Programmer's Guide
DLPU005C–October 2011–Revised March 2018

DLPC200 SPI Slave Interface Specification

1 Purpose

The purpose of this document is to provide details on programming DLPC200 chipset over its SPI Slave port. The document contains list of commands along with their detailed specifications. The commands listed in this document provide similar functionality as compared to the DLPC200 API reference manual that is used in controlling DLPC200 over USB port. It must be noted that not all the interfaces available in the API reference manual are supported on SPI interface. This is because the intended use on SPI slave port is to allow real time control functionality.

2 Assumptions

This document does not contain details on SPI Slave hardware interface specifications. Refer to the SPI Slave interface section of DLPC200 datasheet for the SPI slave hardware interface specification.

3 Reference Links

DLPC200 DLP Digital Controller for the DLP5500 DMD (DLPS014)

DLP LightCommander API (DLPA024)

DLPC200 API Programmer's Guide(DLPA014)

4 Overview

The SPI slave bus on DLPC200 is a standard synchronous four-wire serial data link that operates in full duplex mode. There is an additional signal available to read DLPC200 busy status.

The SPI bus specifies four logic signals plus the DLPC200 BUSY status.

- SLAVE_SPI_CLK Serial Clock (Output from master)
- SLAVE_SPI_MOSI Master Output, Slave Input (Output from master)
- SLAVE_SPI_MISO Master Input, Slave Output (Output from DLPC200)
- SLAVE_SPI_CS Slave Select (Active LOW; output from master) Additional signal
- SLAVE_SPI_ACK Slave is busy (Output from DLPC200)

The data transmission over SPI bus happens in byte-by-byte mode. To begin communication the master should first check for SLAVE_SPI_ACK signal status LOW. In the next step the master pulls the SLAVE_SPI_CS signal to LOW. After this the master can proceed with clocking data at a frequency less than or equal to 5MHz. During each SPI clock cycle, a full duplex data transmission occurs; that is the master sends a bit on the SLAVE_SPI_MOSI line; the slave reads it from that same line and the slave sends a bit on the SLAVE_SPI_MISO line; the master reads it from that same line. The master must always check SLAVE_SPI_ACK LOW before transmitting or receiving data byte, see Figure 1.

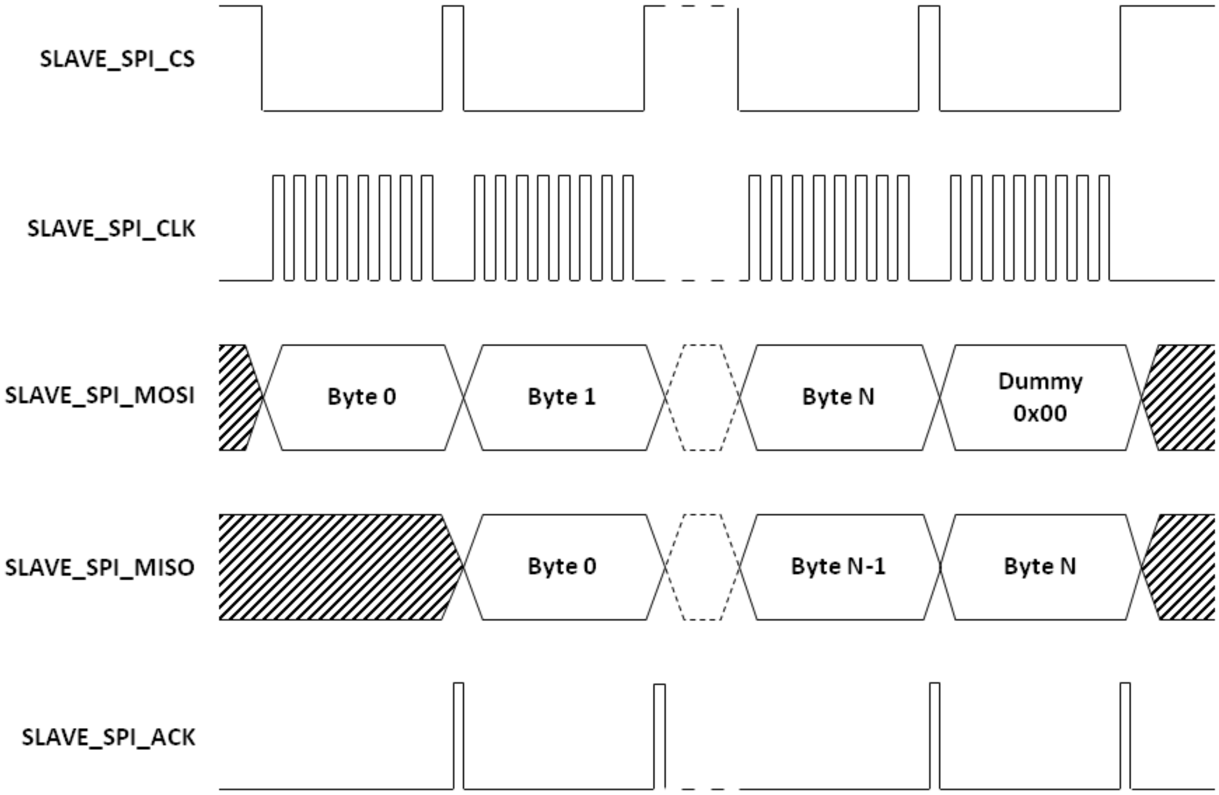


Figure 1. SPI Byte Transfer

At a high level the commands supported over SPI interface in two categories. The first category of commands are called "Extended Packet ". The second category commands are called "Low Level Packet". An Extended Packet is represented as an array of bytes with a maximum array size up to 511 bytes. [NOTE: The packet is limited to 511 bytes because if a packet is 512 bytes the additional Dummy byte will cause the data length to be 513 and the dummy byte will be pushed into the next packet]

Commands larger than one Extended Packet size are called as multi-packet commands. For the full list of commands supported by the extended packet communication interface, please see the List of Contents later in this section. Similar to Extended Packet the Low Level Packet is also represented as an array of bytes with a maximum size up to 511 bytes. These command packets allow low level control like Flash Programming, Firmware Update, EDID update, and direct image download. For the full list of commands, please see List of Contents of Low Level Commands in Section 7.

5 Command/Response Protocol

The DLPC200 SPI slave interface follows the command/response mode of operation. The SPI master sends a command and then reads a response from the slave. The SPI master cannot directly send the next command before reading the response for the previous command.

5.1 Sending Command

Each byte written by master is echoed back on the SLAVE_SPI_MISO pin of the DLPC200 controller. Basically the first byte in the command is echoed during transmission of the second byte of the command. Therefore the master has to send one extra dummy byte to read the entire command. This mechanism provides an option for the master to validate if the command is delivered successfully. If the master finds the command is not delivered successfully then it can resend the command again. It should be noted that in case of an unsuccessful delivery the master still has to read the command response for the unsuccessful command. Aborting command transmission in the middle is not allowed. In case of multi packet command, the master should read the response only at the end of the last packet of the command. See Figure 2 and Figure 3 for command transmit and response flow charts.

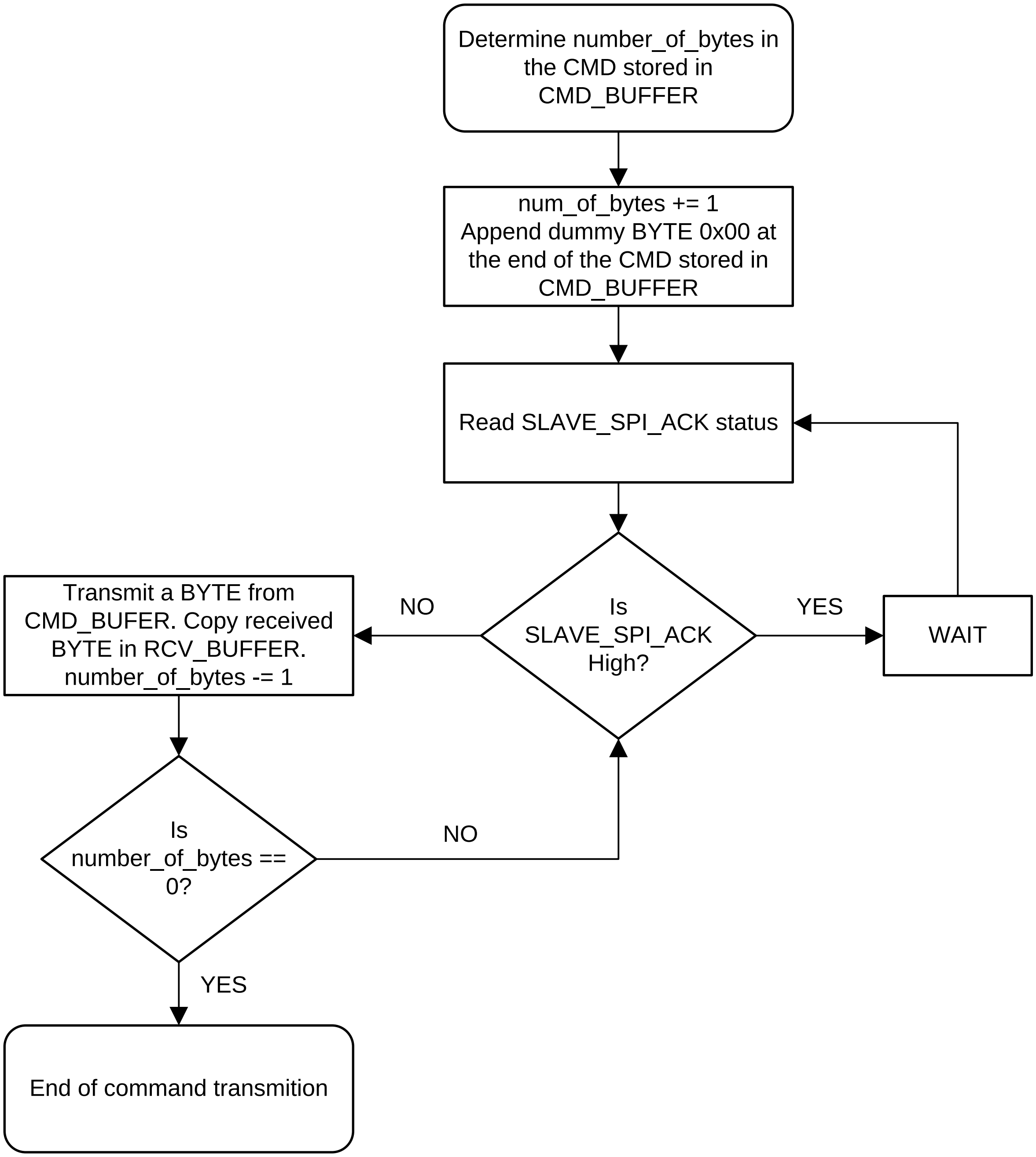

Determine number_of_bytes in the CMD stored in CMD_BUFFER
num_of_bytes += 1 Append dummy BYTE 0x00 at the end of the CMD stored in CMD_BUFFER
Read SLAVE_SPI_ACK status
Is SLAVE_SPI_ACK High?
YES
WAIT
NO
Transmit a BYTE from CMD_BUFER. Copy received BYTE in RCV_BUFFER. number_of_bytes -= 1
Is number_of_bytes == 0?
NO
YES
End of command transmition

Figure 2. Write Command Transmit Flowchart

5.2 Reading Command Response

After sending the command the DLPC200 sets SLAVE_SPI_ACK signal HIGH to indicate it is busy in processing the command packet. This signal is set to LOW at the end of command execution. The master should not read the response until the signal is LOW. The amount of time taken to process the command packet varies on the type request made in the command. To read the response from the DLPC200 the SPI master must transmit a number of dummy (0x00) bytes and the response will be sent through the SLAVE_SPI_MISO signal. The amount of dummy bytes to be sent by the master can be predetermined by the expected command response structure or determined dynamically by reading the length of the response in its header. For details on the packet response header structure see Extended Packet Definition. It is important to note, that if a trailing dummy byte was sent with the SPI command, when attempting to read the command response the dummy (0x00) byte will be echoed before the command response and must be disregarded. The read response flow chart in Figure 3 assumes that a trailing dummy byte was not sent with the SPI command.

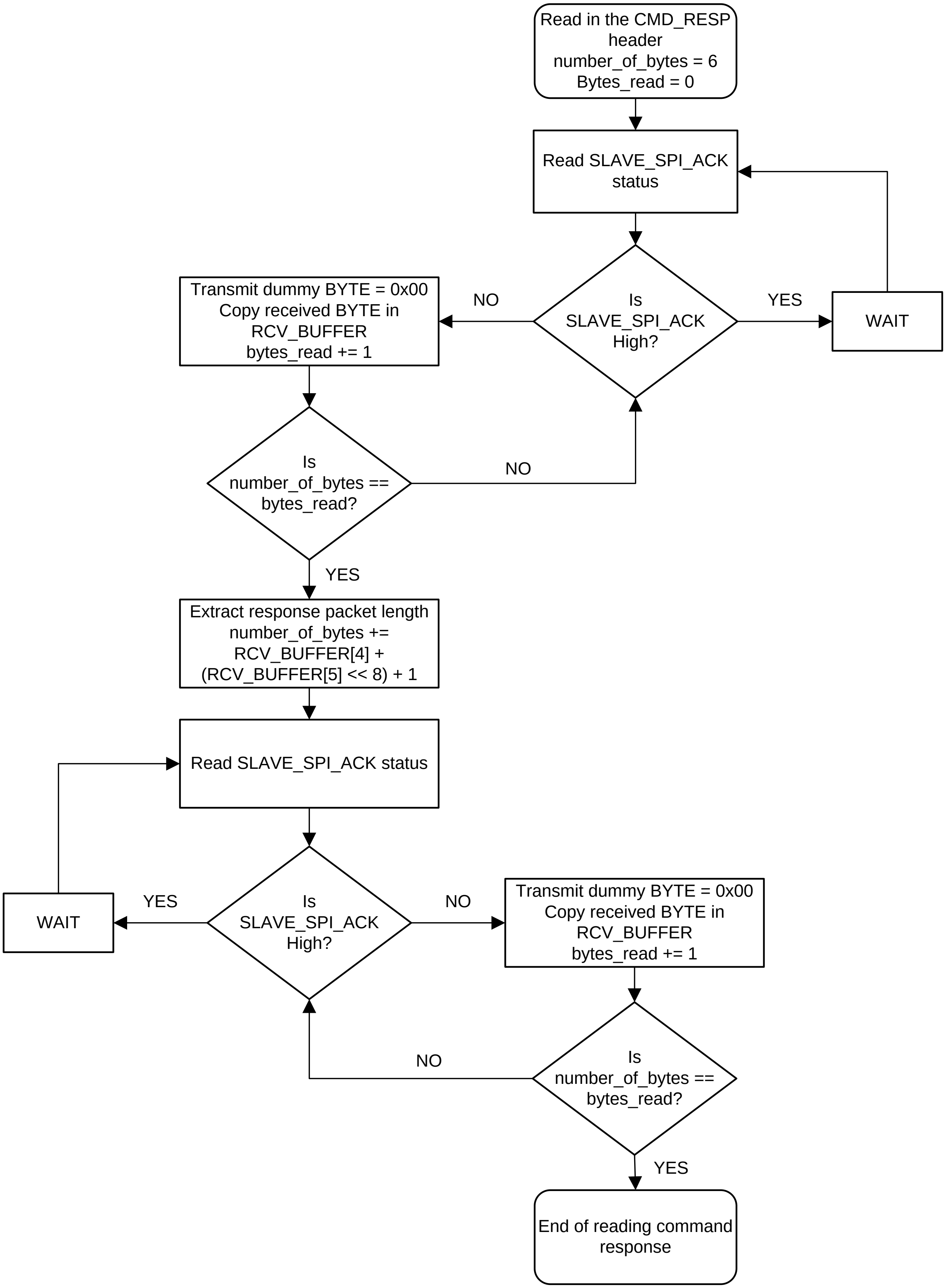

Read in the CMD_RESP header
number_of_bytes = 6
Bytes_read = 0
Read SLAVE_SPI_ACK status
Is SLAVE_SPI_ACK High?
YES
WAIT
NO
Transmit dummy BYTE = 0x00
Copy received BYTE in RCV_BUFFER
bytes_read += 1
Is number_of_bytes == bytes_read?
NO
YES
Extract response packet length
number_of_bytes += RCV_BUFFER[4] + (RCV_BUFFER[5] << 8) + 1
Read SLAVE_SPI_ACK status
Is SLAVE_SPI_ACK High?
YES
WAIT
NO
Transmit dummy BYTE = 0x00
Copy received BYTE in RCV_BUFFER
bytes_read += 1
Is number_of_bytes == bytes_read?
NO
YES
End of reading command response

Figure 3. Read Command Response Flowchart

6 Extended Packet Definition

The structure of this 511-byte extended packet is shown below:

HEADER						DATA	CHECKSUM
CMD1 1 byte	CMD2 1 byte	CMD3 1 byte	CMD4 1 byte	Len_LSB 1 byte	Len_MSB 1 byte	0-504 bytes	1 byte

Definition of CMD1

- WRITE – 0x02 (for set/enable/configure type functionality)
- WRITE_RESP – 0x03
- READ – 0x04 (for status/query type functionality)
- READ_RESP – 0x05

Definition of CMD2

- Unique ID assigned to CMD2 as 0xAA to identify the packet is of Extended Packet type.

Definition of CMD3

- CMD3 is reserved or unused. By default CMD3 should be set to 0x00

Definition of CMD4

- CMD4 is meant to serve as a flag to track where in a multi-packet transfer it is. For example, there may be many packets needed to download an ImageOrderLut (CMD ID = 0x000D). This flag will track where in this packet transfer stream it is – beginning, middle, or end.
  - Only transfer of command – 0x00, that is single register accesses
  - First of many transfers – 0x01
  - Middle of many transfer – 0x02
  - Last of many transfers – 0x04

At a high level the commands are categorized as Set/Enable/Configure type or Get/GetStatus type. For all Set/Enable/Configure type commands the response is the same. Upon receipt of a command, the DLPC200 will send a response back to the master. If the command was received and accepted, a value of 0x0000 would be returned in the data buffer. It would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x03 (WRITE_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Checksum = 0x02 (of everything after CMD4)

If the transfer was corrupted in some way then the DLPC200 will set certain bits in the data buffer Data[0] and Data[1] as indicated in the following table.

Data[0][1][2]	
Bit 0: CMD_ERR_CHK_SUM_ERROR	Request packet has checksum error.
Bit 1: CMD_ERR_INVALID_CMD1	Request packet has invalid CMD1 value.
Bit 2: CMD_ERR_INVALID_CMD2	Request packet has invalid CMD2 value.

[1] The flags shaded in GRAY will be applicable for the Extended Packet response; other flags are not applicable and will be set to '0'.

[2] If the packet is processed without errors then all the flags defined in Data[0] and Data[1] of response packet will be set to '0' to indicate success.

Data[0] [1][2]	
Bit 3: CMD_ERR_INVALID_CMD3	Not applicable for Extended Packet.
Bit 4: CMD_ERR_INVALID_CMD4	Request packet has invalid CMD4 value.
Bit 5: CMD_ERR_INVALID_ADDRESS	Not applicable for Extended Packet.
Bit 6: CMD_ERR_CMD_EXE_FAILED	Extended Packet processing failed.
Bit 7: CMD_ERR_ABRUPT_TERM_OF_MUL_PKT_CMD	Will be set when a packet containing a CMD4 value of 0x01 or 0x00 is sent after a packet containing CMD4 value of 0x01 or 0x02.
Data[1] [1][2][3]	
Bit 0: RFU (Reserved for Future Use)	Not applicable for Extended Packet.
Bit 1: RFU	Not applicable for Extended Packet.
Bit 2: RFU	Not applicable for Extended Packet.
Bit 3: CMD_ERR_INSUF_CMD_DATA	Insufficient or excess data passed in the extended command packet.
Bit 4: CMD_ERR_INVALID_ADD_OFST	Not applicable for Extended Packet.
Bit 5: CMD_ERR_FLSH_DWLD_FAIL	Not applicable for Extended Packet.
Bit 6: CMD_ERR_EDID_UPDATE_FAIL	Not applicable for Extended Packet.
Bit 7: CMD_ERR_CORRUPT_PKT_RCVD	Not applicable for Extended Packet.

[3] If error encountered while processing a command one of the flag (shaded in GRAY) along with Bit 6 is SET. In the event that only Bit 6 of Data[0] is SET, GetExtendedPktFailReason (CMD ID = 0x0000) can be sent to find specific reason for failure.

The response for Get/GetStatus type commands are different for each command, refer to a command's detailed specifications for the expected response structure.

Table 1. List of Commands

SECTION NO.	TITLE
6.1	GetExtendedPktFailReason (0x0000)
6.2	DisplayPatternManualStep (0x0001)
6.3	DisplayPatternManualForceFirstPattern (0x0002)
6.4	DisplayPatternAutoStepRepeatForMultiplePasses (0x0003)
6.5	DisplayStop (0x0004)
6.6	ParkDMD (0x0005)
6.7	UnparkDMD (0x0006)
6.8	SetDegammaEnable (0x0007)
6.9	HorizontalFlip (0x0008)
6.10	VerticalFlip (0x0009)
6.11	LEDintensity (0x000A)
6.12	LEDdriverEnable (0x000B)
6.13	SetLEDEnable (0x000C)
6.14	WriteImageOrderLut (0x000D)
6.15	SetDataSource (0x000E)
6.16	SetExternalTriggerEdge (0x000F)
6.17	SetTestPattern (0x0010)
6.18	SetSyncEnable (0x0011)
6.19	SyncConfigure (0x0012)
6.20	GetDMDparkState (0x0013)
6.21	GetDMDhardwareParkState (0x0014)
6.22	GetDMDsoftwareParkState (0x0015)
6.23	GetSeqRunState (0x0016)
6.24	GetEEPROMfault (0x0017)
6.25	GetDADfault (0x0018)
6.26	GetLEDdriverFault (0x0019)
6.27	GetUARTfault (0x001A)
6.28	GetFlashProgrammingMode (0x001B)
6.29	GetDADcommStatus (0x001C)
6.30	GetDMDcommStatus (0x001D)
6.31	GetLEDcommStatus (0x001E)
6.32	GetSeqDataMode (0x001F)
6.33	GetSeqDataNumPatterns (0x0020)
6.34	GetSeqDataBPP (0x0021)
6.35	GetSeqDataFrameRate (0x0022)
6.36	GetSeqDataExposure (0x0023)
6.37	GetFlashSeqCompilerVersion (0x0024)
6.38	GetDlpControllerSWVersion (0x0025)
6.39	GetDlpControllerVersion (0x0026)
6.40	GetBISTdone (0x0027)
6.41	GetBISTfail (0x0028)
6.42	GetInitFromParallelFlashFail (0x0029)
6.43	GetOverallLEDlampLitState (0x002A)
6.44	GetLEDdriverLitState (0x002B)
6.45	GetOverallLEDdriverTempTimeoutState (0x002C)
6.46	GetLEDdriverTempTimeoutState (0x002D)
6.47	GetOverallLEDdriverStrobeTimeoutState (0x002E)
6.48	GetLEDdriverStrobeTimeoutState (0x002F)
6.49	DownloadBPPfromFlashToExtMem (0x0030)
6.50	LoadSolutionFromFlash (0x0031)
6.51	PWMSeqEnable (0x0032)

Table 1. List of Commands (continued)

SECTION NO.	TITLE
6.52	DisplayPatternAutoStepForSinglePass (0x0033)
6.53	GenerateSWVSync(0x0034)
6.54	ConfigurePWMPeriod(0x0035)
6.55	ConfigurePWMDutyCycle(0x0036)

6.1 GetExtendedPktFailReason (0x0000)

This packet is used to find the reason the last Extended Packet sent failed. The two bytes allocated Data[0] Data[1] in the response packet will give some level of information which is generic in nature. This packet will give reason for failure specific to Extended Packets.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x00 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x02 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x04
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
//Extended Packet processing failure reason 16bit value
Data[2] = 0xXX (LSB)
Data[3] = 0xXX (LSB+1)
Checksum = 0xXX (sum of everything after CMD4)

To interpret 16bit Extended Packet Processing failure reason use the values below:

0x0000: No error occurred while processing the packet.
0x0001: Unknown Extended Packet ID
0x0002: CMD1 mismatch that is, the packet is query or status type but packet received as write (CMD1 = 0x02) OR the packet is set/configure type while the packet received as read (CMD1 = 0x04)
0x0003: Invalid or wrong parameter the packet definition does not match.
0x0004: Test Pattern display failed; system is NOT in “Video Mode”.
0x0005: Load Solution failed - Device cannot be accessed
0x0006: Load Solution failed - Solution offset address is invalid
0x0007: Load Solution failed - Flash is not programmed
0x0008: Load Solution failed - Failures detected while loading the Solution
0x0009 – 0x00FF: Reserved/Unused

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

NOTE: After reading the reason for failure, the error would be reset. That means sending this packet twice (one after another) returns 0x0000 (no error) for the second packet.

6.2 DisplayPatternManualStep (0x0001)

Repeatedly displays the next structured light image (as defined in the Image Order LUT-0x000D). Once bit plane patterns have been loaded into external memory and the DLP Control Chip is fully configured, this function can be called to command the DMD and DLP Controller Chip to display the patterns as defined in the Image Order LUT. If multiple image patterns are available for display, then each pattern can be displayed upon request by calling this function (repeatedly). When using an internally-programmed VSYNC, a single pattern will be displayed continuously, which allows for (human) visual inspection.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x01 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x03 (sum of everything after CMD4)

6.3 DisplayPatternManualForceFirstPattern (0x0002)

Repeatedly displays the first structured light image (as defined in the Image Order LUT-0x000D). Once bit plane patterns have been loaded into external memory, and the DLP Control Chip is fully configured, this function can be called to command the DMD and DLP Controller Chip to display the first pattern as defined in the Image Order LUT. Use this function in conjunction with DLP_Display_Display Pattern Manual Step to start at the beginning of the Image Order LUT and index through it to see its corresponding images.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x02 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x04 (sum of everything after CMD4)

6.4 DisplayPatternAutoStepRepeatForMultiplePasses (0x0003)

Continuously displays all structured light images (as defined in Image Order LUT-0x000D), one image per frame. Once bit plane patterns have been loaded into external memory, and the DLP Control Chip is fully configured, this function can be called to command the DMD and DLP Controller Chip to display the patterns. If multiple image patterns are available for display, then the series of patterns can repeatedly be displayed by calling this function (once). If there are 10 patterns available for display, calling this function will display all 10 patterns then repeat.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x03 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x05 (sum of everything after CMD4)

6.5 DisplayStop (0x0004)

This function will disable the display of bit plane patterns.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x04 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x06 (sum of everything after CMD4)

6.6 ParkDMD (0x0005)

Mirrors in the DMD will be parked upon receipt of this packet. LEDs will be turned off so that there will not be any light output.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x05 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x07 (sum of everything after CMD4)

6.7 UnparkDMD (0x0006)

Mirrors in the DMD will be unparked upon receipt of this packet.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x06 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x08 (sum of everything after CMD4)

NOTE: Unlike ParkDMD LEDs will not be turned ON upon receipt of this packet. This is done to preserve the user set state for the LEDs ON or OFF.

6.8 SetDegammaEnable (0x0007)

Enable/disable degamma function. The degamma function is typically associated with video data and is used to linearize the light output (by removing the applied Gamma transfer function). When enabling degamma, a corresponding LUT must have already been programmed with non-zero values. The degamma LUT is an output of the LOGIC application SW.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x07 (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x00 (disable degamma)
0x01 (enable degamma)
Checksum = 0x0X (sum of everything after CMD4)

6.9 HorizontalFlip (0x0008)

Enable/disable horizontal flip of the display image/pattern.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x08 (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x00 (disable horizontal flip)
0x01 (enable horizontal flip)
Checksum = 0x0X (sum of everything after CMD4)

6.10 VerticalFlip (0x0009)

Enable/disable vertical flip of the displayed image/pattern.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x09 (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x00 (disable vertical flip)
0x01 (enable vertical flip)
Checksum = 0x0X (sum of everything after CMD4)

6.11 LEDintensity (0x000A)

Sets the LED intensity. Upon receiving this packet, the DLP Controller SW sends a request to the LED driver to supply current to the LED using the value specified in the packet. For 100% intensity, the LED will be driven using the max allowed LED current.

For example, if the max LED current is 10 A, then sending this packet with a value of 50 (decimal) will set the LED current to 5 A (=10 A×50/100).

```
PACKET STRUCTURE:
        CMD1 = 0x02 (WRITE)
        CMD2 = 0xAA (Extended cmd packet)
        CMD3 = 0x00
        CMD4 = 0x00 (only packet of transfer)
        Len_LSB = 0x05
        Len_MSB = 0x00
        Data[0] = 0x0A (Extended packet ID 16bit value)
        Data[1] = 0x00
        Data[2] =  0x00 (Red LED)
                   0x01 (Green LED)
                   0x02 (Blue LED)
                   0x03 (IR LED)
        //Intensity in percentage in 8.8 format, 0.0% = 0x0000 and 100.0% = 0x6400
        Data[3] = 0xXX (integer portion of 8.8 format)
        Data[4] = 0xXX (fractional portion of 8.8 format)
        Checksum = 0xXX (sum of everything after CMD4)
```

GetLEDintensity (READ)

Gets the LED intensity being applied to the LEDs.

```
 PACKET STRUCTURE:
         CMD1 = 0x04 (READ)
         CMD2 = 0xAA (Extended cmd packet)
         CMD3 = 0x00
         CMD4 = 0x00 (only packet of transfer)
         Len_LSB = 0x03
         Len_MSB = 0x00
         Data[0] = 0x0A (Extended packet ID 16bit value)
         Data[1] = 0x00
         Data[2] =  0x00 (Red LED)
                    0x01 (Green LED)
                    0x02 (Blue LED)
                    0x03 (IR LED)
         Checksum = 0xXX (sum of everything after CMD4)
```

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

```
 RESPONSE PACKET STRUCTURE
         CMD1 = 0x05 (READ_RESP)
         CMD2 = 0xAA
         CMD3 = 0x00
         CMD4 = 0x00 (only packet of transfer)
         Len_LSB = 0x04
         Len_MSB = 0x00
         Data[0] = 0x00 (request processed successfully)
         Data[1] = 0x00 (request processed successfully)
         //Intensity in percentage in 8.8 format, 0.0% = 0x0000 and 100.0% = 0x6400
         Data[2] = 0xXX (integer portion of 8.8 format)
         Data[3] = 0xXX (fractional portion of 8.8 format)
         Checksum = 0xXX (sum of everything after CMD4)
```

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.12 LEDdriverEnable (0x000B)

This is the debug packet for testing driver enable/disable. Send 0 to disable LED driver; send 0×01 to enable. This packet can be used to re-enable the LED driver for cases where it has "timed out" and become disabled because of missing strobes or over temperature.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x0B (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x01 (enable driver)
0x00 (disable driver)
Checksum = 0x0X (sum of everything after CMD4)

6.13 SetLEDEnable (0x000C)

Debug interface to enable or disable individual LED driver channel. If the PWM Seq has been built with the specified LED, then this API can be used to disable (then re-enable) the LED.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x04
Len_MSB = 0x00
Data[0] = 0x0C (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x00 (Red LED)
0x01 (Green LED)
0x02 (Blue LED)
0x03 (IR LED)
Data[3] = 0x00 (disable LED)
0x01 (enable LED)
Checksum = 0xXX (sum of everything after CMD4)

NOTE: If the specified LED does not have a PWM sequence built in the current solution, enabling the LED using this command will not cause the LED to turn on.

6.14 WriteImageOrderLut (0x000D)

Sets the Image Order LUT. This function allows users to define the display order for image data stored in external memory which will be used in structured light mode. This allows the display order of the images to be changed without having to re-load the image data to external memory.

Example: Three 8-bit image patterns are stored in external memory as {image 0, image 1, image2}, but user wants display order = {2, 1, 0}

The LightCommander™ system has 1Gbit NOR flash to store the patterns. On a 1Gbit flash the maximum number of patterns allowed are 960 for 1bpp type and 120 for 8bpp type patterns. Due to the packet structure limitation, a maximum of 249 patterns information allowed in a packet. When the number of patterns exceeds 249, a multi-packet transaction will be required. Therefore maximum of 4 packets will be required to load the Image Order LUT for entire 960 1bpp patterns in the external memory. Typically, for 8bpp patterns all the 120 patterns can fit into a single packet.

Case 1:

The number of patterns in Image Order LUT is ≤ 249 (single packet transaction), below is an example with 250 entries is shown.

```
PACKET STRUCTURE:
        CMD1 = 0x02 (WRITE)
        CMD2 = 0xAA (Extended cmd packet)
        CMD3 = 0x00
        CMD4 = 0x00 (only packet of transfer)
        Len_LSB = 0xF9
        Len_MSB = 0x01
        Data[0] = 0x0D (Extended packet ID 16bit value)
        Data[1] = 0x00

        //Bit per pixel, 0x01 = 1bpp or 0x08 = 8bpp
        Data[2] = 0x01

        //Number of images in pattern order LUT
        //Valid range 0-960 if 1bpp, else 0-120
        Data[3] = 0xFA (LSB)
        Data[4] = 0x00 (LSB+1)

        //Image pattern display order entries
        //1st image: external memory image index [0:959]
        Data[5] = 0xXX
        Data[6] = 0xXX
        //2nd image: external memory image index [0:959]
        Data[7] = 0xXX
        Data[8] = 0xXX
        ...
        //249th image: external memory image index [0:959]
        Data[501] = 0xXX
        Data[502] = 0xXX

        Checksum = 0xXX (sum of everything after CMD4)
```

The response for LUT entires of a single packet, will have the same response as a typical Set/Enable/Configure command, see for details.

Case 2:

The number of patterns > 249 (multi-packet transaction). The example shown below is for 960 LUT entries.

NOTE: Only fetch write response after sending all packets (e.g. don't get write response after each packet).

```
PACKET STRUCTURE (1st packet):
        CMD1 = 0x02 (WRITE)
        CMD2 = 0xAA (Extended cmd packet)
        CMD3 = 0x00
        CMD4 = 0x01 (First of many)
        Len_LSB = 0xF9
        Len_MSB = 0x01
        Data[0] = 0x0D (Extended packet ID 16bit value)
        Data[1] = 0x00

        //Bits per pixel, 0x01 = 1bpp or 0x08 = 8bpp//
        Data[2] = 0x01

        //Number of images in pattern order LUT
        //Valid range 0-960 if 1bpp, else 0-120
        Data[3] = 0x03 (LSB)
        Data[4] = 0xC0 (LSB+1)

        //Image pattern display order entries
        //1st image: external memory image index [0:959]
        Data[5] = 0xXX
        Data[6] = 0xXX
        //2nd image: external memory image index [0:959]
        Data[7] = 0xXX
        Data[8] = 0xXX
        ...
        //249th image: external memory image index [0:959]
        Data[501] = 0xXX
        Data[502] = 0xXX

        Checksum = 0xXX (sum of everything after CMD4)


PACKET STRUCTURE (2nd packet):
        CMD1 = 0x02 (WRITE)
        CMD2 = 0xAA (Extended cmd packet)
        CMD3 = 0x00
        CMD4 = 0x02 (Intermediate packet)
        Len_LSB = 0xF9
        Len_MSB = 0x01
        Data[0] = 0x0D (Extended packet ID 16bit value)
        Data[1] = 0x00
```

```
            //Bit per pixel, 0x01 = 1bpp or 0x08 = 8bpp//
            Data[2] = 0x01

            //Number of images in pattern order LUT
            //Valid range 0-960 if 1bpp, else 0-120
            Data[3] = 0x03 (LSB)
            Data[4] = 0xC0 (LSB+1)

            //Image pattern display order entries
            //250th image: external memory image index [0:959]
            Data[5] = 0xXX
            Data[6] = 0xXX
            //252nd image: external memory image index [0:959]
            Data[6] = 0xXX
            Data[7] = 0xXX
            ...
            //500th image: external memory image index [0:959]
            Data[501] = 0xXX
            Data[502] = 0xXX

            Checksum = 0xXX (sum of everything after CMD4)


      PACKET STRUCTURE (3rd packet):
            CMD1 = 0x02 (WRITE)
            CMD2 = 0xAA (Extended cmd packet)
            CMD3 = 0x00
            CMD4 = 0x02 (Intermediate packet)
            Len_LSB = 0xF9
            Len_MSB = 0x01
            Data[0] = 0x0D (Extended packet ID 16bit value)
            Data[1] = 0x00

            //Bit per pixel, 0x01 = 1bpp or 0x08 = 8bpp
            Data[2] = 0x01

            //Number of images in pattern order LUT
            //Valid range 0-960 if 1bpp, else 0-120
            Data[3] = 0x03 (LSB)
            Data[4] = 0xC0 (LSB+1)

            //Image pattern display order entries
            //501st image: external memory image index [0:959]
            Data[5] = 0x0X
            Data[6] = 0xXX
            //502nd image: external memory image index [0:959]
            Data[7] = 0x0X
            Data[8] = 0xXX
            ...
            //750th image: external memory image index [0:959]
            Data[501] = 0x0X
            Data[502] = 0xXX
```

Checksum = 0xXX (sum of everything after CMD4)

PACKET STRUCTURE (4th and last packet):

```
CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x04 (last of many packet)
Len_LSB = 0xA9
Len_MSB = 0x01
Data[0] = 0x0D (Extended packet ID 16bit value)
Data[1] = 0x00

//Bit per pixel, 0x01 = 1bpp or 0x08 = 8bpp
Data[2] = 0x01

//Number of images in pattern order LUT
//Valid range 0-960 if 1bpp, else 0-120
Data[3] = 0x03 (LSB)
Data[4] = 0xC0 (LSB+1)

//Image pattern display order entries
//751st image: external memory image index [0:959]
Data[5] = 0xXX
Data[6] = 0xXX
//752nd image: external memory image index [0:959]
Data[7] = 0xXX
Data[8] = 0xXX
...
//960th image: external memory image index [0:959]
Data[423] = 0xXX
Data[424] = 0xXX

Checksum = 0xXX (sum of everything after CMD4)
```

Upon receipt of the last packet, the DLPC200 will send a response back to the master. If the command was received and accepted, Data[0] = 0x00 and Data[1] = 0x00 in the command response. The response packet will have the following structure:

RESPONSE PACKET STRUCTURE:

```
CMD1 = 0x03 (WRITE_RESP)
CMD2 = 0x06 (Function Group name)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x08
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)

// Zero bytes
Data[2] = 0x00
Data[3] = 0x00
```

// No. of packets received by controller, 32bit value
Data[4] = 0xXX (LSB)
Data[5] = 0xXX (LSB+1)
Data[6] = 0xXX (LSB+2)
Data[7] = 0xXX (LSB+3)

Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

bpp — bits-per-pixel

1bpp — Pixel represented in 1bit.

For example the number of bits required to represent a 1bpp XGA pattern will be 1024*768*1 = 786432 bits

8bpp — Pixel represented in 8bits

For example the number of bits required to represent a 8bpp XGA pattern will be 1024*768*8 = 6291456 bits

NOTE: Sending wrong info on bpp and/or number of images (more than actual patterns loaded in the DDR2 memory) and/or wrong pattern slot number will result in displaying corrupted data.

NOTE: On encountering an invalid slot number (slot number > 959) no further processing of the packet will be done. GetExtendedPktFailReason packet will return the appropriate error code tht is, 0x0003: Invalid or wrong parameter.

6.15 SetDataSource (0x000E)

Sets the input data source.

```
 PACKET STRUCTURE:
        CMD1 = 0x02 (WRITE)
        CMD2 = 0xAA (Extended cmd packet)
        CMD3 = 0x00
        CMD4 = 0x00 (only packet of transfer)
        Len_LSB = 0x03
        Len_MSB = 0x00
        Data[0] = 0x0E (Extended packet ID 16bit value)
        Data[1] = 0x00
        Data[2] =  0x00 (DVI - port 0)
                   0x01 (Expansion port - port 1)
                   0x02 (Test pattern generator, frame triggers VSYNC)
                   0x03 (Structured light auto gen)
                   0x04 (Structured light external trigger 3.3V)
                   0x05 (Structured light external trigger 1.8V)
                   0x06 (Structured light software generated Trigger)
        Checksum = 0xXX (sum of everything after CMD4)
```

6.16 SetExternalTriggerEdge (0x000F)

Programs the trigger edge.

```
 PACKET STRUCTURE:
        CMD1 = 0x02 (WRITE)
        CMD2 = 0xAA (Extended cmd packet)
        CMD3 = 0x00
        CMD4 = 0x00 (only packet of transfer)
        Len_LSB = 0x03
        Len_MSB = 0x00
        Data[0] = 0x0F (Extended packet ID 16bit value)
        Data[1] = 0x00
        Data[2] =  0x00 (Falling Edge)
                   0x01 (Rising Edge)
        Checksum = 0xXX (sum of everything after CMD4)
```

6.17 SetTestPattern (0x0010)

When the input source is Test Pattern Generator, this command programs which test pattern to display. Valid values for spatial repetition parameter in XGA type device are 1,2,4,8,16,32,64,128,256,512. Patterns that display horizontally will repeat according to this parameter. Due to the non-power of 2 vertical DMD resolutions, vertical repetition of patterns is loosely based on the spatial repetition value. Test Patterns can be displayed in Video Mode only.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x06
Len_MSB = 0x00
Data[0] = 0x10 (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x00 (Solid Field)
0x01 (Horizontal Ramp)
0x02 (Vertical Ramp)
0x03 (Horizontal Lines)
0x04 (Diagonal Lines)
0x05 (Vertical Lines)
0x06 (Horizontal Stripes)
0x07 (Vertical Stripes)
0x08 (Grid)
0x09 (Checkerboard)
Data[3] = 0x00 (Black)
0x01 (Red)
0x02 (Green)
0x03 (Blue)
0x04 (Yellow)
0x05 (Cyan)
0x06 (Magenta)
0x07 (White)
//Pattern spatial repeat frequency (16bit value)
Data[4] = 0xXX (LSB)
Data[5] = 0xXX (LSB+1)
Checksum = 0xXX (sum of everything after CMD4)

6.18 SetSyncEnable (0x0011)

There are three sync outputs available for the external source trigger. This packet used to enable/disable the specified output sync.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)

Len_LSB = 0x04
Len_MSB = 0x00
Data[0] = 0x11 (Extended packet ID 16bit value)
Data[1] = 0x00
//Sync number to enable
Data[2] = 0x01 (Sync# 1)
0x02 (Sync# 2)
0x03 (Sync# 3)
Data[3] = 0x01 (enable Sync)
0x00 (disable Sync)
Checksum = 0xXX (sum of everything after CMD4)

6.19 SyncConfigure (0x0012)

There are three sync outputs available for the external source trigger. This packet is used to configure the specified output sync.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x0C
Len_MSB = 0x00
Data[0] = 0x12 (Extended packet ID 16bit value)
Data[1] = 0x00

//Sync channel to configure
Data[2] = 0x01 (Sync# 1)
0x02 (Sync# 2)
0x03 (Sync# 3)

//Sync polarity
Data[3] = 0x01 (Positive)
0x00 (Negative)

//Output sync delay in microseconds 32bit value
Data[4] = 0xXX (LSB)
Data[5] = 0xXX (LSB+1)
Data[6] = 0xXX (LSB+2)
Data[7] = 0xXX (LSB+3)

//Output sync pulse width in microseconds 32bit value
Data[8] = 0xXX (LSB)
Data[9] = 0xXX (LSB+1)
Data[10] = 0xXX (LSB+2)
Data[11] = 0xXX (LSB+3)

Checksum = 0xXX (sum of everything after CMD4)

6.20 GetDMDparkState (0x0013)

Gets the DMD park state.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x13 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x15 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (DMD is un-parked)
0x01 (DMD is parked)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.21 GetDMDhardwareParkState (0x0014)

Helps in determining if DMD isparked by the hardware switch.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x14 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x16 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (hardware switch not activated)
0x01 (DMD park active by hardware switch)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.22 GetDMDsoftwareParkState (0x0015)

Determines if the DMD parked state is caused by a software issued command (that is ParkDMD – 0x0005).

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x15 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x17 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (DMD park not requested by the software)
0x01 (DMD park requested by the software)

Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.23 GetSeqRunState (0x0016)

Gets the PWM sequence run state: running or stopped.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x16 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x18 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (PWM sequence stopped)
0x01 (PWM sequence running)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.24 GetEEPROMfault (0x0017)

The EEPROM device stores the EDID info. This packet is used to check if an EEPROM fault is detected.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)

Len_LSB = 0x02

Len_MSB = 0x00

Data[0] = 0x17 (Extended packet ID 16bit value)

Data[1] = 0x00

Checksum = 0x19 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)

CMD2 = 0xAA

CMD3 = 0x00

CMD4 = 0x00 (only packet of transfer)

Len_LSB = 0x03

Len_MSB = 0x00

Data[0] = 0x00 (request processed successfully)

Data[1] = 0x00 (request processed successfully)

Data[2] = 0x00 (no fault detected)
0x01 (fault detected)

Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.25 GetDADfault (0x0018)

Checks for DLPA200 (aka DAD = Analog reset driver) fault.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)

CMD2 = 0xAA (Extended cmd packet)

CMD3 = 0x00

CMD4 = 0x00 (only packet of transfer)

Len_LSB = 0x02

Len_MSB = 0x00

Data[0] = 0x18 (Extended packet ID 16bit value)

Data[1] = 0x00

Checksum = 0x1A (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)

CMD2 = 0xAA

CMD3 = 0x00

CMD4 = 0x00 (only packet of transfer)

Len_LSB = 0x03

Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (no fault detected)
0x01 (fault detected)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.26 GetLEDdriverFault (0x0019)

Checks for LED Driver fault.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x19 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x1B (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (no fault detected)
0x01 (fault detected)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.27 GetUARTfault (0x001A)

Checks for UART port fault.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x1A (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x1C (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (no fault detected)
0x01 (fault detected)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.28 GetFlashProgrammingMode (0x001B)

Checks if the system is in flash programming mode.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x1B (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x1D (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (normal mode)
0x01 (flash programming mode)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.29 GetDADcommStatus (0x001C)

Gets DLPA200/DAD SPI communication status.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x1C (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x1E (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (DAD comm-stat OK)
0x01 (DAD comm-stat Failure)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.30 GetDMDcommStatus (0x001D)

Gets DMD SPI communication status.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x1D (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x1F (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (DMD comm-stat OK)
0x01 (DMD comm-stat Failure)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.31 GetLEDcommStatus (0x001E)

Gets LED Driver SPI communication status.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00

Data[0] = 0x1E (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x20 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:
CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (LED comm-stat OK)
0x01 (LED comm-stat Failure)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.32 GetSeqDataMode (0x001F)

Gets sequence data mode. Meta data describing the currently loaded PWM sequence.

PACKET STRUCTURE:
CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x1F (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x21 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:
CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)

Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (Structured light, non real-time input)
0x01 (Structured light, real-time input)
0x02 (Video mode)
0x03 (Video plus Structured Light)
0x04 (Object mode)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.33 GetSeqDataNumPatterns (0x0020)

Gets sequence data - number of patterns per frame. Meta data describing the number of patterns per frame is used for building currently loaded PWM sequences.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x20 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x22 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
//Number of patterns output per frame (16bit value)
Data[2] = 0xXX (LSB)
Data[3] = 0xXX (LSB+1)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.34 GetSeqDataBPP (0x0021)

Gets sequence data BPP (number of bits per pixel). Meta data describing the BPP used for building the currently loaded PWM sequence. BPP is only applicable in structured light mode.

PACKET STRUCTURE:

```
    CMD1 = 0x04 (READ)
    CMD2 = 0xAA (Extended cmd packet)
    CMD3 = 0x00
    CMD4 = 0x00 (only packet of transfer)
    Len_LSB = 0x02
    Len_MSB = 0x00
    Data[0] = 0x21 (Extended packet ID 16bit value)
    Data[1] = 0x00
    Checksum = 0x23 (sum of everything after CMD4)
```

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

```
    CMD1 = 0x05 (READ_RESP)
    CMD2 = 0xAA
    CMD3 = 0x00
    CMD4 = 0x00 (only packet of transfer)
    Len_LSB = 0x03
    Len_MSB = 0x00
    Data[0] = 0x00 (request processed successfully)
    Data[1] = 0x00 (request processed successfully)
    Data[2] =  0x01 (1bpp)
               0x08 (8bpp)
    Checksum = 0xXX (sum of everything after CMD4)
```

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.35 GetSeqDataFrameRate (0x0022)

Meta data describing the frame rate (in Hz) used for building the currently loaded PWM sequence.

PACKET STRUCTURE:

```
    CMD1 = 0x04 (READ)
    CMD2 = 0xAA (Extended cmd packet)
    CMD3 = 0x00
    CMD4 = 0x00 (only packet of transfer)
    Len_LSB = 0x02
```

Len_MSB = 0x00
Data[0] = 0x22 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x24 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x05
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
//Sequence frame rate 20bit value in u16.4 format
Data[2] = 0xXX (LSB)
Data[3] = 0xXX (LSB+1)
Data[4] = 0xXX (LSB+2)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.36 GetSeqDataExposure (0x0023)

Gets sequence data for structured light exposure. Meta data describing the exposure time (in µsec) used for building the currently loaded PWM sequence.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x23 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x25 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00

CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x04
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
//Exposure time in uSecs (16bit value)
Data[2] = 0xXX (LSB)
Data[3] = 0xXX (LSB+1)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.37 GetFlashSeqCompilerVersion (0x0024)

Gets the version number of the sequence compiler DLL used to build the PWM sequence. Format: Major.Minor.patch.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x24 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x26 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x05
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0xXX (major version)
Data[3] = 0xXX (minor version)
Data[4] = 0xXX (patch version)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.38 GetDlpControllerSWVersion (0x0025)

Gets the DLPC200 controller software version. Format: Major.Minor.Patch.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x25 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x27 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x05
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0xXX (major version)
Data[3] = 0xXX (minor version)
Data[4] = 0xXX (patch version)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.39 GetDlpControllerVersion (0x0026)

Gets the DLPC200 HW version number. Format: Major.Minor.patch.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02

Len_MSB = 0x00
Data[0] = 0x26 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x28 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x06
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0xXX (major version)
Data[3] = 0xXX (minor version)
Data[4] = 0xXX (patch version LSB)
Data[5] = 0xXX (patch version LSB+1)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.40 GetBISTdone (0x0027)

Get External Memory BIST done state. The Packet can be used to determine if the Built-In-Self-Test (BIST) operation has completed. A BIST operation is performed on the external frame memory every power up.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x27 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x29 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (BIST Not done)
0x01 (BIST done)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.41 GetBISTfail (0x0028)

Gets the BIST fail state. Every power up, a Built-In-Self-Test (BIST) operation is run on the external memory, and the pass/fail state is saved off. This packet is used to check whether the BIST passed or failed.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x28 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x2A (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (BIST passed)
0x01 (BIST failed)

Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.42 GetInitFromParallelFlashFail (0x0029)

As part of system power on, the user stored default settings will be loaded from parallel flash. This packet returns the result of initialization from flash.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x29 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x2B (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
// Parallel flash initialization status
Data[2] = 0x00 (Success)
0x01 (Failed for unknown reason)
0x02 (No flash init attempt was made)
0x03 (Fail mode - Parallel flash could not be accessed)
0x04 (Fail mode - Invalid flash signature detected)
0x05 (Fail mode - Error in TI power up section)
0x06 (Fail mode - Error in user power up section)
0x07 (Fail mode - Error in requested solution)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.43 GetOverallLEDlampLitState (0x002A)

Gets overall LED/Lamp lit state. Checks all channels on the LED driver to determine if all LEDs are lit (e.g. ready for operation).

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x2A (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x2C (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (LEDs Not lit)
0x01 (All LEDs lit)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.44 GetLEDdriverLitState (0x002B)

Gets the LED driver lit state by LED type. Checks for individual driver ready for operation.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03

Len_MSB = 0x00
Data[0] = 0x2B (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x00 (Red LED)
0x01 (Green LED)
0x02 (Blue LED)
0x03 (IR LED)
Checksum = 0xXX (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (LED Not lit)
0x01 (LED lit)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.45 GetOverallLEDdriverTempTimeoutState (0x002C)

Gets overall LED driver over-temperature-timeout state.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x2C (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x2E (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (LEDs operating normally)
0x01 (Driver shutdown because of over temperature)
Checksum = 0xXX (sum of everything after CMD4

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.46 GetLEDdriverTempTimeoutState (0x002D)

Gets individual LED driver over-temperature -timeout state.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x2D (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x00 (Red LED)
0x01 (Green LED)
0x02 (Blue LED)
0x03 (IR LED)
Checksum = 0xXX (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)

Data[2] = 0x00 (LEDs operating normally)
0x01 (Driver shutdown because of over temperature)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.47 GetOverallLEDdriverStrobeTimeoutState (0x002E)

Get overall LED driver strobe-timeout state.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x2E (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x30 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (LEDs operating normally)
0x01 (Driver shutdown because of strobe timeout)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.48 GetLEDdriverStrobeTimeoutState (0x002F)

Get individual LED driver strobe-timeout state.

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x2F (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x00 (Red LED)
0x01 (Green LED)
0x02 (Blue LED)
0x03 (IR LED)
Checksum = 0xXX (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (LEDs operating normally)
0x01 (Driver shutdown because of strobe timeout)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.49 DownloadBPPfromFlashToExtMem (0x0030)

This packet helps in downloading the stored image pattern from the flash to DDR2 memory. The system uses inbuilt DMA engine to download the pattern into External Memory (DDR2 memory). The packet takes three parameters namely, the DDR2 memory index, the flash offset address where pattern stored and the pattern size in number of bytes. The packet allows multiple pattern download requests. Due to packet data size limitation a maximum of 50 pattern download requests can be made in a packet.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0xXX (LSB)

Len_MSB = 0xXX (LSB+1)
Data[0] = 0x30 (Extended packet ID 16bit value)
Data[1] = 0x00

//1st pattern details//
//DDR2 memory location or pattern slot number (range: 0 to 959)
Data[2] = 0xXX (LSB)
Data[3] = 0xXX (LSB+1)
//Offset location of the flash where pattern is stored
Data[4] = 0xXX (LSB)
Data[5] = 0xXX (LSB+1)
Data[6] = 0xXX (LSB+2)
Data[7] = 0xXX (LSB+3)
//Number of bytes of pattern data to be transferred
Data[8] = 0xXX (LSB)
Data[9] = 0xXX (LSB+1)
Data[10] = 0xXX (LSB+2)
Data[11] = 0xXX (LSB+3)

//2nd pattern detail//
//DDR2 memory location pattern slot number (range: 0 to 959)
Data[12] = 0xXX (LSB)
Data[13] = 0xXX (LSB+1)
//Offset location of the flash where pattern is stored
Data[14] = 0xXX (LSB)
Data[15] = 0xXX (LSB+1)
Data[16] = 0xXX (LSB+2)
Data[17] = 0xXX (LSB+3)
//Number of bytes of pattern data to be transferred
Data[18] = 0xXX (LSB)
Data[19] = 0xXX (LSB+1)
Data[20] = 0xXX (LSB+2)
Data[21] = 0xXX (LSB+3)

//Nth pattern detail where, 2 < N ≤50 and M = (N-1)*10+2//
//DDR2 memory location pattern slot number (range: 0 to 959)
Data[M] = 0xXX (LSB)
Data[M+1] = 0xXX (LSB+1)
//Offset location of the flash where pattern is stored
Data[M+2] = 0xXX (LSB)
Data[M+3] = 0xXX (LSB+1)
Data[M+4] = 0xXX (LSB+2)
Data[M+5] = 0xXX (LSB+3)
//Number of bytes of pattern data to be transferred
Data[M+6] = 0xXX (LSB)
Data[M+7] = 0xXX (LSB+1)
Data[M+8] = 0xXX (LSB+2)
Data[M+9] = 0xXX (LSB+3)

Checksum = 0xXX (sum of everything after CMD4)

NOTE: Passing incorrect offset location would result in displaying of garbage data.

NOTE: On passing invalid DDR2 memory slot number result in Command execution fail. GetExtendedPktFailReason(0x0000) would return the “Error Invalid or wrong parameter” error number 0x0003.

NOTE: Packet is processed until the first invalid slot # in the packet.

6.50 LoadSolutionFromFlash (0x0031)

This packet loads a solution from the parallel flash. The packet contains information regarding the 32-bit solution offset address. The offset address points to the parallel flash offset address location where the solution is stored. The solution can be loaded with or without full system reset; this is available in the form of an option. The solution addresses can be found when The DLP LightCommander Control Software builds the flash image. When a project is built by selecting Build Flash Image from the Execute toolbar the flash binary is created along with a XML map of the binary file. These files are located in the LightCommander Control project directory in the Flash folder. In the XML file the StartAddress is specified for each solution.

XML map example:

```
<FlashImage>
  <Records>
    <Type>TIPowerUp</Type>
    <StartAddress>32768</StartAddress>
    <Length>512</Length>
  </Record>
  <Record>
    <Type>ActiveSolution</Type>
    <StartAddress>98304</StartAddress>
    <Length>4</Length>
  </Record>
  <Record>
    <Type>Solution</Type>
    <StartAddress>131072</StartAddress>
    <Length>15350</Length>
    <Name>SolutionName1</Name>
  </Record>
  <Record>
    <Type>Solution</Type>
    <StartAddress>262144</StartAddress>
    <Length>14336</Length>
    <Name>SolutionName2</Name>
  </Record>
```

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x07
Len_MSB = 0x00
Data[0] = 0x31 (Extended packet ID 16bit value)
Data[1] = 0x00

//Offset location of the flash where a solution is stored
Data[2] = 0xXX (LSB)
Data[3] = 0xXX (LSB+1)
Data[4] = 0xXX (LSB+2)
Data[5] = 0xXX (LSB+3)

//Load solution option
Data[6] = 0x00 (Load solution without full system reset)
0x01(Load solution with full system reset)

Checksum = 0xXX (sum of everything after CMD4)

NOTE: ‘Solution’ is basically a collection of batch-files and/or configuration data.

NOTE: Passing incorrect solution offset address would result in command response error; to know the reason for failure use GetExtendedPktFailReason query packet.

6.51 PWMSeqEnable (0x0032)

This packet controls the PWM Sequence running in the system. The packet takes a boolean parameter as ‘enable = 0x01’ or 'disable = 0x00’. When the packet is sent with the ‘enable’ option the PWM sequence starts running and when the packet is sent with the ‘disable’ option it stops.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x32 (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x00 (disable PWM Sequence)
0x01 (enable PWM Sequence)
Checksum = 0x3X (sum of everything after CMD4)

NOTE: Sending PWMSeqEnable packet with the 'disable' option turns OFF the illumination system.

PWMSeqEnable (READ)

Gets the current state of PWM Sequence

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x32 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x34 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Data[2] = 0x00 (PWM Sequence stopped)
0x01 (PWM Sequence running)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

6.52 *DisplayPatternAutoStepForSinglePass (0x0033)*

This packet enables one-time display of all structured light images (as defined in Image Order LUT-0x000D), one image per frame. Once bit plane patterns have been loaded into external memory, and the DLP Control Chip is fully configured, this function can be called to command the DMD and DLP Controller Chip to display the patterns. If multiple image patterns are available for display, then the series of patterns can be displayed by calling this function (once). If there are 10 patterns available for display, calling this function will display all 10 patterns, then display is disabled. There must be sufficient delay after issuing this command in order for the DLPC200 to receive and accept the package before issuing the next command. The necessary delay time can be calculated as follows:

- tPat = pattern exposure time (in microseconds)
- nPat = the number of patterns
- Total Pattern display time T = tPat*nPat (in microseconds)

- Be sure to have a 2*T delay before sending the next command
- A response from the DLPC200 should also be received before sending the next command

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x33 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x35 (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in data[0] and data[1] in the response packet to signify this error.

6.53 GenerateSWVsync (0x0034)

This packet generates one VSYNC inside DLP control chip. This packet works only when the data source is set to software generated trigger mode. Refer to SetDataSource (0x000E) extended packet for more details. This command performs the same functionality as the trigger pulse in External hardware trigger mode. Sending this command once invokes the same functionality as a single trigger pulse input in the external hardware trigger input mode.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x34 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x36 (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in data[0] and data[1] in the response packet to signify this error.

NOTE: This command packet is not supported by DLPC200 firmware version before 2.1.6.

6.54 ConfigurePWMPeriod (0x0035)

There are four programmable PWM ports in the DLP Control chip, refer to DLPC200 datasheet for PWM0-PWM4 I/O pin details. The PWM port(s) enable control of low cost LED driver systems that accept PWM type input signals for LED current strength adjustment. Apart from LED driver control the PWM port(s) can also be used as general purpose PWM ports.

This packet is used to configure the PWM ports period value. The DLP Control chip sets the same base period for all the four ports. Therefore, it is not possible to set different PWM period for each port. The packet takes two byte numbers ranging from 0 to 2047, which generate a period of 0 (0*40nSec) to 81.88µSec (2047*40nSec) respectively.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x04
Len_MSB = 0x00
Data[0] = 0x35 (Extended packet ID 16bit value)
Data[1] = 0x00
// PWM base period, valid range: 0 to 2047
Data[2] = 0xXX (LSB)
Data[3] = 0xXX (LSB+1)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in data[0] and data[1] in the response packet to signify this error.

ConfigurePWMPeriod (READ)

This commands packet returns the base PWM period set to the four PWM ports

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x35 (Extended packet ID 16bit value)
Data[1] = 0x00
Checksum = 0x37 (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x04
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
// base PWM period 2 Bytes
Data[2] = 0xXX (LSB)
Data[3] = 0xXX (LSB+1)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

NOTE: This command packet is not supported by DLPC200 firmware version before 2.1.6.

6.55 ConfigurePWMDutyCycle(0x0036)

This packet sets the PWM port(s) duty cycle. It requires the PWM port # and a two byte value. Upon setting the PWM period the value sent in this packet defines the output duty cycle for the PWM port .

PACKET STRUCTURE:

```
    CMD1 = 0x02 (WRITE)
    CMD2 = 0xAA (Extended cmd packet)
    CMD3 = 0x00
    CMD4 = 0x00 (only packet of transfer)
    Len_LSB = 0x05
    Len_MSB = 0x00
    Data[0] = 0x36 (Extended packet ID 16bit value)
    Date[1] = 0x00
    Data[2] =  0x00 (PWM0)
               0x01 (PWM1)
               0x02 (PWM2)
               0x03 (PWM4)
               0x04 (ALL PWM ports)
    // PWM duty cycle, valid range: 0 to 2047
    Data[3] = 0xXX (LSB)
    Data[4] = 0xXX (LSB+1)
    Checksum = 0xXX (sum of everything after CMD4)
```

For example, set the PWM Period to 0x100 via ConfigurePWMPeriod (0x0035) extended packet. This will cause a (256*40nSec) 10.24µSec period signal.

Setting a value of 0x40 in the PWM duty cycle packet will generate 25% duty cycle that is, 0x40/0x100

Setting a value of 0x80 in the PWM duty cycle packet will generate 50% duty cycle that is, 0x80/0x100

Setting a value of 0xC0 in the PWM duty cycle packet will generate 75% duty cycle that is, 0xC0/0x100

Setting a value 0x100 in the PWM duty cycle packet will generate 100% duty cycle that is, 0x100/0x100

NOTE: Setting ANY value >= PWM period will generate a 100% Duty Cycle

NOTE: Setting 0x0000 as the PWM Duty Cycle will generate a 0% Duty Cycle

If the command packet was corrupted in some way then the embedded application would set bits in data[0] and data[1] in the response packet to signify this error.

ConfigurePWMDutyCycle (READ)

Gets the base PWM DutyCycle set to the four PWM ports

PACKET STRUCTURE:

CMD1 = 0x04 (READ)
CMD2 = 0xAA (Extended cmd packet)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x03
Len_MSB = 0x00
Data[0] = 0x36 (Extended packet ID 16bit value)
Data[1] = 0x00
Data[2] = 0x00 (Get PWM0 duty cycle value)
          0x01 (Get PWM1 duty cycle value)
          0x02 (Get PWM2 duty cycle value)
          0x03 (Get PWM3 duty cycle value)
Checksum = 0x3X (sum of everything after CMD4)

Upon receipt of the packet, the DLPC200 will send a response back to the host. If the packet was received and accepted the response would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x05 (READ_RESP)
CMD2 = 0xAA
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x04
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
//PWM duty cycle configuration
Data[2] = 0xXX (LSB)
Data[3] = 0xYY (LSB+1)
Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 6 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

NOTE: PWM functionality control is supported from DLPC200 firmware version 2.1.6 onwards

7 Low Level Packet Definition

The structure of this 510-byte extended packet is shown below:

HEADER						DATA	CHECKSUM
CMD1 1 byte	CMD2 1 byte	CMD3 1 byte	CMD4 1 byte	Len_LSB 1 byte	Len_MSB 1 byte	0-505 bytes	1 byte

Definition of CMD1

- WRITE – 0x02 (for set/enable/configure type functionality)
- WRITE_RESP – 0x03
- READ – 0x04 (for status/query type functionality)
- READ_RESP – 0x05

Definition of CMD2

CMD2 is meant to be a way to group major functions together with CMD3 then enumerating the specific function or meaning within this group. At the end of this section is a table that lists the functional groups.

Definition of CMD3

As mentioned above, the value in CMD3 is a function of the value in CMD2.

- When CMD2 = 0x00, CMD3 will specify number of Address and Data pairs.
- When CMD2 = 0x03, CMD3 will specify how many LUT entries to write from the current packet
- When CMD2 = 0x04, CMD3 should be set to 0x00
- When CMD2 = 0x06, CMD3 will specify whether it is download type 0x00 - Firmware Update 0x01 - User Configuration Flash
- When CMD2 = 0x07, CMD3 will specify which flash to be erased
- When CMD2 = 0x08, CMD3 should be set to 0x00

Definition of CMD4

- CMD4 is meant to serve as a flag to track the progress of a multi-packet transfer. For example, there will be many packets needed to update firmware or download an image. This flag will track where in this packet transfer stream it is – beginning, middle, or end.
  - Only transfer of command – 0x00
  - First of many transfers – 0x01
  - Middle of many transfer – 0x02
  - Last of many transfers – 0x04

Upon receipt of a command, the DLPC200 will send a response back to the master. The master should read the response only if the sent command packet is a *single* packet transaction or if the sent packet is the *last* packet of multi packet transaction. If the command was received and accepted Data[0] = 0x00 Data[1] = 0x00 would be set in the command response. It would look like the following:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x03 (WRITE_RESP)
CMD2 = 0xXX (Function Group name)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x02
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)
Checksum = 0xXX (sum of everything after CMD4)

If the transfer was corrupted in some way then the DLPC200 will set certain bits in the data buffer Data[0] and Data[1] as mentioned below.

Data[0]	
Bit 0: CMD_ERR_CHK_SUM_ERROR	Request packet has checksum error.
Bit 1: CMD_ERR_INVALID_CMD1	Request packet has invalid CMD1 value.
Bit 2: CMD_ERR_INVALID_CMD2	Request packet has invalid CMD2 value.
Bit 3: CMD_ERR_INVALID_CMD3	Request packet has invalid CMD3 value.
Bit 4: CMD_ERR_INVALID_CMD4	Request packet has invalid CMD4 value.
Bit 5: CMD_ERR_INVALID_ADDRESS	Request packet contained invalid 16-bit address.
Bit 6: CMD_ERR_CMD_EXE_FAILED	Unknown error occurred while processing request.
Bit 7: CMD_ERR_ABRUPT_TERM_OF_MUL_PKT_CMD	Will be set when a packet containing a CMD4 value of 0x01 or 0x00 is sent after a packet containing CMD4 value of 0x01 or 0x02.
Data[1]	
Bit 0: CMD_ERR_MAILBOX_DWLD	Request packet in LUT mode has invalid mailbox name.
Bit 1: RESERVED	Reserved.
Bit 2: RESERVED	Reserved.
Bit 3: CMD_ERR_INSUF_CMD_DATA	Insufficient or excess data passed in the extended command packet.
Bit 4: CMD_ERR_INVALID_ADD_OFST	Invalid Address offset provided while flash programming.
Bit 5: CMD_ERR_FLSH_DWLD_FAIL	Error occurred while trying to access flash device.
Bit 6: CMD_ERR_EDID_UPDATE_FAIL	EDID update failed.
Bit 7: RESERVED	Reserved.

Table 2. List of Contents of Low Level Commands

SECTION NO.	TITLE
7.1	Register Access (CMD2 = 0x00)
7.2	LUT Mailbox Access (CMD2 = 0x03)
7.3	Full Image Download (CMD2 = 0x04)
7.4	Flash Download (CMD2 = 0x06)
7.5	Flash Erase (CMD2 = 0x07)
7.6	EDID Update (CMD2 = 0x08)
7.7	DLPC200 Reset (CMD2 = 0x00)

7.1 Register Access

The Register Access low level control is provided to send the register level configuration information on the SPI bus. The DLP controller chip's register map is currently undocumented.

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0x00 (Register Access Functional Group)
CMD3 = 0x01 (One address-data register pair)
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x06
Len_MSB = 0x00
Data[0] = 0xXX (LSB Address)
Data[1] = 0xXX (LSB+1 Address)
Data[2] = 0xXX (LSB Data)
Data[3] = 0xXX (LSB+1 Data)
Data[4] = 0xXX (LSB+2 Data)
Data[5] = 0xXX (LSB+3 Data)
Checksum = 0xXX (sum of everything after CMD4)

NOTE: To write to multiple registers within the same packet the SPI master needs to populate CMD3 to the number of address-data pairs desired and would append the subsequent address-data information to the Data portion of the packet. Since each address-data pair requires 6 bytes of storage, the maximum number of register writes that can fit in a single packet is 84.

WARNING

Sending register addresses with *unknown* data leads to corrupting DLP Control chip internal state machine and will stop responding.

7.2 LUT/Mailbox Access

The DLP Control chip contains an internal hardware mailbox. The LUT/Mailbox Access function is used to translate the mailbox related information from a configuration Batch File into command packets to be sent on the SPI bus.

Example: Below is a reference Batch File section

Portion of the *Batch File* containing mailbox information

```
<line 1>   $L2.5 DLP_RegIO_BeginLUTdata SEQ_LUT
<line 2>   $L2.5 WriteReg 0x1111 0x000000f8 # data byte 000000
<line 3>   $L2.5 WriteReg 0x1111 0x00000008 # data byte 000004
<line 4>   $L2.5 WriteReg 0x1111 0x00260005 # data byte 000008
<line 5>   $L2.5 WriteReg 0x1111 0x00080004 # data byte 00000c
<line 6>   $L2.5 WriteReg 0x1111 0x00080004 # data byte 000010
<line 7>   $L2.5 WriteReg 0x1111 0x00080004 # data byte 000014
<line 8>   $L2.5 WriteReg 0x1111 0x00080004 # data byte 000018
<line 9>   $L2.5 WriteReg 0x1111 0x00080004 # data byte 00001c
<line 10>  $L2.5 WriteReg 0x1111 0x00080004 # data byte 000020
<line 11>  $L2.5 WriteReg 0x1111 0x00080004 # data byte 000024
<line 12>  $L2.5 WriteReg 0x1111 0x00080004 # data byte 000028
<line 13>  $L2.5 WriteReg 0x1111 0x00080004 # data byte 00002c
<line 14>  $L2.5 WriteReg 0x1111 0x00080004 # data byte 000030
<line 15>  $L2.5 WriteReg 0x1111 0x00080004 # data byte 000034
<line 16>  $L2.5 WriteReg 0x1111 0x00080004 # data byte 000038
<line 17>  $L2.5 WriteReg 0x1111 0x00080004 # data byte 00003c
<line 18>  $L2.5 DLP_RegIO_EndLUTdata SEQ_LUT
```

As per the Batch File syntax and API documentation there are four LUT types

- 0x01 = RWC_LUT
- 0x02 = SEQ_LUT
- 0x06 = CMT_LUT
- 0x08 = UMCTDM_LUT

The LUT/Mailbox related portion in the Batch File can be identified with the DLP_RegIO_BeginLUTdata and DLP_RegIO_EndLUTdata terms. The portion between the Begin and End LUT contains the LUT data. The above LUT is translated into the SPI command packet as follows

```
PACKET STRUCTURE:
        CMD1 = 0x02 (WRITE)
        CMD2 = 0x03 (LUT/Mailbox access functional group)
        CMD3 = 0x10 (Number of LUT data entries in the mailbox. In the above example it is 16)
        CMD4 = 0x00 (only packet of transfer as ALL LUT entries can be put in single packet)
        Len_LSB = 0x41 (65 bytes = 1 byte LUT name + 16*4 bytes)
        Len_MSB = 0x00

        // LUT/Mailbox type, see above description LUT type lists
        Data[0] = 0x02 (SEQ_LUT)

        //1st data entry
        Data[1] = 0xF8 (LSB)
        Data[2] = 0x00 (LSB+1)
        Data[3] = 0x00 (LSB+2)
```

Data[4] = 0x00 (LSB+3)

//2nd data entry

Data[5] = 0x08 (LSB)

Data[6] = 0x00 (LSB+1)

Data[7] = 0x00 (LSB+2)

Data[8] = 0x00 (LSB+3)

.....

//16th data entry

Data[61] = 0x04 (LSB)

Data[62] = 0x00 (LSB+1)

Data[63] = 0x08 (LSB+2)

Data[64] = 0x00 (LSB+3)

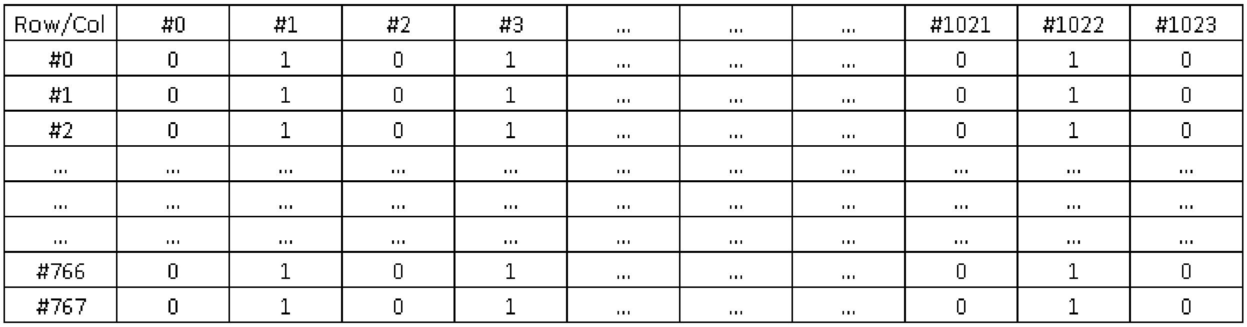
Checksum = 0xXX (sum of everything after CMD4)

NOTE: When the LUT size requires more than one packet to transfer then the LUT data entry starts from Data[0] position from the second packet onwards.

7.3 Full Image Download

The Full Image Download interface is used to download a full image pattern (XGA resolution of 1024x768) over SPI bus. Here, full image corresponds to 1 bit-per-pixel width. So, for a 8 bit-per-pixel image a full image download is repeated 8 times for a complete download.

*Example:*Suppose the user wants to download a 1 bit-per-pixel full image pattern (XGA resolution of 1024x768) of alternating 1 pixel wide vertical stripes over SPI. The structure of this pattern would look like the following:



Row/Col	#0	#1	#2	#3	...	...	...	#1021	#1022	#1023
#0	0	1	0	1	...	...	...	0	1	0
#1	0	1	0	1	...	...	...	0	1	0
#2	0	1	0	1	...	...	...	0	1	0
...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...
#766	0	1	0	1	...	...	...	0	1	0
#767	0	1	0	1	...	...	...	0	1	0

Figure 4. Alternating Black and White 1 Bit-per-pixel Image

The number of packets needed to perform the full download is 196:

- The total number of pixels in the image is 1024*768 = 786432.
- Due to the 10-bit *Memory index* and the requirement that the number of bytes with pixel data be a multiple of 4, the first packet has 500 bytes available for pixel data. With 8 pixels per byte, the first packet can store values for 500*8 = 4000 pixels.
- All remaining packets have 502 bytes available for pixel data. With 8 pixels per byte, these packets can store values for 504*8 = 4032 pixels.
- 1*4000 + 194*4032 = 786208. Therefore, 195 packets not enough storage.
- 1*4000 + 195*4032 = 790240. Therefore, 196 packets is enough storage.

The image content is converted into packets as follows:

PACKET-1 STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0x04 (Full Image Pattern Download)
CMD3 = 0x00 (RESERVED should always be set to 0x00)
CMD4 = 0x01 (First of many packets of transfer)
Len_LSB = 0xF6
Len_MSB = 0x01

Data[0] = 0xE3 (LSB External Memory Index, valid range: 0 to 959)
Data[1] = 0x00 (LSB+1)

Data[2] = 0x55 (Pixel values for line 0, pixels 0-7)
Data[3] = 0x55 (Pixel values for line 0, pixels 8-15)
Data[4] = 0x55 (Pixel values for line 0, pixels 16-23)
Data[5] = 0x55 (Pixel values for line 0, pixels 17-31)
...
Data[501] = 0x55 (Pixel values for line 3, pixel 920-927)

Checksum = 0xXX (sum of everything after CMD4)

PACKET-2 STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0x04 (Full Image Pattern Download)
CMD3 = 0x00 (RESERVED should always be set to 0x00)
CMD4 = 0x02 (Middle of many packets of transfer)
Len_LSB = 0xF8
Len_MSB = 0x01

Data[0] = 0x55 (Pixel values for line 3, pixels 928-935)
Data[1] = 0x55 (Pixel values for line 3, pixels 936-943)
Data[2] = 0x55 (Pixel values for line 3, pixels 944-951)
Data[3] = 0x55 (Pixel values for line 3, pixels 952-959)
...
Data[502] = 0x55 (Pixel values for line 7, pixel 848 to 855)
Data[503] = 0x55 (Pixel values for line 7, pixel 856 to 863)

Checksum = 0xXX (sum of everything after CMD4)

Packets 3-195 follow same structure as Packet 2.

PACKET-196 STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0x04 (Full Image Pattern Download)
CMD3 = 0x00 (RESERVED should always be set to 0x00)
CMD4 = 0x04 (last of many packets of transfer)
Len_LSB = 0x1C
Len_MSB = 0x00

Data[0] = 0x55 (Pixel values for line 767, pixels 800 to 807)
Data[1] = 0x55 (Pixel values for line 767, pixels 808 to 815)
...
Data[26] = 0x55 (Pixel values for line 767, pixel 1008-1015)
Data[27] = 0x55 (Pixel values for line 767, pixel 1016-1023)

Checksum = 0xXX (sum of everything after CMD4)

Upon receipt of the last packet, the DLPC200 will send a response back to the master. If the command was received and accepted, Data[0] = 0x00 and Data[1] = 0x00 in the command response. The response packet will have the following structure:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x03 (WRITE_RESP)
CMD2 = 0x06 (Function Group name)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x08
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)

// zero value bytes
Data[2] = 0x00

Data[3] = 0x00

// No. of packets received by controller, 32bit value
Data[4] = 0xC4 (LSB)
Data[5] = 0x00 (LSB+1)
Data[6] = 0x00 (LSB+2)
Data[7] = 0x00 (LSB+3)

Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 7 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

NOTE: *WriteImageOrderLUT*sets how the images are to be displayed on the DMD. When using *Full Image Download* interface for downloading the image it is important to keep track of *Memory Index* information of image being downloaded. This information helps in calling *WriteImageOrderLUT* with valid values.

NOTE: By default when 1 bit-per-pixel configuration is used the images are stored with *Memory Index* order 0,1,2,....and so forth.

NOTE: By default when 8 bit-per-pixel configuration is used each 8 bit image allocated into consecutive memory locations as follows:

- First 8 bit image will be loaded into *Memory Index* order 0,1, 2,....7 where 0th location occupied with bi #0 of 8 bit image and 7th location occupied with bit #7 of 8 bit image.
- Similarly, second 8 bit image will be loaded into *Memory Index* order 8, 9,....15.
- Repeated as necessary for the number of 8 bit images used in the configuration.

7.4 Flash Download

The Flash Update interface is used to update the flash devices (Serial Flash and Parallel Flash) connected to DLPC200 over SPI bus. On the Serial Flash the DLPC200 firmware is stored. On the Parallel Flash user 'configuration' or Solution is stored. The packet contains the information indicating which flash device it needs to be written to. The first packet contains a four byte flash offset address location from where the flash update starts. The command packet structure allows programming 256 bytes per packet.

7.4.1 Serial Flash Download

The typical Serial Flash firmware update binary file size if fixed to 5.0MB or 5242880 byte. It requires a total of 5242880/256 = 20480 number of packets to complete the download. The Serial Flash (DLPC200 firmware) binary content is converted into packets as follows:

```
PACKET-1 STRUCTURE:
        CMD1 = 0x02 (WRITE)
        CMD2 = 0x06 (Flash Update)
        CMD3 = 0x01 (RESERVED should always be set to 0x01 for Serial Flash update)
        CMD4 = 0x01 (First of many packets of transfer)
        Len_LSB = 0x04
        Len_MSB = 0x01
        // Serial Flash Offset Address = 0x00300000
        Data[0] = 0x00 (LSB)
        Data[1] = 0x00 (LSB+1 Flash Offset)
        Data[2] = 0x30 (LSB+2 Flash Offset)
        Data[3] = 0x00 (LSB+3 Flash Offset)
        Data[4] = Byte0 (Flash data byte0)
        Data[5] = Byte1 (Flash data byte1)
        ...
        Data[259] = Byte255 (Flash data byte255)
        Checksum = 0xXX (sum of everything after CMD4)

PACKET-2 STRUCTURE:
        CMD1 = 0x02 (WRITE)
        CMD2 = 0x06 (Flash Update)
        CMD3 = 0x01 (RESERVED should always be set to 0x01 for Serial Flash update)
        CMD4 = 0x02 (Middle of many packets of transfer)
        Len_LSB = 0x00
        Len_MSB = 0x01
        Data[0] = Byte256 (Flash data byte256)
        Data[1] = Byte257 (Flash data byte257)
        Data[2] = Byte258 (Flash data byte258)
        Data[3] = Byte259 (Flash data byte259)
        ...
        Data[254] = Byte510(Flash data byte510)
        Data[255] = Byte511 (Flash data byte511)
        Checksum = 0xXX (sum of everything after CMD4)
```

All intermediate Packets #3 to #20479 follow same structure as Packet 2.

PACKET-20480 LAST STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0x06 (Flash Update)
CMD3 = 0x01 (RESERVED should always be set to 0x01 for Serial Flash update)
CMD4 = 0x04 (last of many packets of transfer)
Len_LSB = 0x00
Len_MSB = 0x01
Data[0] = ByteXX (Flash data byteXX)
Data[1] = ByteXX (Flash data byteXX+1)
...
Data[254] = Byte 5242878 (Flash data byte5242878)
Data[255] = Byte5242879 (Flash data byte5242879)
Checksum = 0xXX (sum of everything after CMD4)

Upon receipt of the last packet, the DLPC200 will send a response back to the master. If the command was received and accepted, Data[0] = 0x00 and Data[1] = 0x00 in the command response. The response packet will have the following structure:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x03 (WRITE_RESP)
CMD2 = 0x06 (Function Group name)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x08
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)

// CRC16 Checksum of written memory contents
Data[2] = 0xXX (LSB)
Data[3] = 0xXX (LSB+1)

// No. of packets received by controller, 32bit value
Data[4] = 0xXX (LSB)
Data[5] = 0xXX (LSB+1)
Data[6] = 0xXX (LSB+2)
Data[7] = 0xXX (LSB+3)

Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 7 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

NOTE: Before calling Flash Update it is important to Erase Flash using Flash Erase command.

NOTE: This interface is intended for firmware upgrades in the field. This interface will work only when the Serial Flash has a valid firmware previously loaded and the DLPC200 is able to boot.

NOTE: For blank flash or corrupt flash, there is a different firmware file available that can be programmed via a standard flash programmer tool. The firmware size for a blank or corrupted flash is 8 MB.

7.4.2 Parallel (User Configuration) Flash Download

Parallel Flash stores user configuration data. The User Configuration binary file is built *offline* using DLP LightCommander GUI control software. The size of the user configuration varies depending on the number of *Solutions* in the *Project* and the number of images loaded with a *Non-Real Time Structured Light* solution:

```
PACKET-1 STRUCTURE:
    CMD1 = 0x02 (WRITE)
    CMD2 = 0x06 (Flash Update)
    CMD3 = 0x00 (RESERVED should always be set to 0x00 for Parallel Flash update)
    CMD4 = 0x01 (First of many packets of transfer)
    Len_LSB = 0x04
    Len_MSB = 0x01
    // Flash offset address 32bit value
    Data[0] = 0xXX (LSB)
    Data[1] = 0xXX(LSB+1)
    Data[2] = 0xXX (LSB+2)
    Data[3] = 0xXX (LSB+3)
    Data[4] = Byte0 (Flash data byte0)
    Data[5] = Byte1 (Flash data byte1)
    ...
    Data[259] = Byte255 (Flash data byte255)
    Checksum = 0xXX (sum of everything after CMD4)

PACKET-2 STRUCTURE:
    CMD1 = 0x02 (WRITE)
    CMD2 = 0x06 (Flash Update)
    CMD3 = 0x00 (RESERVED should always be set to 0x00 for Parallel Flash update)
    CMD4 = 0x02 (Middle of many packets of transfer)
    Len_LSB = 0x00
    Len_MSB = 0x01
    Data[0] = Byte256 (Flash data byte256)
    Data[1] = Byte257 (Flash data byte257)
    Data[2] = Byte258 (Flash data byte258)
    Data[3] = Byte259 (Flash data byte259)
    ...
    Data[254] = Byte510(Flash data byte510)
    Data[255] = Byte511 (Flash data byte511)
    Checksum = 0xXX (sum of everything after CMD4)
```

All intermediate Packets #3 to #(N-1)th follow same structure as Packet 2.

PACKET-Nth or LAST STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0x06 (Flash Update)
CMD3 = 0x00 (RESERVED should always be set to 0x00 for Parallel Flash update)
CMD4 = 0x04 (last of many packets of transfer)
Len_LSB = 0x00
Len_MSB = 0x01
Data[0] = ByteXX (Flash data byteXX)
Data[1] = ByteXX (Flash data byteXX+1)
...
Data[254] = ByteXX or 0xFF (Flash data byte(Last-1) or 0xFF padding)
Data[255] = ByteXX or 0xFF (Flash data byte(Last) or 0xFF padding)
Checksum = 0xXX (sum of everything after CMD4)

Upon receipt of the last packet, the DLPC200 will send a response back to the master. If the command was received and accepted, Data[0] = 0x00 and Data[1] = 0x00 in the command response. The response packet will have the following structure:

RESPONSE PACKET STRUCTURE:

CMD1 = 0x03 (WRITE_RESP)
CMD2 = 0x06 (Function Group name)
CMD3 = 0x00
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x08
Len_MSB = 0x00
Data[0] = 0x00 (request processed successfully)
Data[1] = 0x00 (request processed successfully)

// CRC16 Checksum of written memory contents
Data[2] = 0xXX (LSB)
Data[3] = 0xXX (LSB+1)

// No. of packets received by controller, 32bit value
Data[4] = 0xXX (LSB)
Data[5] = 0xXX (LSB+1)
Data[6] = 0xXX (LSB+2)
Data[7] = 0xXX (LSB+3)

Checksum = 0xXX (sum of everything after CMD4)

If the command packet was corrupted in some way then the embedded application would set bits in Data[0] and Data[1] in the response packet to signify this error, see Section 7 for details. If that was the case, the response packet might only include Data[0] and Data[1] depending on which part of the command packet the error was located.

NOTE: If the number of bytes in the *Last packet* < 256 then Byte padding with 0xFF must be done in the data packet to make the packet data size = 256.

7.5 Flash Erase

This command packet supports *Erasing* of Serial and Parallel flash. The command packet requires a Begin and End offset address for the region of the flash to be erased.

7.5.1 Parallel Flash Erase

Parallel Flash Erase packet structure looks like the following:

```
 PACKET STRUCTURE:
         CMD1 = 0x02 (WRITE)
         CMD2 = 0x07 (Flash Erase functional group)
         CMD3 = 0x10 (Must be set to 0x10 for Parallel Flash Erase)
         CMD4 = 0x00 (only packet of transfer)
         Len_LSB = 0x08
         Len_MSB = 0x00
         Data[0] = 0xXX (LSB) (Erase Begin Address)
         Data[1] = 0xXX (LSB+1)
         Data[2] = 0xXX (LSB+2)
         Data[3] = 0xXX (LSB+3)
         Data[4] = 0xXX (LSB) (Erase End Address)
         Data[5] = 0xXX (LSB+1)
         Data[6] = 0xXX (LSB+2)
         Data[7] = 0xXX (LSB+3)
         Checksum = 0xXX (sum of everything after CMD4)
```

NOTE: Depending upon the flash device specification and the requested size/region to be erased, the command response time varies. The SPI master needs to wait until the flash is erase or Poll SLAVE_SPI_ACK pin before excepting the response from DLP Control chip. It is suggested to erase the flash from Begin offset = 0x00000000 and setting End Address offset from the user configuration binary file size information that is generated via DLP LightCommander GUI.

7.5.2 Serial Flash Erase

The Serial Flash or DLPC200 controller firmware size is always fixed. So, the Erase Begin and End Address are fixed. Serial Flash Erase packet structure looks like the following:

```
 PACKET STRUCTURE:
         CMD1 = 0x02 (WRITE)
         CMD2 = 0x07 (Flash Erase functional group)
         CMD3 = 0x11 (Must be set to 0x11 for Parallel Flash Erase)
         CMD4 = 0x00 (only packet of transfer)
         Len_LSB = 0x08
         Len_MSB = 0x00
         //Flash erase begin address 32bit value
         Data[0] = 0x00 (LSB)
         Data[1] = 0x00 (LSB+1)
         Data[2] = 0x30 (LSB+2)
         Data[3] = 0x00 (LSB+3)
         //Flash erase end address 32bit value
```

Data[4] = 0xFF (LSB)

Data[5] = 0xFF (LSB+1)

Data[6] = 0x7F (LSB+2)

Data[7] = 0x00 (LSB+3)

Checksum = 0xB5 (sum of everything after CMD4)

7.6 EDID Update

DLP Controller chip support updating the EEPROM to store EDID information. This commands packet is used to send the 128 bytes EDID info to be programmed in the EEPROM. The command packet structure looks like the following:

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0x08 (EDID update functional group)
CMD3 = 0x00 (RESERVED has to be set to 0x00)
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x06
Len_MSB = 0x00
Data[0] = 0x39 (Should always be set for DLP5500 DMD)
Data[1] = 0xXX (Offset location from where to start update) valid range <0 - 127>
Data[2] = 0xYY (Number of bytes to update from the <offset> location)
Data[3] = 0xXX (Data 0)
Data[4] = 0xXX (Data 1)
Data[5] = 0xXX (Data 2)
...
Data[N-1] = 0xXX (Data n-1)
Data[N] = 0xXX (Data n) where 'n' + 1 = Data[2]
Checksum = 0xXX (sum of everything after CMD4)

NOTE: It is preferred to update entire EDID content (that is, from the offset location <0> to <128>) length of data bytes in the payload.

7.7 DLPC200 Reset

This is a special command packet which will cause a *reset* of DLPC200. The command packet structure looks like the following:

PACKET STRUCTURE:

CMD1 = 0x02 (WRITE)
CMD2 = 0x00
CMD3 = 0x01
CMD4 = 0x00 (only packet of transfer)
Len_LSB = 0x06
Len_MSB = 0x00
Data[0] = 0x80
Data[1] = 0x04
Data[2] = 0x4A
Data[3] = 0x00
Data[4] = 0x00
Data[5] = 0x00
Checksum = 0xD4 (sum of everything after CMD4)

NOTE: When Parallel Flash is not used in the reference design or Parallel Flash is Empty/Corrupt, sending this command will create a Black-White Checker Board Test Pattern after the reset completion.

NOTE: The controller will start the reset process as soon as it decodes this command, meaning the controller will not issue a command response (even after it is done rebooting).

Revision History

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

Changes from B Revision (November 2014) to C Revision **Page**

- Changed maximum byte array size from 512 to 511 bytes 2
- Changed wording to clarify the response is read form the slave 3
- Changed Figure 2 to increase readability of text and maintain uniformity with Figure 3 4
- Added information on how to read the response packet, using a predetermined amount of bytes or dynamically reading the response length 5
- Updated Figure 3 on how to dynamically read the length of a response packet 6
- Changed 512-byte extended packet to 511-byte extended packet 7
- Changed 0-505 to 0-503 to match 511-byte structure 7
- Changed bullet content into a sentence, since it was only one item 7
- Added a tabled note explaining the GRAY tables entries, previously a bullet item above the table 7
- Added a clarifying note that the response of query commands have different structures 8
- Changed the comments in the command structure to promote uniformity between packet structures 10
- Added response packet header 11
- Added an explanation that the length of the response packet might be different if it had any errors 11
- Deleted the sentences that explained the WRITE command response packet since it was already explained in Section 6 16
- Deleted Data[3] and Data[4] from the packet definition of GetLEDintensity since they were not part of the command structure 17
- Added the response packet definition for GetLEDintensity(READ) command 17
- Changed extended packet ID for LEDdriverEnable command in its packet definition from 0x0A to the correct 0x0B 18

- Added a clarifying note explaining that enabling and LED using SetLEDEnable command will not cause the LED to turn on if the current solution dose not have a PWM sequence built for it 18
- Changed the number of patterns from 250 to 249 to match the 511-byte extended packet size limit 19
- Changed the comments in the command structure for WriteImageOrderLut, to promote uniformity between packet structures and clarity 19
- Changed the 250th image to 249th image to match the 511-byte extended packet size limit 19
- Changed Data[503] and Data[504] to Data[501] and Data[502] to match the 511-byte extended packet size 19
- Changed Data[503] and Data[504] to Data[501] and Data[502] to match the 510-byte extended packet size 20
- Changed the data index value from 505, which exceeded the maximum packet size, to 502 in second packet description of WriteImageOrderLut 20
- Changed Data[503] and Data[504] to Data[501] and Data[502] to match the 510-byte extended packet size 21
- Changed the data index value from 505, which exceeded the maximum packet size, to 502 in second packet description of WriteImageOrderLut 21
- Changed Data[503] and Data[504] to Data[501] and Data[502] to match the 510-byte extended packet size 21
- Changed the data index value from 505, which exceeded the maximum packet size, to 502 in third packet description of WriteImageOrderLut 21
- Added the response packet structure for a multi-packet WriteImageOrderLut command 22
- Changed the data source option to be part of the packet structure definition 24
- Moved comment to the command description, previously in the packet structure 25
- Changed sentence wording from "data - structured light exposure percentage" to "data for structure light exposure", percentage wording was misleading 38
- Deleted "For example the Red LED Driver." sentence, to keep packet structure definition uniform 44
- Changed the comments in the command structure, to promote uniformity between packet structures 44
- Changed packet length from 0x02 to the correct length of 0x03 in GetLEDdriverLitState packet description 44
- Changed packet length from 0x02 to the correct length of 0x03 in GetLEDdriverTempTimeoutState packet description 46
- Changed extended packet ID from 0x2C to the correct ID value of 0x2D in GetLEDdriverTempTimeoutState packet description 46
- Changed packet length from 0x02 to the correct length of 0x03 in GetLEDdriverStrobeTimeoutState packet description 48
- Changed extended packet ID from 0x2C to the correct ID value of 0x2F in GetLEDdriverStrobeTimeoutState packet description 48
- Changed from "0x04 (READ)" to correct value of "0x02 (WRITE)" in DownloadBPPfromFlashToExtMem packet description 48
- Changed the range of N from "N≤49" to a more accurate "2<N≤50" in DownloadBPPfromFlashToExtMem packet description 49
- Changed sentence wording form "loaded without" to "loaded with or without" 50
- Changed Data index values to "2, 3, 4, 5, 6" from "4, 5, 6, 7, 8" which skipped indexes 2 and 3 in LoadSolutionFromFlash packet description 51
- Changed 512-byte extended packet to 510-byte extended packet 57
- Added a sentence clarifying that the controllers register map is undocumented 59
- Deleted references to the unsupported LightCommander GUI software 59
- Changed the packet structure definition, so that it does not follow a LightCommander GUI example 59
- Deleted references to the unsupported LightCommander GUI software 59
- Changed command description to remove references to DLP LightCommander® 60
- Corrected the length of the last packet for FullImageDownload command 63
- Changed the comments to specify what pixels of the image are being modified in FullImageDownload command 63
- Changed the data index of the last packet of FullImageDownload command from 223 to 27 63
- Added the response packet structure for a multi-packet FullImageDownload command 63
- Added the response packet structure for a multi-packet SerialFlashDownload command 66
- Added the response packet structure for a multi-packet ParallelFlashDownload command 68
- Added a clarifying note that the controller will not respond to a DLPC200Reset command 72

Changes from A Revision (August 2012) to B Revision — Page

- Added instructions for how to find the solution address needed in the LoadSolutionFromFlash packet 50
- Added calculation of necessary delay time for DisplayPatternAutoStepForSinglePass packet 52

Changes from Original (October 2011) to A Revision **Page**

- Overview section is updated with SPI byte transfer timing diagram 1
- Command/Response protocol explained in other section 3
- New Extended Packet supported - GenerateSWVsync 53
- New Extended Packet supported - ConfigurePWMPeriod 53
- New Extended Packet supported - ConfigurePWMDutyCycle 55
- Low Level Packet control definitions added 57

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated (‘TI”) technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, “TI Resources”) are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI’s provision of TI Resources does not expand or otherwise alter TI’s applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED “AS IS” AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI’s standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265