4.3.4 UINT8 DLP_FlashCompile_GetCompileMode ()

"フラッシュ・コンパイル"モード・フラグを取得します。

戻り値

=1（イネーブル）、=0（それ以外）

4.4 フラッシュ・プログラムAPI

関数

- UINT8 DLP_FlashProgram_ProgramSerialFlash (UINT32 *nSkipBytesInFlash* , UINT8 *pBuf, UINT32 *nBytesInBuf* , UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 * *out* CRC)
- UINT8 DLP_FlashProgram_ProgramParallelFlash (UINT32 nSkipBytesInFlash, UINT8 *pBuf, UINT32 nBytesInBuf, UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 *outCRC)
- UINT8 DLP_FlashProgram_EraseParallelFlash (UINT32 *nSkipBytesInFlash* , UINT32 *nBytesToErase*)
- UINT8 DLP_FlashProgram_ReadParallelFlashToFile (const char *filename, UINT32 flash_byte_offset, UINT32 tot_bytes)

フラッシュ・プログラムAPIは、システム内のシリアルまたはパラレル・フラッシュ・デバイスのプログラミングに使用されます。シリアル・フラッシュには、DLPデジタル・コントローラのファームウェアが格納され、パラレル・フラッシュには、起動時または要求に応じて適用されるDLPデジタル・コントローラの設定（LUT、レジスタ、SL画像データなど）が格納されています。

4.4.1 UINT8 DLP_FlashProgram_ProgramSerialFlash (UINT32 *nSkipBytesInFlash* , UINT8 * *pBuf,* UINT32 *nBytesInBuf* , UINT8 (*)(double completionPerCent)pf, UINT8 *verifyFlag* , UINT16 * *outCRC)*

シリアル・フラッシュに新しいDLPコントローラ・ファームウェアをプログラミングします。

P指定されたバッファの内容をシリアル・フラッシュにコピーすることにより、フラッシュ・ダウンロード操作を実行します。スキップするバイト数を指定して、書き込み開始アドレスを定義できます。

パラメータ

nSkipBytesInFlas 0を設定 将来のリリースでは、DLPコントローラ・ファームウェアの一部を更新するために0以外の値が必要となる場合もあります。

pBuf バイト・バッファへのポインタ

nBytesInBuf バッファ内のバイト数 = フラッシュに書き込むバイト数 pf function 呼び出し元に進捗状況を送信するための関数ポインタ。このコールバックが0以外を返した場合、ダウンロードはキャンセルされています。

verifyFlag CRC16の計算を行う場合は1を設定、計算を行わない場合は0を設定

outCRC 指定データに対して計算されたCRC16

戻り値 成功時は0

4.4.2 UINT8 DLP_FlashProgram_ProgramParallelFlash (UINT32 *nSkipBytesInFlash* , UINT8 * *pBuf* , UINT32 *nBytesInBuf* , UINT8(*)(double completionPerCent)pf, UINT8 verifyFlag, UINT16 * *outCRC*)

パラレル・フラッシュにDLPコントローラの設定をプログラミングします。

指定されたバッファの内容をパラレル・フラッシュにコピーすることにより、フラッシュ・ダウンロード操作を実行します。スキップするバイト数を指定して、書き込み開始アドレスを定義できます。

パラメータ

nSkipBytesInFlash 書き込み開始アドレス

pBuf バイト・バッファへのポインタ

nBytesInBuf バッファ内のバイト数 = フラッシュに書き込むバイト数

pf f呼び出し元に進捗状況を送信するための関数ポインタ。このコールバックが0以外を返した場合、ダウンロードはキャンセルされています。

verifyFlag CRC16の計算を行う場合は1を設定、計算を行わない場合は0を設定

outCRC 指定データに対して計算されたCRC16

戻り値

成功時は0

4.4.3 UINT8 DLP_FlashProgram_EraseParallelFlash (UINT32 *nSkipBytesInFlash* , UINT32 *nBytesToErase*)

パラレル・フラッシュを消去するデバッグ・ルーチンです。

パラメータ

nSkipBytesInFlash 消去を行う前にスキップするバイト数。理想的には、セクタ境界に揃える必要があります。

nBytesToErase 消去するバイト数

4.4.4 UINT8 DLP_FlashProgram_ReadParallelFlashToFile (const char **filename* , UINT32 *flash_byte_offset* , UINT32 *tot_bytes*)

パラレル・フラッシュの任意の場所をリード・バックするデバッグ・ルーチンです。

パラメータ

filename フラッシュ・データを書き込むファイルの名前。指定された名前のファイルが既に存在する場合は、そのファイルに上書きします。

flash_byte_offset フラッシュからの読み取り開始位置のオフセット（バイト数）。フラッシュ読み取りアドレスに対して使用されます。

tot_bytes 読み取る合計バイト数

戻り値

成功時は0

4.5 画像転送API

関数

- UINT8 DLP_Img_DownloadBitplanePatternToExtMem (const UINT8 *pBuf, UINT32 tot_bytes, UINT16 bpnum0b)
- UINT8 DLP_Img_DownloadBitplanePatternFromFlashToExtMem (UINT32 flash_byte_offset, UINT32 tot_bytes, UINT16 bpnum0b)
- Status WriteExternalImage (String name, Byte imageIndex)
- IntPtr GetAllBitPlanes (String name, UInt32 bpp, UInt32 *bitPlaneSize)
- Status destroyBitPlanes (IntPtr bitPlanes, UInt32 bpp)

画像転送APIは、DLPコントローラの外部フレーム・メモリにグレースケール画像データを転送するために使用されます。転送された画像は、SLモードでの表示に使用されます。

ダウンロードされた画像パターンは、ImageOrder LUT API（DLP_RegIO_WriteImageOrderLut）で定義される表示順に従って表示されます。

4.5.1 UINT8 DLP_Img_DownloadBitplanePatternToExtMem (const UINT8 *pBuf, UINT32 tot_bytes, UINT16 bpnum0b)

1つのビット平面画像をUSB/SPI経由で外部メモリ(DDR)にダウンロードします。

このAPIを使用するには、最初に入力画像データを適切な形式で用意しておく必要があります。これには、次の2つの手順が必要です。

1. 各ビット平面を個別に分割します。
2. 各バイトに8ピクセルをパックします(例:pixel#0をLSBにマップ、pixel#1をLSB+1にマップ、など)。

このAPIは、TBD LOGICで提供される画像フォーマット関数とともに使用できます。

パラメータ

pBuf パックされたビット平面バッファへのポインタ

tot_bytes ビット平面データの合計バイト数

bpNum0b 0から始まるビット平面番号。DDR書き込みアドレスの計算に使用されます。

戻り値

成功時は0

4.5.2 UINT8 DLP_Img_DownloadBitplanePatternFromFlashToExtMem (UINT32 flash_byte_offset, UINT32 tot_bytes, UINT16 bpnum0b)

1つのビット平面画像をパラレル・フラッシュから外部メモリにダウンロードします。フラッシュに格納された画像データを静的な画像バッファ・メモリにコピーします。フラッシュ・オフセットについては、バイナリ・フラッシュ・ファイルの作成時に使用したフラッシュ・レイアウト・オフセットを確認してください。グレースケール画像を転送する場合は、ビット長の各ビットについて1回ずつ、このAPIを呼び出す必要があります(例:BPP = 8のときは8回呼び出す)。

パラメータ

flash_byte_offset ビット平面データの開始位置オフセット(バイト数)。フラッシュ読み取りアドレスに対して使用されます。

tot_bytes ビット平面データの合計バイト数

bpNum0b 0から始まるビット平面番号。外部メモリ書き込みアドレスに対して使用されます。

戻り値

成功時は0

4.5.3 Status WriteExternalImage (String name, Byte imageIndex)

DBI画像ファイルを画像バッファ・メモリにダウンロードします。

DBI 画像ファイルをDLPコントローラ・チップの外部画像バッファ・メモリにダウンロードします。DBIファイルは、LOGICアプリケーションSWによる “画像インポート”プロセスで作成され、”ファイル・ヘッダ＋適切な形式の画像データ”から構成されます。DBIファイルの仕様の詳細については、SWのユーザ・マニュアルを参照してください。

パラメータ

name DBI画像のファイル名

imageIndex 画像バッファ・メモリに格納する際の書き込みアドレス(= インデックス)

戻り値

成功時はSTAT_OK

4.5.4 IntPtr GetAllBitPlanes (String name, UInt32 bpp, UInt32 *bitPlaneSize)

メモリ割り当てを行ってから、割り当てたバッファにDBI画像を読み込みます。

指定されたDBI画像ファイルを使用して、ビット平面データを保持するためのメモリ割り当てを行います。このAPIは、DBI画像データをパラレル・フラッシュに格納する必要がある場合に役立ちます。

パラメータ

name DBI画像ファイル名。DBIファイルには、ヘッダに続いて、適切な形式の画像データが格納されています。

bpp 画像のピクセルあたりのビット数

bitPlaneSize 各ビット平面画像の出力バイト数。画像データの合計サイズはbpp*bitPlaneSizeとなります。

戻り値

割り当てられた画像バッファへのポインタ(2D)。解放するには、destroyBitPlanesを呼び出します。

4.5.5 Status destroyBitPlanes (IntPtr bitPlanes, UInt32 bpp)

ビット平面メモリのメモリ割り当て解除を行います。

このAPIは、GetAllBitPlanesとともに使用します。

パラメータ

bitPlanes 割り当てられた画像バッファへのポインタ(2D)

戻り値

成功時はSTAT_OK

4.6 LED API

関数

- UINT8 DLP_LED_SetLEDintensity (LED_t LED, double intensity_perCent)
- UINT8 DLP_LED_GetLEDintensity (LED_t LED, double *intensityPerCent)
- void DLP_LED_LEDdriverEnable (UINT8 enable)
- void DLP_LED_GetLEDdriverTimeout (UINT8 *timedOut)
- UINT8 DLP_LED_SetLEDEnable (LED_t LED, UINT8 enableBit)

LED APIは、ライトエンジン内部に搭載されたLED発光回路の制御に使用されます。

4.6.1 UINT8 DLP_LED_SetLEDintensity (LED_t LED, double intensity_perCent)

LEDの輝度を設定します。

LEDドライバに対して、指定された値でLEDに電流を供給するよう要求を送信します。100%の輝度の場合、LEDは許容される最大LED電流を使用して駆動されます。

例えば、最大LED電流が10Aの場合、このAPIを値50で呼び出すと、LED電流が5A(= 10*50/100)に設定されます。

パラメータ

LED target LED illuminator (see enum)

intensity_perCent 目的のLED輝度レベル(%):0.0 < 範囲 ≤ 100.0

戻り値

成功時は0

4.6.2 UINT8 DLP_LED_GetLEDintensity (LED_t LED, double *intensityPerCent)

LEDの輝度を取得します。

パラメータ

LED 対象のLED発光回路（enumを参照）

intensityPerCent intensityPerCent 輝度（%）: 0.0 < 範囲 ≤ 100.0

戻り値

成功時は0

4.6.3 void DLP_LED_LEDdriverEnable (UINT8 enable)

LEDドライバのイネーブル制御用デバッグAPIです。

0を送信すると、LEDドライバがディスエーブルになります。0より大きな値を送信すると、イネーブルになります。

以前は、LEDドライバが“タイムアウト”してディスエーブルになった場合に、このAPIを使用して再度イネーブルにしていました。

4.6.4 void DLP_LED_GetLEDdriverTimeout (UINT8 *timedOut)

LEDドライバがタイムアウトしたかどうかを確認するデバッグAPIです。

LEDドライバがタイムアウトしている場合は、1を返します。

注：最新のOSRAM LEDドライバの場合、ドライバがタイムアウトすることはありません。

パラメータ

timedOut タイムアウト状態を保持する出力変数

4.6.5 UINT8 DLP_LED_SetLEDEnable (LED_t LED, UINT8 enableBit)

個々のLEDをイネーブルまたはディスエーブルにするデバッグAPIです。

PWMシーケンスが指定されたLEDで構築されている場合は、このAPIを使用して、そのLEDをディスエーブル（その後、再度イネーブル）にできます。

パラメータ

対象のLED発光回路（enumを参照）LEDをイネーブルにするには0以外、ディスエーブルにするには 0 を設定

戻り値

成功時は0

4.7 MISC APIs

関数

- const char * 4.7.1 ()
- void DLP_Misc_SetLoggingCallback (UINT8 logVisibilityLevel, void(*pf)(const char *))
- void DLP_Misc_SetLoggingVisibilityLevel (UINT8 logVisibilityLevel)
- UINT8 DLP_Misc_InitAPI ()
- UINT8 DLP_Misc_EnableCommunication ()
- UINT8 DLP_Misc_DisableCommunication (const char *ofileName)

- void DLP_Misc_DisableCommunication_FlushOutputFileToDisk ()
- UINT8 DLP_Misc_PassThroughExecute (const char *args[ ])
- UINT8 DLP_Misc_ProgramEDID (DMD_t DMD, UINT8 offset, UINT8 numBytes, const char *fileName)
- Status ChangeLogLevel (Byte logLevel)
- Status InitPortabilityLayer (Byte logLevel, Byte detail, OutputCallback callback)
- UINT8 DLP_Misc_GetTotalNumberOfUSBDevicesConnected (UINT8 *oNumDev)
- UINT8 DLP_Misc_SetUSBDeviceNumber (UINT8 devNum0b)
- UINT8 DLP_Misc_GetUSBDeviceNumber (UINT8 *oCurDevNum0b)

各種APIは、他のカテゴリに属さない各種の機能に使用されます。

4.7.1 const char * DLP_Misc_GetVersionString ()

4.7.2 void DLP_Misc_SetLoggingCallback (UINT8 logVisibilityLevel, void(*pf)(const char *))

APIのロギングを設定する初期化関数です。

パラメータ

loglvl ロギング詳細レベル、範囲 = {0, 1, 2}。0より大きい値を送信すると、追加のデバッグ・ロギングがイネーブルになります。

値が大きいほど、詳細なデバッグ・ロギングが行われます。

*pf*constのchar*引数とvoidの戻り値を持つ関数への関数ポインタ 。

例 1: void LogIt(const char* it) { ... }

DLP_Misc_SetLoggingCallback(0, LogIt);

Iこの例では、ロギング詳細レベルとして0が指定されているため、LogIt関数にAPIデバッグ・ロギングは送信されません。

例 2: void LogIt(const char* it) { ... }

DLP_Misc_SetLoggingCallback(1, LogIt);

この例では、ロギング詳細レベルとして1が指定されているため、LogIt関数に“レベル1”のAPIデバッグ・ロギングが送信されます。

例 3: void LogIt(const char* it) { ... }

DLP_Misc_SetLoggingCallback(2, LogIt);

この例では、ロギング詳細レベルとして2が指定されているため、LogIt関数に“レベル1および2”のAPIデバッグ・ロギングが送信されます。

4.7.3 void DLP_Misc_SetLoggingVisibilityLevel (UINT8 logVisibilityLevel)

ロギング詳細レベルを設定します（デフォルトは0 = デバッグ・ロギングなし）。

パラメータ

*loglvl*ロギング詳細レベル、範囲 = {0, 1, 2}。0より大きい値を送信すると、追加のデバッグ・ロギングがイネーブルになります。

値が大きいほど、詳細なデバッグ・ロギングが行われます。

4.7.4 UINT8 DLP_Misc_InitAPI ()

APIを初期化します。

API呼び出しを行う前に、この関数を呼び出す必要があります。

戻り値

成功時は0

4.7.5 UINT8 DLP_Misc_DisableCommunication (const char *ofileName)

リアルタイム通信をディスエーブルにして、指定した出力ファイル名に通信ペイロードを格納するためのデバッグAPIです。

一般に、この関数は、デバッグまたは回帰テスト用にのみ使用されます。

4.7.6 void DLP_Misc_DisableCommunication_FlushOutputFileToDisk ()

通信パケット・データを強制的にディスクへ移動するためのデバッグAPIです。

このAPIは、USBまたはSPI通信がディスエーブルのときにのみ使用できます。

4.7.7 UINT8 DLP_Misc_PassThroughExecute (const char *args[ ])

パススルー・コマンドを実行するためのデバッグAPIです。

設定バッチファイルには、APIの一部だけを呼び出すようにするための“パススルー”コマンドを含めることができます。

パラメータ

*args[ ]*引数文字列の配列。最大要素数 = 16。最後の実要素の後にNULLポインタを配置してください。

戻り値

成功時は0

4.7.8 UINT8 DLP_Misc_ProgramEDID (DMD_t DMD, UINT8 offset, UINT8 numBytes, const char *fileName)

バイナリ・ファイルに格納されたデータをEDIDにプログラミングします。注:EDID 1.3データ構造に基づきます。

部分更新と完全更新の両方に対応します。

パラメータ

DMD DMDタイプ(enumを参照)

*offset*PROM内で新しいデータの更新を開始する位置のオフセット(有効範囲:0～127)

numBytes 上記オフセットから開始してPROMに書き込む、指定ファイル内のバイト数

fileName PROMに書き込むデータを含むバイナリ・ファイル

戻り値

成功時は0

4.7.9 Status ChangeLogLevel (Byte logLevel)

PortAbilityLayerロギング・レベルを設定します。

PortAbilityLayer(SWレイヤ = 2.5)ロギング・レベルを設定します。

低レベル(SWレイヤ = 2.0)関数に対してロギング・レベルを設定するには、4.7.3を参照してください。

パラメータ

logLevel（SWレイヤ = 2.5）ロギング・レベルを設定します。

戻り値

成功時はSTAT_OK

4.7.10 Status InitPortabilityLayer (Byte logLevel, Byte detail, OutputCallback callback)

何らかのAPIを呼び出す前に呼び出す必要のある初期化APIです。

例

```
void Logit(String it) { printf("%s\n", it);}

...

Status rc = InitPortabilityLayer(loglvl+1, loglvl, Logit);
```

パラメータ

logLevel SWレイヤ2.5のロギング・レベル。デバッグ・ロギングの量を増やすには、1より大きい値を設定します。

detail SWレイヤ2.0のロギング・レベル。デバッグ・ロギングの量を増やすには、0より大きい値を設定します。

*callback*ロギング・コールバック関数

戻り値

成功時はSTAT_OK

4.7.11 UINT8 DLP_Misc_GetTotalNumberOfUSBDevicesConnected (UINT8 *oNumDev)

現在PC USBドライバに接続されているUSBデバイスの合計数を取得します。

パラメータ

oNumbDev 接続されているUSBデバイス数の出力値

戻り値

成功時は0

4.7.12 UINT8 DLP_Misc_SetUSBDeviceNumber (UINT8 devNum0b)

以降のAPI呼び出しの送信先となるUSBデバイスを設定します。

入力引数の範囲は {0, 1, ..., N–1} であり、NはPC USBドライバに接続されているUSBデバイスの合計数です。

例えば、USBドライバに4つのデバイスが接続されている場合、このAPIの呼び出しは、{0, 1, 2, 3}の引数を使用してデバイス間の通信切り替えを行います。

デフォルトのデバイス番号として、デバイス0が使用されます。

デバイス番号からPC上のUSBポートへのマッピングは、各マシンに依存し、ユーザは使用しているPCでのマッピングを実験によって確認する必要があります。

パラメータ

devNum0b 0から始まるUSBデバイス番号。デフォルトは0。

戻り値

成功時は0

4.7.13 UINT8 DLP_Misc_GetUSBDeviceNumber (UINT8 *oCurDevNum0b)

API呼び出しの送信先である現在の(0から始まる)USBデバイス番号を取得します。

パラメータ

oCurDevNum0b 現在のUSBデバイス番号の出力値(0から始まる値)

戻り値

成功時は0

4.8 レジスタおよびLUT読み取り/書き込みAPI

モジュール

LUT APIs

関数

- UINT8 DLP_RegIO_InitFromParallelFlashOffset (UINT32 offset, UINT8 reset)
- UINT8 DLP_RegIO_ReadLUT (const char * LUTname)
- UINT8 DLP_RegIO_ReadRegisterField (UINT16 addr, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b, UINT32 * ovalue)
- UINT8 DLP_RegIO_WriteImageOrderLut (UINT8 bpp, const UINT16 8 arrStartImgNum, UINT16 nArrElements)
- UINT8 DLP_RegIO_WriteRegisterField (UINT16 addr, UINT32 value, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b)

レジスタ/LUT APIは、DLPデジタル制御チップの設定に使用されます。

ほとんどの場合、低レベルのレジスタ書き込みAPIは、直接呼び出すことを意図していません。一般には、設定バッチファイルの実行プロセス中に呼び出されます。設定バッチファイルは、LOGIC SWアプリケーションによって作成されます。

APIリリースV1.2.0の時点では、DLPデジタル制御チップの低レベル・レジスタ・マップの仕様が公開される予定はありません。

4.8.1 UINT8 DLP_RegIO_InitFromParallelFlashOffset (UINT32 offset, UINT8 reset)

パラレル・フラッシュ・オフセットから初期化を行います。

パラレル・フラッシュから新しいソリューションをロードします。アドレス・オフセットは、パラレル・フラッシュ・ヘッダに設定されたソリューションのオフセットと一致する必要があります。

パラメータ

offset パラレル・フラッシュ・メモリ・リセットの開始位置へのアドレス・オフセット。初期化前にシステム・リセットを実行する場合は1、リセットをスキップする場合は0。

戻り値

成功時は0

4.8.2 UINT8 DLP_RegIO_ReadLUT (const char * LUTname)

この関数は、LUTの内容を読み取ってログに記録するために呼び出します。

パラメータ「LUT名」を参照 LUT名

戻り値

成功時は0

4.8.3 UINT8 DLP_RegIO_ReadRegisterField (UINT16 addr, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b, UINT32 * ovalue)

DLPコントローラ・レジスタを読み取るためのデバッグAPIです。.

32ビット・レジスタ内の特定の場所から値を読み取ります。

パラメータ

*addr*HWレジスタ・アドレス

full_reg_LSB_bitnum0b 32ビット・レジスタ内のLSBビット位置（0から開始）。範囲 = 0～31

full_reg_MSB_bitnum0b 32ビット・レジスタ内のMSBビット位置（0から開始）。範囲 = 0～31

*ovalue*レジスタ・フィールドからの出力値（範囲はビット数に対応 = MSBbitnum - LSBbitnum+1）

戻り値

成功時は0

例：レジスタ0x1100のビット9:8に2ビット・レジスタ・フィールドが格納されている場合UINT32 oval; UINT8 rc = DLP_RegIO_ReadRegisterField(0x1100, 8,9, oval); (結果 oval = 0x2)

パラメータ

addr 入力 full_reg_LSB_bitnum0b 入力 full_reg_MSB_bitnum0b 入力 ovalue 出力

4.8.4 UINT8 DLP_RegIO_WriteImageOrderLut (UINT8 bpp, const UINT16 8 arrStartImgNum, UINT16 nArrElements)

ImageOrder LUTを設定します。

構造化ライト・モードで使用される、外部メモリに格納された画像データの表示順を定義できます。これにより、画像データを外部メモリに再ロードする必要なしに、画像の表示順を変更できます。

例：外部メモリに3つの8ビット画像パターンが {画像0, 画像1, 画像2} のように格納されているが、ユーザは {2, 1, 0} の順に表示したい場合

UINT16 arrStartImgNum0b[3]={ 2,1,0 };

UINT8 rc = DLP_RegIO_WriteImageOrderLut(8, arrStartImgNum0b, 3);

パラメータ

bpp ピクセルあたりのビット数（対応する各画像ファイルごと）

arrStartImgNum 先頭の画像番号（0から開始）を含む配列

nArrElements 配列の要素数0

戻り値

成功時は0

4.8.5 UINT8 DLP_RegIO_WriteRegisterField (UINT16 addr, UINT32 value, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b)

DLPコントローラ・レジスタに書き込むためのデバッグAPIです。

32ビット・レジスタ内の特定の場所に値を書き込みます。一般に、このAPIは、設定バッチファイルの処理中に呼び出されます。DLPコントローラ・チップのレジスタ・マップは、現在文書化されていません。

パラメータ

addr HWレジスタ・アドレス

value レジスタ・フィールドに書き込む値

full_reg_LSB_bitnum0b 32ビット・レジスタ内のLSBビット位置（0から開始）。範囲 = 0～31

full_reg_MSB_bitnum0b 32ビット・レジスタ内のMSBビット位置（0から開始）。範囲 = 0～31 例：レジスタ0x1100のビット9:8に2ビット・レジスタ・フィールドが格納されている場合 UINT8 rc = DLP_RegIO_WriteRegisterField(0x1100, 0x3, 8,9); 完全なレジスタ書き込みを行うには、LSBビット番号=0かつMSBビット番号=31を設定します。

戻り値

成功時は0

4.9 LUT APIs

関数

- UINT8 DLP_RegIO_BeginLUTdata (const char * LUTname)
- UINT8 DLP_RegIO_EndLUTdata (const char * LUTname)

4.9.1 UINT8 DLP_RegIO_BeginLUTdata (const char * LUTname)

LUTデータの開始を定義するために使用される設定バッチファイル関数です。

設定バッチファイルを処理する際には、レジスタ書き込みによっていずれかのLUTデータ呼び出しを開始する前に、この関数を呼び出します。

パラメータ

*LUTname*を参照 LUT名

戻り値

成功時は0

4.9.2 UINT8 DLP_RegIO_EndLUTdata (const char * LUTname)

LUTデータの終了を定義するために使用される設定バッチファイル関数です。この関数は、（レジスタ書き込みによる）最後のLUTデータ呼び出しの実行後に呼び出します。

パラメータ

LUTname を参照 LUT名

戻り値

成功時は0

4.10 データ・ソースAPI

関数

- UINT8 DLP_Source_SetDataSource (DATA_t source)
- UINT8 DLP_Trigger_SetExternalTriggerEdge (UINT8 edge)

データ・ソースAPIは、入力データ・ソース（ピクセル・データなど）の設定に使用されます。

トリガAPIは、入力されるVSYNCトリガの特性を（DLPデジタル・コントローラの観点から）定義するために使用されます。

4.10.1 UINT8 DLP_Source_SetDataSource (DATA_t source)

入力データ・ソースを設定します。

パラメータ

source 指定されたデータソース

戻り値

成功時は0

4.10.2 UINT8 DLP_Trigger_SetExternalTriggerEdge (UINT8 edge)

トリガ・エッジをプログラミングします。

パラメータ

edge 立ち上がりエッジの場合は0以外、立ち下がりエッジの場合は0を設定

戻り値

成功時は0

4.11 ビデオ・テストパターンジェネレーター

関数

- UINT8 DLP_TPG_SetTestPattern (DMD_t DMD, TPG_Col_t testPattern, TPG_Col_t color, UINT16 patternFreq)

TPG APIは、ビデオ・モード用の内部テストパターンの定義に使用されます。各パターンの空間周波数をAPI入力パラメータで制御できます。

4.11.1 UINT8 DLP_TPG_SetTestPattern (DMD_t DMD, TPG_Col_t testPattern, TPG_Col_t color, UINT16 patternFreq)

入力ソースがテストパターン発生回路ジェネレーターである場合に、表示するテストパターンをプログラミングします。

パラメータ

DMD DMDタイプ (enum)

*testPattern*表示するテストパターン（enumを参照）

color テスト・パターンの色選択（enumを参照）

patternFreq パターンを空間的に繰り返す回数。XGAの場合、有効な値は1、2、4、8、16、32、64、128、256、512です。このパラメータに従って、水平パターンが繰り返されます。垂直DMD解像度は2の累乗でないため、垂直パターンの頻度は、この値に厳密には従いません。

戻り値

成功時は0

4.12 同期API

関数

- UINT8 DLP_Sync_Configure (UINT8 syncNumber, UINT8 polarity, UINT32 delay, UINT32 width)
- UINT8 DLP_Sync_SetEnable (UINT8 syncNumber, UINT8 enableBit)

同期APIは、出力同期信号の調整に使用されます。出力同期を使用して、カメラなどの外部オブジェクトを同期させることができます（例えば、カメラをLightCommanderプロジェクタに“従属”させることが可能）。

4.12.1 UINT8 DLP_Sync_Configure (UINT8 syncNumber, UINT8 polarity, UINT32 delay, UINT32 width)

指定された出力同期を設定します。

パラメータ

syncNumber 設定する出力同期番号（1～3）

polarity 正極性の場合は0以外、負極性の場合は0を設定

delay 出力同期の遅延時間をμs単位で設定

width 出力同期のパルス幅をμs単位で設定

4.12.2 UINT8 DLP_Sync_SetEnable (UINT8 syncNumber, UINT8 enableBit)

指定された出力同期をイネーブル/ディスエーブルにします。

パラメータ

*syncNumber*イネーブル/ディスエーブルにする出力同期番号（1～3）

enableBit イネーブルにするには0以外、ディスエーブルにするには0を設定

戻り値

成功時は0

4.13 Status APIs

関数

- UINT8 DLP_Status_CommunicationStatus ()
- UINT8 DLP_Status_GetBISTdone (UINT8 * state_out)
- UINT8 DLP_Status_GetBISTfail (UINT8 * state_out)
- UINT8 DLP_Status_GetDADcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDADfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetDlpControllerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetDMDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDMDhardwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDparkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDsoftwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetEEPROMfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetFlashProgrammingMode (UINT8 * mode_out)
- UINT8 DLP_Status_GetFlashSeqCompilerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetInitFromParallelFlashFail (UINT8 * state_out)
- UINT8 DLP_Status_GetLEDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetLEDdriverFault (UINT8 * fault_out)
- UINT8 DLP_Status_GetLEDdriverLitState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetLEDdriverTempTimeoutState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetMCUversionString (const char ** ver_out)

- UINT8 DLP_Status_GetOverallLEDdriverTempTimeoutState (UINT8 * state_out)
- UINT8 DLP_Status_GetOverallLEDlampLitState (UINT8 * state_out)
- UINT8 DLP_Status_GetSeqDataBPP (UINT8 * bpp_out)
- UINT8 DLP_Status_GetSeqDataExposure (double * exp_out)
- UINT8 DLP_Status_GetSeqDataFrameRate (double * fr_out)
- UINT8 DLP_Status_GetSeqDataMode (SEQDATA_t * mode_out)
- UINT8 DLP_Status_GetSeqDataNumPattern s (UINT16 * numPatterns_out)
- UINT8 DLP_Status_GetSeqRunState (UINT8 * state_out)
- UINT8 DLP_Status_GetUARTfault (UINT8 * fault_out)

ステータスAPIは、システム・ステータス情報の取得に使用します。

4.13.1 UINT8 DLP_Status_CommunicationStatus ()

USB/SPI通信ステータスを検出します。

戻り値

通信が正常な場合は0、USB/SPI経由で通信できない場合は1 注:v1.2の時点では、USB通信のみがサポートされています。

4.13.2 UINT8 DLP_Status_GetBISTdone (UINT8 * state_out)

外部メモリBIST完了ステータスを取得します。

組み込み自己テスト(BIST)動作が完了したかどうかを確認するためのAPIです。BIST動作は、起動のたびに外部フレーム・メモリに対して実行されます。

パラメータ

state_out 状態フラグ(出力)。BISTが完了している場合は1、それ以外の場合は0。

戻り値

成功時は0

4.13.3 UINT8 DLP_Status_GetBISTfail (UINT8 * state_out)

BISTの障害状態を取得します。

起動のたびに、外部メモリに対してBIST(組み込み自己テスト)動作が実行され、その正常/障害状態が保存されます。このAPIは、このテストの正常/障害状態を返します。

パラメータ

state_out 状態フラグ(出力):BISTが障害状態の場合は1、それ以外の場合は0

戻り値

成功時は0

4.13.4 UINT8 DLP_Status_GetDADcommStatus (UINT8 * status_out)

DLPA200/DAD SPI通信ステータスを取得します。

パラメータ

ステータス・フラグ(出力):正常な場合は0、DAD SPI障害の場合は1

戻り値

成功時は0

4.13.5 UINT8 DLP_Status_GetDADfault (UINT8 * fault_out)

DLPA200（DAD、アナログ・リセット・ドライバとも呼ばれます）の障害をチェックします。.

パラメータ

fault_out 障害フラグ（出力）：DAD障害が検出されると1に設定、それ以外は0

戻り値

成功時は0

4.13.6 UINT8 DLP_Status_GetDlpControllerVersionString (const char ** ver_out)

DLP制御チップのバージョン番号（major.minor.patch形式）を取得します。

パラメータ

ver_out バージョン文字列（出力）：バージョン文字列を格納する静的バッファを指します。

戻り値

成功時は0

4.13.7 UINT8 DLP_Status_GetDMDcommStatus (UINT8 * status_out)

DMD SPI通信ステータスを取得します。

パラメータ status_out ステータス・フラグ（出力）：正常な場合は0、DMD SPI障害の場合は1

戻り値

成功時は0

4.13.8 UINT8 DLP_Status_GetDMDhardwareParkState (UINT8 * state_out)

DMDがハードウェア・スイッチによってパーキングされているかどうかを確認します。

パラメータ state_out 状態フラグ（出力）：HWスイッチがアクティブ化（パーキング）されている場合は1に設定、それ以外は0

戻り値

成功時は0

4.13.9 UINT8 DLP_Status_GetDMDparkState (UINT8 * state_out)

DMDパーク状態を取得します。

パラメータ

state_out 状態フラグ（出力）：パーキングされていない場合は0、パーキングされている場合は1

戻り値

成功時は0

4.13.10 UINT8 DLP_Status_GetDMDsoftwareParkState (UINT8 * state_out)

ソフトウェアによってDMDパークが要求されているかどうかを確認します。

パラメータ

state_out 状態フラグ（出力）：SWによってパークが要求されている場合は1に設定、それ以外は0

戻り値

成功時は0

4.13.11 UINT8 DLP_Status_GetEEPROMfault (UINT8 * fault_out)

EEPROM障害をチェックします。

パラメータ

fault_out 障害フラグ（出力）：EEPROM障害が検出されると1に設定、それ以外は0

戻り値

成功時は0

4.13.12 UINT8 DLP_Status_GetFlashProgrammingMode (UINT8 * mode_out)

フラッシュ・プログラミング・モードを取得します。

パラメータ

mode_out モード（出力）：通常モードの場合は0、フラッシュ・プログラミング・モードの場合は1

戻り値

成功時は0

4.13.13 UINT8 DLP_Status_GetFlashSeqCompilerVersionString (const char ** ver_out)

PWMシーケンス・コンパイラDLLのバージョン番号。

PWMシーケンスの構築に使用されるシーケンス・コンパイラDLLのバージョン番号を取得します。“major.minor.patch”の形式です。

パラメータ

ver_out バージョン文字列（出力）：バージョン文字列を格納する静的バッファを指します。

戻り値

成功時は0

4.13.14 UINT8 DLP_Status_GetInitFromParallelFlashFail (UINT8 * state_out)

パラレル・フラッシュからの初期化の失敗状態を取得します。

パラレル・フラッシュの初期化には3つの状態があります。フラッシュの初期化が正常に完了した場合、APIは0を返します。フラッシュの初期化が試行され、失敗した場合、APIは1を返します。フラッシュの初期化が試行されなかった場合、APIは2を返します。

パラメータ

state_out 状態フラグ（出力）：フラッシュの初期化に成功した場合は0、初期化に失敗した場合は1、初期化が試行されなかった場合は2

戻り値

成功時は0

4.13.15 UINT8 DLP_Status_GetLEDcommStatus (UINT8 * status_out)

LEDドライバのSPI通信ステータスを取得します。

パラメータ

status_out ステータス・フラグ（出力）：正常な場合は0、LEDドライバSPI障害の場合は1

戻り値

成功時は0

4.13.16 UINT8 DLP_Status_GetLEDdriverFault (UINT8 * fault_out)

LEDドライバ障害をチェックします。

パラメータ

fault_out 障害フラグ（出力）：LEDドライバ障害が検出されると1に設定、それ以外は0

戻り値

成功時は0

4.13.17 UINT8 DLP_Status_GetLEDdriverLitState (LED_t LED, UINT8 * state_out)

LEDドライバ点灯状態をLEDタイプごとに取得します。

パラメータ

LED LED発光回路（enumを参照） state_out 状態フラグ（出力）：ドライバがオフの場合は0、オンの場合は1

戻り値

成功時は0

4.13.18 UINT8 DLP_Status_GetLEDdriverTempTimeoutState (LED_t LED, UINT8 * state_out)

LEDドライバの温度タイムアウトをLEDタイプごとに取得します。

パラメータ

LED LED発光回路（enumを参照） state_out タイムアウト・フラグ（出力）：ドライバが温度によってタイムアウトした場合は1、それ以外の場合は0

戻り値

成功時は0

4.13.19 UINT8 DLP_Status_GetMCUversionString (const char ** ver_out)

内部マイクロコントローラSWのバージョン番号。

MCUソフトウェアのバージョン（major.minor.patch形式）を取得します。

パラメータ

ver_out バージョン文字列（出力）：バージョン文字列を格納する静的バッファを指します。

戻り値

成功時は0

4.13.20 UINT8 DLP_Status_GetOverallLEDdriverTempTimeoutState (UINT8 * state_out)

LEDドライバ全体の温度タイムアウト状態を取得します。

パラメータ

state_out 状態フラグ（出力）：LEDドライバがタイムアウトした場合は1、それ以外の場合は0

戻り値

成功時は0

4.13.21 UINT8 DLP_Status_GetOverallLEDlampLitState (UINT8 * state_out)

全体のLED/ランプ点灯状態を取得します。

LEDドライバの全チャネルをチェックして、すべてのLEDが点灯（動作準備完了）しているかどうかを確認します。

パラメータ

state_out 状態フラグ（出力）：点灯していない場合は0、点灯している場合は1

戻り値

成功時は0

4.13.22 UINT8 DLP_Status_GetSeqDataBPP (UINT8 * bpp_out)

シーケンス・データ（BPP：ビット数/ピクセル）を取得します。

現在ロードされているPWMシーケンスの構築に使用されたBPPを記述するメタデータ。BPPは、構造化ライト・モードのみで適用されます。

パラメータ

bpp_out ビット数/ピクセル（出力）

戻り値

成功時は0

4.13.23 UINT8 DLP_Status_GetSeqDataExposure (double * exp_out)

シーケンス・データ（構造化ライトの照射パーセンテージ）を取得します。

現在ロードされているPWMシーケンスの構築に使用された照射時間（μs）を記述するメタデータ。BPPは、構造化ライト・モードのみで適用されます。

パラメータ

exp_out 照射パーセンテージ（出力）

戻り値

成功時は0

4.13.24 UINT8 DLP_Status_GetSeqDataFrameRate (double * fr_out)

シーケンス・データ（フレーム・レート）を取得します。

現在ロードされているPWMシーケンスの構築に使用されたフレーム・レート（Hz）を記述するメタデータ。

パラメータ

fr_out シーケンス・フレーム・レート（出力）

戻り値

成功時は0

4.13.25 UINT8 DLP_Status_GetSeqDataMode (SEQDATA_t * mode_out)

シーケンス・データ・モードを取得します。

現在ロードされているPWMシーケンスを記述するメタデータ。

パラメータ

mode_out 現在のシーケンス・データ・モード

戻り値

成功時は0

4.13.26 UINT8 DLP_Status_GetSeqDataNumPatterns (UINT16 * numPatterns_out)

シーケンス・データ（パターン数/フレーム）を取得します。

現在ロードされているPWMシーケンスの構築に使用されたパターン数/フレームを記述するメタデータ。

パラメータ

numPatterns_out パターン数（出力）

戻り値

成功時は0

4.13.27 UINT8 DLP_Status_GetSeqRunState (UINT8 * state_out)

PWMシーケンス実行状態を取得します。状態フラグ（出力）：停止中の場合は0、正常に実行中の場合は1

戻り値

成功時は0

4.13.28 UINT8 DLP_Status_GetUARTfault (UINT8 * fault_out)

UART障害をチェックします。

パラメータ

fault_out 障害フラグ（出力）：UARTポート障害が検出されると1に設定、それ以外は0

戻り値

成功時は0

4.14 データ型、変数、定数

モジュール

- LUT名

定義

- #define DMD_t_CUR_VERNUM 1
- #define DATA_t_CUR_VERNUM 1
- #define TPG_t_CUR_VERNUM 1
- #define TPG_Col_t_CUR_VERNUM 1
- #define LED_t_CUR_VERNUM 1
- #define SEQDATA_t_CUR_VERNUM 1

型定義

- typedef char Char
- typedef double Double
- typedef signed char Int8
- typedef signed short Int16
- typedef signed int Int32
- typedef unsigned char UInt8
- typedef unsigned short UInt16
- typedef unsigned int UInt32
- typedef const char * String
- typedef UInt8 Byte
- typedef void Void
- typedef int Boolean
- typedef Byte ** IntPtr
- typedef char * StringBuilder

列挙

- enum DMD_t { DMD_XGA = 0 }
- enum DATA_t {
  DVI = 0, EXP,
  TPG ,
  SL_AUTO ,
  SL_EXT3P3 ,
  SL_EXT1P8 ,
  SL_SW }
- enum TPG_t {
  SOLID = 0,
  HORIZ_RAMP ,
  VERT_RAMP ,
  HORIZ_LINES ,
  DIAG_LINES ,
  VERT_LINES ,
  HORIZ_STRIPES ,
  VERT_STRIPES ,
  GRID ,
  CHECKERBOARD }
- enum TPG_Col_t {
  TPG_BLACK = 0,
  TPG_RED ,
  TPG_GREEN ,
  TPG_BLUE ,
  TPG_YELLOW ,
  TPG_CYAN ,
  TPG_MAGENTA ,
  TPG_WHITE }
- enum LED_t {

LED_R = 0,
LED_G = 1,
LED_B = 2,
LED_IR = 3 }

- enum SEQDATA_t {
  SDM_SL = 0,
  SDM_SL_RT = 1,
  SDM_VIDEO = 2,
  SDM_MIXED = 3,
  SDM_OBJ = 4 }
- enum Status {
  STAT_OK ,
  STAT_ERROR }

変数

- static const char arTPGnames [NUM_PATTERNS][32]
- static const char arTPGcolorNames [8][16]
- static char LED_Color [NUM_LEDS][3]

4.14.1 定義の説明

4.14.1.1 #define LED_t_CUR_VERNUM 1

使用可能なLED発光回路。

注: LEDを別の種類のLEDに置き換えることは可能ですが、このenumは、実際には制御信号(リアルタイムLEDイネーブル)を作成するために定義されています。

4.14.2 列挙型の説明

4.14.2.1 enum DATA_t

データ・ソース。

列挙子:

DVI DVIポート EXP 拡張ポート

TPG テスト・パターン発生回路

SL_AUTO 静的フレーム・メモリ、自動生成VSYNC

SL_EXT3P3 静的フレーム・メモリ、3.3V外部VSYNC

SL_EXT1P8 静的フレーム・メモリ、1.8V外部VSYNC

SL_SW 静的フレーム・メモリ、ソフトウェアVSYNC

4.14.3 enum DMD_t

サポートされているDMD。

列挙子

DMD_XGA XGA DMD (1024x768).

4.14.4 enum LED_t

LEDのタイプ

列挙子

LED_R 赤色LED

LED_G 緑色LED

LED_B 青色LED

LED_IR 赤外線LED

4.14.5 enum SEQDATA_t

主要な動作モード。

列挙子

SDM_SL 構造化ライト、非リアルタイム入力

SDM_SL_RT 構造化ライト、リアルタイム入力

SDM_VIDEO ビデオ・モード

SDM_MIXED ビデオ＋構造化ライト

SDM_OBJ オブジェクト・モード

4.14.6 enum Status

ステータス

列挙子

STAT_OK OK.

STAT_ERROR エラー

4.14.7 enum TPG_Col_t

TPGカラー。

参照 Video Test Pattern Generator APIs

列挙子

TPG_BLACK TPG 黒

TPG_RED TPG 赤

TPG_GREEN TPG 緑

TPG_BLUE TPG 青

TPG_YELLOW TPG 黄色

TPG_CYAN TPG シアン

TPG_MAGENTA TPG マゼンタ

TPG_WHITE TPG 白

4.14.8 enum TPG_t

TPGパターン。

参照 Video Test Pattern Generator APIs

列挙子

SOLID TPG 塗りつぶし

HORIZ_RAMP TPG 水平ランプ

VERT_RAMP TPG 垂直ランプ

HORIZ_LINES TPG 水平線

DIAG_LINES TPG 斜線

VERT_LINES TPG 垂直線

HORIZ_STRIPES TPG 水平ストライプ

VERT_STRIPES TPG 垂直ストライプ

GRID TPG グリッド

CHECKERBOARD TPG チェッカーボード

4.15 LUT名

定義

#define REGIO_SEQ_LUT "SEQ_LUT"

#define REGIO_CMT_LUT "CMT_LUT"

#define REGIO_RWC_LUT "RWC_LUT"

#define REGIO_UMCTDM_LUT "UMCTDM_LUT"

4.15.1 #define REGIO_SEQ_LUT "SEQ_LUT"

PMWシーケンスLUT名。

LUT APIs の呼び出しに使用します。

4.15.2 #define REGIO_CMT_LUT "CMT_LUT"

CMT(= デガンマ)LUT名。

LUT APIs の呼び出しに使用します。

4.15.3 #define REGIO_RWC_LUT "RWC_LUT"

RWC LUT名(ほとんど使用されません)。

LUT APIs の呼び出しに使用します。

4.15.4 #define REGIO_UMCTDM_LUT "UMCTDM_LUT"

UMCTDM LUT名(ほとんど使用されません)。

LUT APIs の呼び出しに使用します。

4.16 APIのバージョン番号およびリリース・ノート

定義

#define DLP_API_VERSION_NUM "1.8.0"

- **1.8.0** 2010/9/30 - v1.1リリース
- **1.6.1** 2010/05/21 - 製品版初期リリース
- **1.6.0** 2010/05/11 - ベータ候補

4.17 バッチファイルAPI

関数

- Status RunBatchCommand (String command)
- Status RunBatchFile (String name, Boolean stopOnError)

バッチファイルAPIは、設定バッチファイル内のコマンドを処理するために使用されます。これらのコマンドは現在、DLP LightCommanderキットに付属のLOGICアプリケーションSWによって生成されます。

4.17.1 Status RunBatchCommand (String command)

設定バッチファイルのコマンドを処理します。

このAPIは、バッチファイル・コマンドからAPI関数へのトランスレータとして機能します。

参照 Batchfile Language Specification V1.0 現在サポートされているコマンドの一覧については、

パラメータ

command バッチファイル・コマンド

戻り値

成功時はSTAT_OK;

4.17.2 Status RunBatchFile (String name, Boolean stopOnError)

設定バッチファイルを処理します。

設定バッチファイルは、LOGIC SWアプリケーションによって作成され、DLPコントローラ・チップに対してルックアップ・テーブル（LUT）、レジスタ、および画像データの設定を行います。設定バッチファイルでは、複数のBFコマンドをそれぞれ個別の行に指定できます。 現在サポートされているコマンドの一覧については、Batchfile Language Specification V1.0 を参照。

パラメータ

name ファイル名

stopOnError エラー検出後の動作を制御するブーリアン・フラグ。エラー検出後にバッチファイル処理を停止する場合には1を設定します。0を設定した場合は、戻り値が常にSTAT_OKとなります。

戻り値

成功時はSTAT_OK;

4.18 バッチファイル言語仕様V1.0

モジュール

- Batchfile Status Commands
- Batchfile Display Commands
- Batchfile LED Commands
- Batchfile Image Commands
- Batchfile Misc Commands
- Batchfile LUT/Register Commands
- Batchfile Source/VSYNC Trigger Commands
- Batchfile Internal Test Pattern Generator Commands
- Batchfile Sync Commands

背景

設定バッチファイルは、DLPコントローラ・チップを制御/設定するコマンドを記述したプレーン・テキスト・ファイルです。現在は、LOGICおよびTIによって開発されたアプリケーションSWを使用して、設定バッチファイルを生成できます。このバッチファイルは、同じSWで実行するか、またはRunBatchFile APIを実装する別のSWアプリケーションを通して実行できます。

バッチファイルによって実行できるリアルタイム動作には、以下のものがあります。

- LUTおよび(一部の)レジスタへの書き込み
- システム・ステータスの読み取り
- LEDドライバの制御(例:LEDのイネーブル/ディスエーブル、LED輝度の増加/減少)
- DBI画像ファイルのダウンロード
- EDIDのプログラミング
- DLP表示の制御(例:開始、停止、次の画像、パーク、パーク解除、左右反転、上下反転)
- 入力データ・ポート、内部テスト・パターン、およびVSYNCトリガの選択
- 出力同期信号(= カメラ入力トリガ)の制御

バッチファイルを使用する利点は次のとおりです。

- 一部のLUTは、単純なAPI呼び出しでは生成できず、他のSWを使用して計算する必要があります。いったん計算されれば、LUTは簡単にAPI関数に渡してシステムの設定に使用できます。
- バッチファイルはテキスト・ファイルであるため、変更を加えて新たな設定を追加するのも、または一から作成するのも簡単です。
- 異なるバッチファイルを実行することにより、異なる設定間の切り替えをリアルタイムで簡単に行えます。
- カスタムSWアプリケーションを作成することなく、システムの設定を行えます。

ほとんどの場合、バッチファイル・コマンドと対応するAPI関数の間には1対1の関係が成り立ち、可能な限り、両方に同じ名前が使用されます。例えば、コマンド"$L2.5 DLP_Display_ParkDMD"を含むバッチファイルを実行すると、DLP_Display_ParkDMDが呼び出されます。

上級ユーザであれば、設定バッチファイルをバイナリ形式に"コンパイル"し、パラレル・フラッシュに格納することで、起動時に自動設定を実行できます。パラレル・フラッシュ・レイアウトの仕様は現在のところ提供されていないため、この機能は実際には将来の使用を意図したものです。 参照 Flash Compile APIs 。

バッチファイルの規則

- バッチファイルでは、1つの行に1つのコマンドを記述する必要があります。
- コメントを追加するには"#"を使用します。行の先頭に記述することを推奨します。
- コマンド構文:$L2.5 コマンド [引数1 引数2 ...] [# オプションのコメント]
- "$L2.5"ディレクティブ(SWレイヤ2.5を示す)は、テスト用および将来の機能のために使用されます。
- ほとんどの場合、コマンドはAPI名と厳密に一致します。
- 行内の各トークンは、スペースまたは","によって区切られます。
- 例:"$L2.5 DLP_RegIO_WriteImageOrderLut 8, 0, 1, 2"
- 読み取り動作を実行するコマンドは、結果をstd outに記録します。

バッチファイル・コマンドの分類

Batchfile Status Commands

Batchfile Display Commands

Batchfile LED Commands

4.19 バッチファイル・ステータス・コマンド

- DLP_Status_GetMCUversionString
  - 説明:APIを参照
  - API: DLP_Status_GetMCUversionString
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetDlpControllerVersionString
  - 説明:APIを参照
  - API: DLP_Status_GetDlpControllerVersionString
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetFlashProgrammingMode
  - 説明:APIを参照
  - API: DLP_Status_GetFlashProgrammingMode
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetDADcommStatus
  - 説明:APIを参照
  - API: DLP_Status_GetDADcommStatus
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetDMDcommStatus
  - 説明:APIを参照
  - API: DLP_Status_GetDMDcommStatus
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetLEDcommStatus
  - 説明:APIを参照
  - API: DLP_Status_GetLEDcommStatus
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetBISTdone
  - 説明:APIを参照
  - API: DLP_Status_GetBISTdone

  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetBISTfail
  - 説明:APIを参照
  - API: DLP_Status_GetBISTfail
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetInitFromParallelFlashFail
  - 説明:APIを参照
  - API: DLP_Status_GetInitFromParallelFlashFail
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetOverallLEDlampLitState
  - 説明:APIを参照
  - API: DLP_Status_GetOverallLEDlampLitState
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetOverallLEDdriverTempTimeoutState
  - 説明:APIを参照
  - API: DLP_Status_GetOverallLEDdriverTempTimeoutState
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetSeqDataFrameRate
  - 説明:APIを参照
  - API: DLP_Status_GetSeqDataFrameRate
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetSeqDataExposure
  - 説明:APIを参照
  - API: DLP_Status_GetSeqDataExposure
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetSeqDataNumPatterns
  - 説明:APIを参照
  - API: DLP_Status_GetSeqDataNumPattern
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetSeqDataBPP
  - 説明:APIを参照
  - API: DLP_Status_GetSeqDataBPP
  - 引数:なし

  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_CommunicationStatus
  - 説明:APIを参照
  - API: DLP_Status_CommunicationStatus
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetLEDdriverTempTimeoutState
  - 説明:APIを参照
  - API: DLP_Status_GetLEDdriverTempTimeoutState (LED_t
  - 引数:1. LEDnum: LED_t を参照
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetSeqDataMode
  - 説明:APIを参照
  - API: DLP_Status_GetSeqDataMode
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetLEDdriverLitState
  - 説明:APIを参照
  - API: DLP_Status_GetLEDdriverLitState
  - 引数:1. LEDnum: LED_t を参照
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetFlashSeqCompilerVersionString
  - 説明:APIを参照
  - API: DLP_Status_GetFlashSeqCompilerVersionString
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetDMDparkState
  - 説明:APIを参照
  - API: DLP_Status_GetDMDparkState
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetDMDhardwareParkState
  - 説明:APIを参照
  - API: DLP_Status_GetDMDhardwareParkState
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetDMDsoftwareParkState
  - 説明:APIを参照
  - API: DLP_Status_GetDMDsoftwareParkState
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照

- DLP_Status_GetSeqRunState
  - 説明:APIを参照
  - API: DLP_Status_GetSeqRunState
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetEEPROMfault
  - 説明:APIを参照
  - API: DLP_Status_GetEEPROMfault
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetDADfault
  - 説明:APIを参照
  - API: DLP_Status_GetDADfault
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetLEDdriverFault
  - 説明:APIを参照
  - API: DLP_Status_GetLEDdriverFault
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照
- DLP_Status_GetUARTfault
  - 説明:APIを参照
  - API: DLP_Status_GetUARTfault
  - 引数:なし
  - 出力(ロギングを参照):出力についてはAPIを参照

4.20 バッチファイル表示コマンド

- DLP_Display_DisplayStop
  - 説明:APIを参照
  - API: DLP_Display_DisplayStop
  - 引数:なし
- DLP_Display_ParkDMD
  - 説明:APIを参照
  - API: DLP_Display_ParkDMD
  - 引数:なし
- DLP_Display_UnparkDMD
  - 説明:APIを参照
  - API: DLP_Display_UnparkDMD
  - 引数:なし
- DLP_Display_SetDegammaEnable
  - 説明:APIを参照

  - API: DLP_Display_SetDegammaEnable
  - 引数:1. enableBit:範囲 = { 0,1 }
- DLP_Display_HorizontalFlip
  - 説明:APIを参照
  - API: DLP_Display_HorizontalFlip
  - 引数:1. enableBit:範囲 = { 0,1 }
- DLP_Display_VerticalFlip
  - 説明:APIを参照
  - API: DLP_Display_VerticalFlip
  - 引数:1. enableBit:範囲 = { 0,1 }
  - Example: $L2.5 DLP_Display_VerticalFlip 1
- DLP_Display_DisplayPatternManualStep
  - 説明:APIを参照
  - API: DLP_Display_DisplayPatternManualStep
  - 引数:なし
- DLP_Display_DisplayPatternManualForceFirstPattern
  - 説明:APIを参照
  - API: DLP_Display_DisplayPatternManualForceFirstPattern
  - 引数:なし
- DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses
  - 説明:APIを参照
  - API: DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses
  - 引数:なし

4.21 バッチファイルLEDコマンド

- DLP_LED_LEDdriverEnable
  - 説明:APIを参照
  - API: DLP_LED_LEDdriverEnable
  - 引数:1. enableBit:範囲 = { 0,1 }
- DLP_LED_SetLEDintensity
  - 説明:APIを参照
  - API: DLP_LED_SetLEDintensity
  - 引数:1. LEDnum: LED_tを参照、2. intensityPerCent:範囲 = { 0 - 100.0 }
- DLP_LED_GetLEDintensity
  - 説明:APIを参照
  - API: DLP_LED_GetLEDintensity
  - 引数:1. LEDnum: LED_tを参照 出力(ロギングを参照):intensityPerCent:範囲 = { 0 - 100.0 }
- DLP_LED_SetLEDEnable
  - 説明:APIを参照
  - API: DLP_LED_SetLEDEnable
  - 引数:1. LEDnum: LED_tを参照、2. intensityPerCent:範囲 = { 0 - 100.0 }

4.22 バッチファイル画像コマンド

- WriteExternalImage
  - 説明:指定された書き込み場所を使用して、DBI画像を静的画像バッファにダウンロードします。
  - API: なし
  - 引数:1. filename:DBIファイルのファイル名、2. imgIndex:画像バッファに書き込む際の画像インデックス(0から開始)
  - 例: $L2.5 WriteExternalImage bob.dbi 0 指定されたDBI画像を静的画像バッファ・メモリの画像スロット番号0に書き込みます。8BPP画像の場合、この画像のデータは、画像バッファの最初の8つの場所(ビット平面インデックス0～7)に格納されます。
- DLP_Img_DownloadBitplanePatternFromFlashToExtMem
  - 説明:APIを参照
  - API: DLP_Img_DownloadBitplanePatternFromFlashToExtMem
  - 引数:1. flashOffsetBytes:1つの画像ビット平面のフラッシュ内での位置、2. totalBytes:1つの画像ビット平面のサイズ(XGAの場合は1024*768/8 = 98304)、3. bitPlaneIndex:静的画像バッファ・メモリへの格納に使用する、0から始まるインデックス

4.23 バッチファイル各種コマンド

- DLP_Misc_GetVersionString
  - 説明:APIのバージョン番号を"major.minor.patch"形式で返します。
  - API: DLP_Misc_GetVersionString
  - 引数:なし
- RunBatchFile
  - 説明:このコマンドは、バッチファイル内から別のバッチファイルを呼び出すために使用します。
  - API: なし
  - 引数:1. filename:実行するバッチファイルのファイル名、2. stopOnError:エラー検出時に実行を停止する場合は1を設定、それ以外の場合は0を設定
  - 例: $L2.5 RunBatchFile img_order_lut.bf 0 指定されたバッチファイルを実行します。バッチファイル内から別のバッチファイルを呼び出します。
- RunBatchCommand
  - 説明:バッチファイル・コマンドを実行します。このコマンドは、アプリケーションが(ディスク上の物理ファイルではなく)メモリ内に格納されたBFコマンドを実行している場合にだけ意味を持ちます。
  - API: なし
  - 引数:1. bfCmd:バッチファイル・コマンド
  - 例: $L2.5 RunBatchCommand WriteExternalImage bob.dbi 0
- ExecutePassthroughDLPAPI
  - 説明:API関数を呼び出す代替方法です。主に、初期のSW開発中に、エクスポートされていないDLL関数を呼び出すために使用します。
  - API: なし
  - 引数:1. APIname:API関数名、2. 0個以上の追加引数(各APIで異なる)
- ChangeLogLevel
  - 説明:出力ロギング・レベルを変更します。値が大きいほど、デバッグ・ロギングが詳細になります。
  - API: なし

  - 引数:1. logLevel:1から始まるロギング・レベル、デフォルト = 1、追加のデバッグ・ロギングを行うには値を増加
- DLP_Misc_EnableCommunication
  - 説明:APIを参照
  - API: DLP_Misc_EnableCommunication
  - 引数:なし
- DLP_Misc_DisableCommunication
  - 説明:APIを参照
  - API: DLP_Misc_DisableCommunication
  - 引数:1. 出力ファイル名
- DLP_Misc_DisableCommunication_FlushOutputFileToDisk
  - 説明:APIを参照
  - API: DLP_Misc_DisableCommunication_FlushOutputFileToDisk
  - 引数:なし
- DLP_Misc_ProgramEDID
  - 説明:APIを参照
  - API: DLP_Misc_ProgramEDID
  - 引数:1. DMDnum: DMD_tを参照、2. byteOffset:スキップ・バイト数、3. numBytes:プログラミングするバイト数、4. filename:入力ファイル名
- DLP_Misc_GetTotalNumberOfUSBDevicesConnected
  - 説明:現在PC USBドライバに接続されているUSBデバイスの合計数を返します。
  - API: DLP_Misc_GetTotalNumberOfUSBDevicesConnected
  - 引数:なし
- DLP_Misc_SetUSBDeviceNumber
  - 説明:APIを参照
  - API: DLP_Misc_SetUSBDeviceNumber
  - 引数:
    - devNum0b:0から始まるUSBデバイス番号。デフォルトは0。
- DLP_Misc_GetUSBDeviceNumber
  - 説明:API呼び出しの送信先である現在の(0から始まる)USBデバイス番号を返します。
  - API: DLP_Misc_GetUSBDeviceNumber
  - 引数:なし

4.24 バッチファイルLUT/レジスタ・コマンド

- ReadReg
  - 説明:32ビット・レジスタ値全体を読み出すための低レベル・デバッグAPI。DLPコントローラ・チップのレジスタ仕様は、公開文書ではありません。
  - API: DLP_RegIO_ReadRegisterField
  - 引数:1. addr:16進または10進のレジスタ・アドレス。現在、レジスタ・アドレスは非公開です。
  - 出力(ロギングを参照):出力についてはAPIを参照
- WriteReg

  - 説明:32ビット・レジスタ値を書き込むための低レベル・デバッグAPI。DLPコントローラ・チップのレジスタ仕様は、公開文書ではありません。
  - API: DLP_RegIO_WriteRegisterField
  - 引数:1. addr:16進または10進のレジスタ・アドレス。現在、レジスタ・アドレスは非公開です。2. val32:32ビットの16進または10進レジスタ値
- DLP_RegIO_WriteImageOrderLut
  - 説明:APIを参照
  - API: DLP_RegIO_WriteImageOrderLut
  - 引数:1. BPP:ダウンロードした画像のビット数/ピクセル、2. arrImgNumbers:画像バッファに格納されているダウンロード済み画像ファイルの再生順に対応する、1つ以上のインデックス番号のリスト
  - 例: $L2.5 DLP_RegIO_WriteImageOrderLut 8, 2, 0, 1 3つのダウンロードされた画像の表示順を2, 0, 1に変更します。画像は8BPPです。
- DLP_RegIO_BeginLUTdata
  - 説明:APIを参照
  - API: DLP_RegIO_BeginLUTdata
  - 引数:1. LUTname: LUT名 名を参照
- DLP_RegIO_EndLUTdata
  - 説明:APIを参照
  - API: DLP_RegIO_EndLUTdata
  - 引数:1. LUTname: LUT名 名を参照
- DLP_RegIO_InitFromParallelFlashOffset
  - 説明:APIを参照
  - API: DLP_RegIO_InitFromParallelFlashOffset
  - 引数:1. offset:バイト・オフセット=読み取り開始アドレス。ソリューションの開始オフセットに対応する必要があります。2. resetFlag:リセット・フラグ。システム・リセット(起動時の初期化を再現)を要求する場合は1に設定

4.25 バッチファイル・ソース/VSYNCトリガ・コマンド

- DLP_Source_SetDataSource
  - 説明:APIを参照
  - API: DLP_Source_SetDataSource
  - 引数:1. source: DATA_t を参照
- DLP_Trigger_SetExternalTriggerEdge
  - 説明:APIを参照
  - API: DLP_Trigger_SetExternalTriggerEdge
  - 引数:1. edge:APIを参照

4.26 バッチファイル内部テスト・パターン生成コマンド

- DLP_TPG_SetTestPattern
  - 説明:APIを参照
  - API: DLP_TPG_SetTestPattern
  - 引数:1. DMDnum: DMD_tを参照、2. patnum:テスト・パターン番号、TPG_tを参照、3. clrnum:テスト・パ

ターン・カラー、TPG_Col_tを参照、4. patFreq：空間パターンの頻度、有効な値についてはAPIを参照

4.27 バッチファイル同期コマンド

- DLP_Sync_SetEnable
  - 説明：APIを参照
  - API: DLP_Sync_SetEnable
  - 引数：1. syncNum：1から始まる同期番号、範囲 = { 1,2,3 }、2. enableBit：範囲 = { 0,1 }
- DLP_Sync_Configure
  - 説明：APIを参照
  - API: DLP_Sync_Configure
  - 引数：1. syncnum：1から始まる同期番号、範囲 = { 1,2,3 }、2. polarity：APIを参照、3. delay：APIを参照、4. width：APIを参照

5 ファイルの説明

5.1 dlp_types.h ファイルのリファレンス

定義

- #define DMD_t_CUR_VERNUM 1
- #define DATA_t_CUR_VERNUM 1
- #define TPG_t_CUR_VERNUM 1
- #define TPG_Col_t_CUR_VERNUM 1
- #define LED_t_CUR_VERNUM 1
- #define SEQDATA_t_CUR_VERNUM 1
- #define REGIO_SEQ_LUT "SEQ_LUT"
- #define REGIO_CMT_LUT "CMT_LUT"
- #define REGIO_RWC_LUT "RWC_LUT"
- #define REGIO_UMCTDM_LUT "UMCTDM_LUT"

列挙

- enum DMD_t {
  DMD_XGA = 0 }
- enum DATA_t {
  DVI = 0,
  EXP ,
  TPG ,
  SL_AUTO ,
  SL_EXT3P3 ,
  SL_EXT1P8 ,
  SL_SW }
- enum TPG_t {
  SOLID = 0,
  HORIZ_RAMP ,
  VERT_RAMP ,
  HORIZ_LINES ,

DIAG_LINES ,
VERT_LINES ,
HORIZ_STRIPES ,
VERT_STRIPES ,
GRID ,
CHECKERBOARD }

- enum TPG_Col_t {
  TPG_BLACK = 0,
  TPG_RED ,
  TPG_GREEN ,
  TPG_BLUE ,
  TPG_YELLOW ,
  TPG_CYAN ,
  TPG_MAGENTA ,
  TPG_WHITE }
- enum LED_t {
  LED_R = 0,
  LED_G = 1,
  LED_B = 2,
  LED_IR = 3 }
- enum SEQDATA_t {
  SDM_SL = 0 ,
  SDM_SL_RT = 1 ,
  SDM_VIDEO = 2,
  SDM_MIXED = 3,
  SDM_OBJ = 4 }

変数

- static const char arTPGnames [NUM_PATTERNS][32]
- static const char arTPGcolorNames [8][16]
- static char LED_Color [NUM_LEDS][3]

5.2 DlpApi.cファイルのリファレンス

#include "DlpAPI.h"

#include <stdio.h>

#include "DLP_types.h"

#include "stdarg.h"

#include "string.h"

#include "stdlib.h"

#include "..\DlpAPIlib\DLP_types.h"

定義

- #define DLP_API_VERSION_NUM 1.6.1

関数

- UINT8 DLP_Display_DisplayPatternManualForceFirstPattern ()
- UINT8 DLP_Display_DisplayPatternManualStep ()
- UINT8 DLP_Display_DisplayPatternAutoStepForSinglePass ()
- UINT8 DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses ()
- UINT8 DLP_Display_DisplayStop ()
- UINT8 DLP_Display_ParkDMD ()
- UINT8 DLP_Display_UnparkDMD ()
- UINT8 DLP_Display_SetDegammaEnable (UINT8 enableBit)
- UINT8 DLP_Display_HorizontalFlip (UINT8 enableBit)
- UINT8 DLP_Display_VerticalFlip (UINT8 enableBit)
- void DLP_FlashCompile_SetCommPacketCallback (void(*pf)(UINT8 *packetBuffer, UINT16 nBufBytes))
- void DLP_FlashCompile_FlushCommPacketBuffer ()
- void DLP_FlashCompile_SetCompileMode (UINT8 *enableBit*)
- UINT8 DLP_FlashCompile_GetCompileMode ()
- UINT8 DLP_FlashProgram_ProgramSerialFlash (UINT32 *nSkipBytesInFlash* , UINT8 *pBuf, UINT32 *nBytesInBuf* , UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 * *out* CRC)
- UINT8 DLP_FlashProgram_ProgramParallelFlash (UINT32 nSkipBytesInFlash, UINT8 *pBuf, UINT32 nBytesInBuf, UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 *outCRC)
- UINT8 DLP_FlashProgram_EraseParallelFlash (UINT32 *nSkipBytesInFlash* , UINT32 *nBytesToErase*)
- UINT8 DLP_Img_DownloadBitplanePatternToExtMem (const UINT8 *pBuf, UINT32 tot_bytes, UINT16 bpnum0b)
- UINT8 DLP_Img_DownloadBitplanePatternFromFlashToExtMem (UINT32 flash_byte_offset, UINT32 tot_bytes, UINT16 bpnum0b)
- UINT8 DLP_LED_SetLEDintensity (LED_t LED, double intensity_perCent)
- UINT8 DLP_LED_GetLEDintensity (LED_t LED, double *intensityPerCent)
- void DLP_LED_LEDdriverEnable (UINT8 enable)
- void DLP_LED_GetLEDdriverTimeout (UINT8 *timedOut)
- UINT8 DLP_LED_SetLEDEnable (LED_t LED, UINT8 enableBit)
- const char * DLP_Misc_GetVersionString ()
- void DLP_Misc_SetLoggingCallback (UINT8 logVisibilityLevel, void(*pf)(const char *))
- void DLP_Misc_SetLoggingVisibilityLevel (UINT8 logVisibilityLevel)
- UINT8 DLP_Misc_InitAPI ()
- UINT8 DLP_Misc_EnableCommunication ()
- UINT8 DLP_Misc_DisableCommunication (const char *ofileName)
- void DLP_Misc_DisableCommunication_FlushOutputFileToDisk ()
- UINT8 DLP_Misc_PassThroughExecute (const char *args[ ])
- UINT8 DLP_Misc_ProgramEDID (DMD_t DMD, UINT8 offset, UINT8 numBytes, const char *fileName)
- UINT8 DLP_RegIO_InitFromParallelFlashOffset (UINT32 offset, UINT8 reset)
- UINT8 DLP_RegIO_ReadLUT (const char * LUTname)
- UINT8 DLP_RegIO_ReadRegisterField (UINT16 addr, UINT8 full_reg_LSB_bitnum0b, UINT8

full_reg_MSB_bitnum0b, UINT32 * ovalue)

- UINT8 DLP_RegIO_WriteImageOrderLut (UINT8 bpp, const UINT16 8 arrStartImgNum, UINT16 nArrElements)
- UINT8 DLP_RegIO_WriteRegisterField (UINT16 addr, UINT32 value, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b)
- UINT8 DLP_RegIO_BeginLUTdata (const char * LUTname)
- UINT8 DLP_RegIO_EndLUTdata (const char * LUTname)
- UINT8 DLP_Source_SetDataSource (DATA_t source)
- UINT8 DLP_Trigger_SetExternalTriggerEdge (UINT8 edge)
- UINT8 DLP_TPG_SetTestPattern (DMD_t DMD, TPG_Col_t testPattern, TPG_Col_t color, UINT16 patternFreq)
- UINT8 DLP_Sync_Configure (UINT8 syncNumber, UINT8 polarity, UINT32 delay, UINT32 width)
- UINT8 DLP_Sync_SetEnable (UINT8 syncNumber, UINT8 enableBit)
- UINT8 DLP_Status_CommunicationStatus ()
- UINT8 DLP_Status_GetBISTdone (UINT8 * state_out)
- UINT8 DLP_Status_GetBISTfail (UINT8 * state_out)
- UINT8 DLP_Status_GetDADcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDADfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetDlpControllerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetDMDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDMDhardwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDparkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDsoftwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetEEPROMfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetFlashProgrammingMode (UINT8 * mode_out)
- UINT8 DLP_Status_GetFlashSeqCompilerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetInitFromParallelFlashFail (UINT8 * state_out)
- UINT8 DLP_Status_GetLEDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetLEDdriverFault (UINT8 * fault_out)
- UINT8 DLP_Status_GetLEDdriverLitState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetLEDdriverTempTimeoutState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetMCUversionString (const char ** ver_out)
- UINT8 DLP_Status_GetOverallLEDdriverTempTimeoutState (UINT8 * state_out)
- UINT8 DLP_Status_GetOverallLEDlampLitState (UINT8 * state_out)
- UINT8 DLP_Status_GetSeqDataBPP (UINT8 * bpp_out)
- UINT8 (double * exp_out)
- UINT8 DLP_Status_GetSeqDataExposure DLP_Status_GetSeqDataFrameRate (double * fr_out)
- UINT8 DLP_Status_GetSeqDataMode (SEQDATA_t * mode_out)
- UINT8 DLP_Status_GetSeqDataNumPattern s (UINT16 * numPatterns_out)
- UINT8 DLP_Status_GetSeqRunState (UINT8 * state_out)
- UINT8 DLP_Status_GetUARTfault (UINT8 * fault_out)

5.3 DlpApi.h ファイルのリファレンス

#include <stdio.h>

#include "DLP_types.h"

関数

- UINT8 DLP_Display_DisplayPatternManualForceFirstPattern ()
- UINT8 DLP_Display_DisplayPatternManualStep ()
- UINT8 DLP_Display_DisplayPatternAutoStepForSinglePass ()
- UINT8 DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses ()
- UINT8 DLP_Display_DisplayStop ()
- UINT8 DLP_Display_ParkDMD ()
- UINT8 DLP_Display_UnparkDMD ()
- UINT8 DLP_Display_SetDegammaEnable (UINT8 enableBit)
- UINT8 DLP_Display_HorizontalFlip (UINT8 enableBit)
- UINT8 DLP_Display_VerticalFlip (UINT8 enableBit)
- void DLP_FlashCompile_SetCommPacketCallback (void(*pf)(UINT8 *packetBuffer, UINT16 nBufBytes))
- void DLP_FlashCompile_FlushCommPacketBuffer ()
- void DLP_FlashCompile_SetCompileMode (UINT8 *enableBit*)
- UINT8 DLP_FlashCompile_GetCompileMode ()
- UINT8 DLP_FlashProgram_ProgramSerialFlash (UINT32 *nSkipBytesInFlash* , UINT8 *pBuf, UINT32 *nBytesInBuf* , UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 * *out* CRC)
- UINT8 DLP_FlashProgram_ProgramParallelFlash (UINT32 nSkipBytesInFlash, UINT8 *pBuf, UINT32 nBytesInBuf, UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 *outCRC)
- UINT8 DLP_FlashProgram_EraseParallelFlash (UINT32 *nSkipBytesInFlash* , UINT32 *nBytesToErase*)
- UINT8 DLP_Img_DownloadBitplanePatternToExtMem (const UINT8 *pBuf, UINT32 tot_bytes, UINT16 bpnum0b)
- UINT8 DLP_Img_DownloadBitplanePatternFromFlashToExtMem (UINT32 flash_byte_offset, UINT32 tot_bytes, UINT16 bpnum0b)
- UINT8 DLP_LED_SetLEDintensity (LED_t LED, double intensity_perCent)
- UINT8 DLP_LED_GetLEDintensity (LED_t LED, double *intensityPerCent)
- void DLP_LED_LEDdriverEnable (UINT8 enable)
- void DLP_LED_GetLEDdriverTimeout (UINT8 *timedOut)
- UINT8 DLP_LED_SetLEDEnable (LED_t LED, UINT8 enableBit)
- const char * DLP_Misc_GetVersionString ()
- void DLP_Misc_SetLoggingCallback (UINT8 logVisibilityLevel, void(*pf)(const char *))
- void DLP_Misc_SetLoggingVisibilityLevel (UINT8 logVisibilityLevel)
- UINT8 DLP_Misc_InitAPI ()
- UINT8 DLP_Misc_EnableCommunication ()
- UINT8 DLP_Misc_DisableCommunication (const char *ofileName)
- void DLP_Misc_DisableCommunication_FlushOutputFileToDisk ()
- UINT8 DLP_Misc_PassThroughExecute (const char *args[ ])

- UINT8 DLP_Misc_ProgramEDID (DMD_t DMD, UINT8 offset, UINT8 numBytes, const char *fileName)
- UINT8 DLP_RegIO_InitFromParallelFlashOffset (UINT32 offset, UINT8 reset)
- UINT8 DLP_RegIO_ReadLUT (const char * LUTname)
- UINT8 DLP_RegIO_ReadRegisterField (UINT16 addr, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b, UINT32 * ovalue)
- UINT8 DLP_RegIO_WriteImageOrderLut (UINT8 bpp, const UINT16 8 arrStartImgNum, UINT16 nArrElements)
- UINT8 DLP_RegIO_WriteRegisterField (UINT16 addr, UINT32 value, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b)
- UINT8 DLP_RegIO_BeginLUTdata (const char * LUTname)
- UINT8 DLP_RegIO_EndLUTdata (const char * LUTname)
- UINT8 DLP_Source_SetDataSource (DATA_t source)
- UINT8 DLP_Trigger_SetExternalTriggerEdge (UINT8 edge)
- UINT8 DLP_TPG_SetTestPattern (DMD_t DMD, TPG_Col_t testPattern, TPG_Col_t color, UINT16 patternFreq)
- UINT8 DLP_Sync_Configure (UINT8 syncNumber, UINT8 polarity, UINT32 delay, UINT32 width)
- UINT8 DLP_Sync_SetEnable (UINT8 syncNumber, UINT8 enableBit)
- UINT8 DLP_Status_CommunicationStatus ()
- UINT8 DLP_Status_GetBISTdone (UINT8 * state_out)
- UINT8 DLP_Status_GetBISTfail (UINT8 * state_out)
- UINT8 DLP_Status_GetDADcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDADfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetDlpControllerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetDMDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDMDhardwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDparkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDsoftwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetEEPROMfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetFlashProgrammingMode (UINT8 * mode_out)
- UINT8 DLP_Status_GetFlashSeqCompilerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetInitFromParallelFlashFail (UINT8 * state_out)
- UINT8 DLP_Status_GetLEDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetLEDdriverFault (UINT8 * fault_out)
- UINT8 DLP_Status_GetLEDdriverLitState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetLEDdriverTempTimeoutState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetMCUversionString (const char ** ver_out)
- UINT8 DLP_Status_GetOverallLEDdriverTempTimeoutState (UINT8 * state_out)
- UINT8 DLP_Status_GetOverallLEDlampLitState (UINT8 * state_out)
- UINT8 DLP_Status_GetSeqDataBPP (UINT8 * bpp_out)
- UINT8 (double * exp_out)
- UINT8 DLP_Status_GetSeqDataExposure DLP_Status_GetSeqDataFrameRate (double * fr_out)

- UINT8 DLP_Status_GetSeqDataMode (SEQDATA_t * mode_out)
- UINT8 DLP_Status_GetSeqDataNumPattern s (UINT16 * numPatterns_out)
- UINT8 DLP_Status_GetSeqRunState (UINT8 * state_out)
- UINT8 DLP_Status_GetUARTfault (UINT8 * fault_out)

5.4 dummy_portabilityLayer.h ファイルのリファレンス

型定義

- typedef char Char
- typedef double Double
- typedef signed char Int8
- typedef signed short Int16
- typedef signed int Int32
- typedef unsigned char UInt8
- typedef unsigned short UInt16
- typedef unsigned int UInt32
- typedef const char * String
- typedef UInt8 Byte
- typedef void Void
- typedef int Boolean
- typedef Byte ** IntPtr
- typedef char * StringBuilder

列挙

- enum Status {
  STAT_OK ,
  STAT_ERROR }

関数

- Status RunBatchFile (String name, Boolean stopOnError)
- Status RunBatchCommand (String command)
- Status ChangeLogLevel (Byte logLevel)
- Status InitPortabilityLayer (Byte logLevel, Byte detail, OutputCallback callback)
- Status WriteExternalImage (String name, Byte imageIndex)
- IntPtr GetAllBitPlanes (String name, UInt32 bpp, UInt32 *bitPlaneSize)
- Status destroyBitPlanes (IntPtr bitPlanes, UInt32 bpp)

ご注意

日本テキサス・インスツルメンツ株式会社（以下TIJといいます）及びTexas Instruments Incorporated（TIJの親会社、以下TIJないしTexas Instruments Incorporatedを総称してTIといいます）は、その製品及びサービスを任意に修正し、改善、改良、その他の変更をし、もしくは製品の製造中止またはサービスの提供を中止する権利を留保します。従いまして、お客様は、発注される前に、関連する最新の情報を取得して頂き、その情報が現在有効かつ完全なものであるかどうかご確認下さい。全ての製品は、お客様とTIJとの間に取引契約が締結されている場合は、当該契約条件に基づき、また当該取引契約が締結されていない場合は、ご注文の受諾の際に提示されるTIJの標準販売契約約款に従って販売されます。

TIは、そのハードウェア製品が、TIの標準保証条件に従い販売時の仕様に対応した性能を有していること、またはお客様とTIJとの間で合意された保証条件に従い合意された仕様に対応した性能を有していることを保証します。検査およびその他の品質管理技法は、TIが当該保証を支援するのに必要とみなす範囲で行なわれております。各デバイスの全てのパラメーターに関する固有の検査は、政府がそれ等の実行を義務づけている場合を除き、必ずしも行なわれておりません。

TIは、製品のアプリケーションに関する支援もしくはお客様の製品の設計について責任を負うことはありません。TI製部品を使用しているお客様の製品及びそのアプリケーションについての責任はお客様にあります。TI製部品を使用したお客様の製品及びアプリケーションについて想定されうる危険を最小のものとするため、適切な設計上および操作上の安全対策は、必ずお客様にてお取り下さい。

TIは、TIの製品もしくはサービスが使用されている組み合せ、機械装置、もしくは方法に関連しているTIの特許権、著作権、回路配置利用権、その他のTIの知的財産権に基づいて何らかのライセンスを許諾するということは明示的にも黙示的にも保証も表明もしておりません。TIが第三者の製品もしくはサービスについて情報を提供することは、TIが当該製品もしくはサービスを使用することについてライセンスを与えるとか、保証もしくは是認するということを意味しません。そのような情報を使用するには第三者の特許その他の知的財産権に基づき当該第三者からライセンスを得なければならない場合もあり、またTIの特許その他の知的財産権に基づきTI からライセンスを得て頂かなければならない場合もあります。

TIのデータ・ブックもしくはデータ・シートの中にある情報を複製することは、その情報に一切の変更を加えること無く、かつその情報と結び付られた全ての保証、条件、制限及び通知と共に複製がなされる限りにおいて許されるものとします。当該情報に変更を加えて複製することは不公正で誤認を生じさせる行為です。TIは、そのような変更された情報や複製については何の義務も責任も負いません。

TIの製品もしくはサービスについてTIにより示された数値、特性、条件その他のパラメーターと異なる、あるいは、それを超えてなされた説明で当該TI製品もしくはサービスを再販売することは、当該TI製品もしくはサービスに対する全ての明示的保証、及び何らかの黙示的保証を無効にし、かつ不公正で誤認を生じさせる行為です。TIは、そのような説明については何の義務も責任もありません。

TIは、TIの製品が、安全でないことが致命的となる用途ないしアプリケーション（例えば、生命維持装置のように、TI製品に不良があった場合に、その不良により相当な確率で死傷等の重篤な事故が発生するようなもの）に使用されることを認めておりません。但し、お客様とTIの双方の権限有る役員が書面でそのような使用について明確に合意した場合は除きます。たとえTIがアプリケーションに関連した情報やサポートを提供したとしても、お客様は、そのようなアプリケーションの安全面及び規制面から見た諸問題を解決するために必要とされる専門的知識及び技術を持ち、かつ、お客様の製品について、またTI製品をそのような安全でないことが致命的となる用途に使用することについて、お客様が全ての法的責任、規制を遵守する責任、及び安全に関する要求事項を満足させる責任を負っていることを認め、かつそのことに同意します。さらに、もし万一、TIの製品がそのような安全でないことが致命的となる用途に使用されたことによって損害が発生し、TIないしその代表者がその損害を賠償した場合は、お客様がTIないしその代表者にその全額の補償をするものとします。

TI製品は、軍事的用途もしくは宇宙航空アプリケーションないし軍事的環境、航空宇宙環境にて使用されるようには設計もされていませんし、使用されることを意図されておりません。但し、当該TI製品が、軍需対応グレード品、若しくは「強化プラスティック」製品としてTIが特別に指定した製品である場合は除きます。TIが軍需対応グレード品として指定した製品のみが軍需品の仕様書に合致いたします。お客様は、TIが軍需対応グレード品として指定していない製品を、軍事的用途もしくは軍事的環境下で使用することは、もっぱらお客様の危険負担においてなされるということ、及び、お客様がもっぱら責任をもって、そのような使用に関して必要とされる全ての法的要求事項及び規制上の要求事項を満足させなければならないことを認め、かつ同意します。

TI製品は、自動車用アプリケーションないし自動車の環境において使用されるようには設計されていませんし、また使用されることを意図されておりません。但し、TIがISO/TS 16949の要求事項を満たしていると特別に指定したTI製品は除きます。お客様は、お客様が当該TI指定品以外のTI製品を自動車用アプリケーションに使用しても、TIは当該要求事項を満たしていなかったことについて、いかなる責任も負わないことを認め、かつ同意します。

弊社半導体製品の取り扱い・保管について

半導体製品は、取り扱い、保管・輸送環境、基板実装条件によっては、お客様での実装前後に破壊/劣化、または故障を起こすことがあります。

弊社半導体製品のお取り扱い、ご使用にあたっては下記の点を遵守して下さい。

1. 静電気
   - 素手で半導体製品単体を触らないこと。どうしても触る必要がある場合は、リストストラップ等で人体からアースをとり、導電性手袋等をして取り扱うこと。
   - 弊社出荷梱包単位（外装から取り出された内装及び個装）又は製品単品で取り扱いを行う場合は、接地された導電性のテーブル上で（導電性マットにアースをとったもの等）、アースをした作業者が行うこと。また、コンテナ等も、導電性のものを使うこと。
   - マウンタやはんだ付け設備等、半導体の実装に関わる全ての装置類は、静電気の帯電を防止する措置を施すこと。
   - 前記のリストストラップ・導電性手袋・テーブル表面及び実装装置類の接地等の静電気帯電防止措置は、常に管理されその機能が確認されていること。
2. 温・湿度環境
   - 温度：0～40℃、相対湿度：40～85%で保管・輸送及び取り扱いを行うこと。（但し、結露しないこと。）
   - 直射日光があたる状態で保管・輸送しないこと。
3. 防湿梱包
   - 防湿梱包品は、開封後は個別推奨保管環境及び期間に従い基板実装すること。
4. 機械的衝撃
   - 梱包品（外装、内装、個装）及び製品単品を落下させたり、衝撃を与えないこと。
5. 熱衝撃
   - はんだ付け時は、最低限260℃以上の高温状態に、10秒以上さらさないこと。（個別推奨条件がある時はそれに従うこと。）
6. 汚染
   - はんだ付け性を損なう、又はアルミ配線腐食の原因となるような汚染物質（硫黄、塩素等ハロゲン）のある環境で保管・輸送しないこと。
   - はんだ付け後は十分にフラックスの洗浄を行うこと。（不純物含有率が一定以下に保証された無洗浄タイプのフラックスは除く。）

以上

2001.11