

OptiLamp™ ASIC Communication String and Programming Guidance

INTRODUCTION

The OptiLamp product family is a series of full color RGB LED packages that merge an integrated ASIC driver into a discrete LED component package geared towards display application. Designed for video wall, signage, and next generation display systems, it delivers high brightness, low power, smooth grayscale, with high dynamic range. The advanced OptiLamp driver technology replaces external driver circuitry, dramatically simplifying PCB design while enabling high speed bidirectional data communication and monitoring—including chip level temperature sensing, voltage measurement, leakage detection, and CRC based error protection.

The OptiLamp ASIC is built on a digital control logic to handle data from a video source (e.g. HDMI receiver) routed through an FPGA for string routing and commands. OptiLamp employs a simple, non-standard return-to-zero (RZ) line code, and each package is designed as a pseudo-repeater with bidirectional communication ports to allow for cascade connection in various serpentine package orientations.

The details in this application note provide detailed information and guidance for navigating the programming and communication data string for the OptiLamp ASIC and the OptiLamp family of components. These details are necessary for custom programming and for receiver/controller interfacing beyond that of built-in environments (e.g. ColorLight, Brompton, Novastar, etc.).

TABLE OF CONTENTS

1. Description	3
1.1 ARCHITECTURE.....	3
1.1.1 Control Logic.....	3
1.1.2 DIO Buffers.....	3
1.1.3 Analog Block.....	4
1.2 EXAMPLE APPLICATION.....	4
2. Communication.....	6
2.1 LINE CODE SPECIFICATION.....	6
2.2 COMMUNICATION PROTOCOL.....	6
2.2.1 START FRAME BYTES	7
2.2.2 DATA STREAM SEQUENCE.....	8
3. Command reference	9
3.1 COMMAND BYTE.....	9
3.2 COMMAND SET.....	10
3.2.1 EOF.....	11
3.2.2 Set Options.....	11
3.2.3 Write Memory.....	16
3.2.4 Write Events.....	17
3.2.5 Set Brightness; Return Data.....	17
3.2.6 SS Brightness.....	17
3.2.7 Skip.....	18
3.2.8 SS Skip.....	18
3.2.9 Reset.....	18
3.2.10 Measure Vf.....	18
3.2.11 Measure Vr.....	19
3.2.12 Measure T.....	19
3.3 CRC.....	19
3.4 Return Data.....	20
3.5 Error Reporting.....	20
3.6 Calibration and Compensation.....	22
3.6.1 Gamma/Decompression Curves.....	23
3.6.2 Decode.....	24
3.6.3 Mapping with Both Decompression Curves and Decode – OLY6D-FKC Examples.....	26
3.6.4 Thermal Compensation Curves.....	27
3.6.5 Curve Transformation.....	28
3.7 Timing.....	28
3.7.1 Events: Blanking and Flash.....	28
3.7.2 PWM Phase.....	29
4. EXAMPLE.....	30
5. ACRONYMS	32
6. Closing Remarks	32

1. DESCRIPTION

1.1 ARCHITECTURE

Figure 1 shows a block diagram of an OptiLamp package. Each of the primary blocks are briefly introduced below.

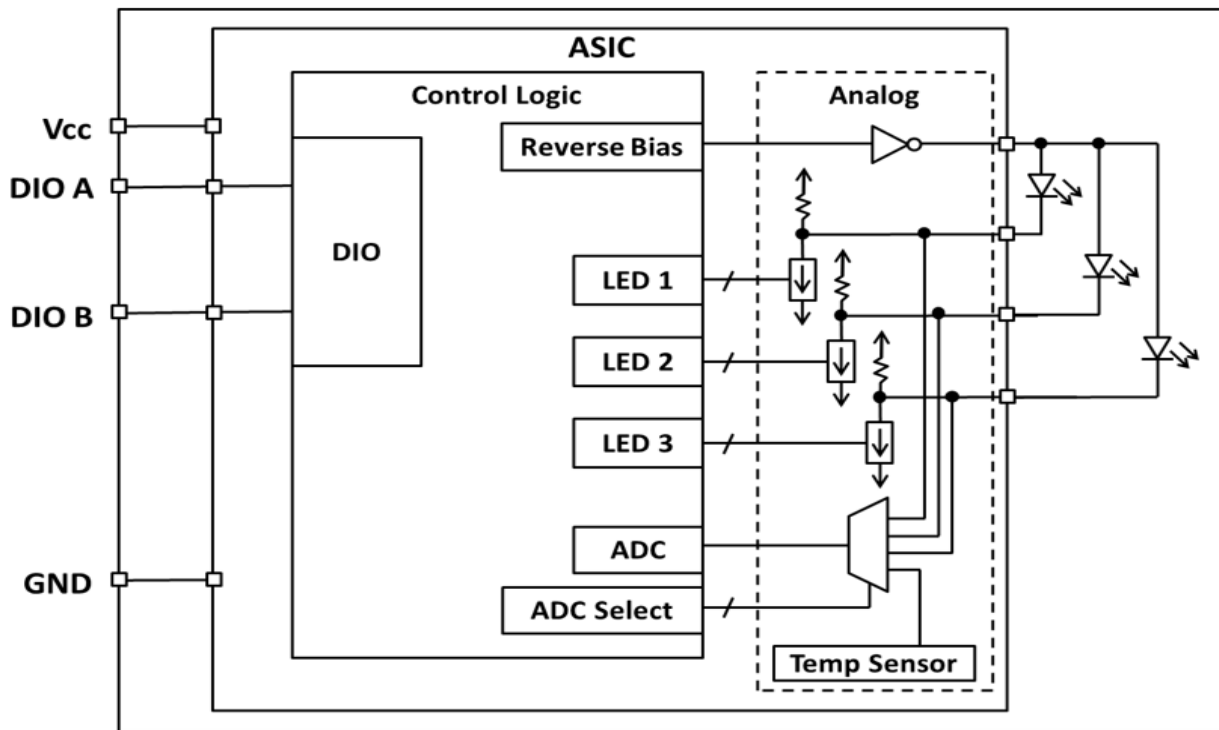


Figure 1. Block Diagram

1.1.1 Control Logic

The control logic routes incoming and outgoing data to and from the appropriate DIO ports, executes the commands according to the command protocol, provides control signals to the LED Drivers, and reports various voltage levels of the LED cathode nodes and the temperature sensor.

1.1.2 DIO Buffers

Two bidirectional ports, DIO A-B, follow LVCMOS33 standards and are compatible with many FPGA devices. Communication speed is determined by the customer, typically within 2-4 Mbps. 3 Mbps is recommended.

1.1.3 Analog Block

The analog block provides the LED driver with the ability to reverse-bias the LEDs, select the LED current, sense temperature and voltage levels, and drive the grey levels of the LEDs. Elements of the Analog block are described as follows:

LED Sink Driver. Fig 1 shows the LED sink driver as a controllable current source and a pull-up resistor. Although the grey levels of the LEDs are primarily controlled by PWM, current is also used, and four current ranges are selectable. The pull-up resistor provides a high resistance path to Vcc to provide reverse current detection in one of the health monitoring modes (reverse leakage measurement).

LED Source Driver. The LED source driver (inverted source to LED anodes) provides forward and reverse operation of the LEDs for LED operation and health monitoring.

Temperature Sensor. An on-chip temperature sensor voltage is provided through the 4x1 multiplexer and 16-bit ADC. The temperature measurement may be used for thermal compensation of the LEDs and may also be reported back to the controller. Thermal shut down the device is achieved via the thermal compensation curve. Specifications for the temperature sensor are contained in the table below:

Symbol	Parameter	Condition	Min	Typ	Max	Units
	Temperature Range		-40		150	°C
	Temperature Accuracy	T _j = -25 °C to 125 °C	-10		10	°C
V _{TC}	TC Voltage Output	-40 °C to 150 °C	910	788	560	mV
a _{TC}	TC Voltage Output Slope	-40 °C to 150 °C		-1.76		mV/°C

1.2 EXAMPLE APPLICATION

OptiLamp packages are typically used for LED video screens and displays. Figure 2 illustrates a simple system with minimal hardware. A standard HDMI receiver chip may be used to provide the data for the screen. An FPGA can be used to route the data to the appropriate strings and provide commands and other control signals to the pixel components. The components may be assembled on a PCB in a 2-dimensional matrix with each LED component being one pixel of an LED display. OptiLamp is equipped with bidirectional communication ports. As such, either port may be used for the input or output. This simplifies the PCB layout as the cascade communication can be laid out in a simple serpentine manner as illustrated in figure 2.

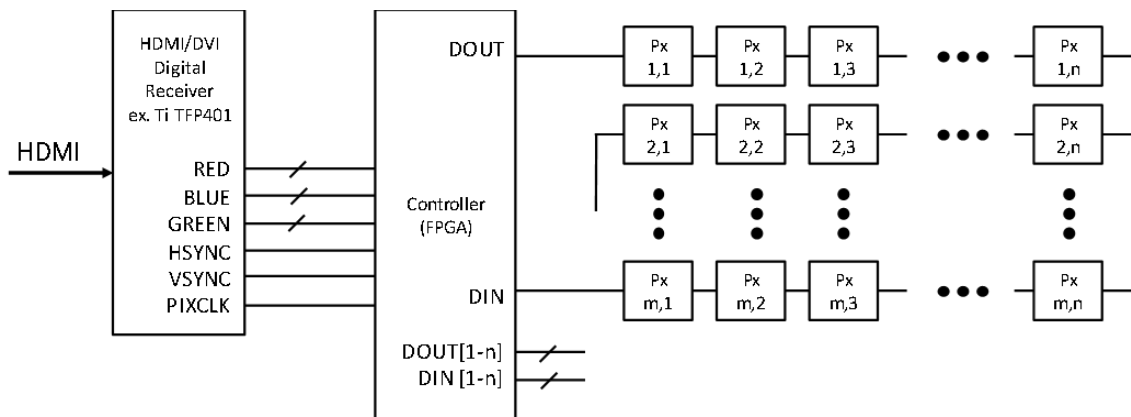


Figure 2. Typical Application

Each component behaves as a pseudo-repeater. When the target component receives a command intended for its location, it executes the command and retransmits altered data in the same time slot with a 3-bit-cycle delay. Data not intended for the component is retransmitted unchanged so that the appropriate component may receive the data. If the end of the communication string is connected back to the controller, all the changed and unchanged data may be analyzed by the controller for signal integrity and health monitoring. The command protocol allows the controller to send and receive data to any component or send data to all components at once (see command reference section).

For video displays, the maximum frame rate depends on several factors. If all components are updated with a new color for each frame, the maximum frame rate is:

$$\text{Frame Rate} = \frac{\text{Bit Rate}}{(\text{Num Color Bits} * \text{NumPx} + 40)}$$

Where 40 is the number of bits required for the command byte with header information (see command reference section), *Num Color Bits* is the number of bits for RGB color (such as 24-bit true color), and *NumPx* is the number of components in a single communication string. One FPGA may control many strings as part of a module, tile, or in some cases, a whole LED display. Typically for large LED screens, a master control unit is used to divide the Video signal into parts respective for each module, tile, or cabinet; and the data conveyed to the array of cabinets via a communication network. Depending on the design, each PCB may use a small FPGA to control strings on the respective PCB. At a bit rate of about 3 Mbps, one FPGA output can control about 2k pixels on one string at 60 FPS and one small FPGA with multiple DIO ports can be used to control a large display tile with 100k pixels or more.

2. COMMUNICATION

2.1 LINE CODE SPECIFICATION

OptiLamp uses a simple, non-standard return-to-zero (RZ) line code designed specifically for reliable clock-less communication in a repeater-intense network. Figure 4 shows that the zero bit (0) is represented by a positive pulse of relatively short duration and the one bit (1) is represented by a positive pulse of relatively long duration.

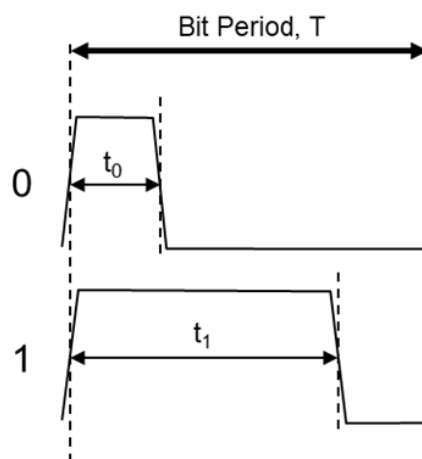


Figure 4. Bit Pattern

Line Code Specifications

Symbol	Parameter	Min	Typ	Max	Units
T	Communication period				us
t_0/T	Ratio of "Zero" pulse width to communication period		0.25		
t_1/T	Ratio of "One" pulse width to communication period		0.75		

Each component samples the incoming signal with its own internal oscillator frequency which is not tightly controlled. Alternative pulse widths and periods may be acceptable, but the above specification is recommended for optimum performance.

2.2 COMMUNICATION PROTOCOL

OptiLamp packages are designed as a pseudo-repeater to be connected in cascade as shown in figure 2. Each component receives serial data from the stream, determines the appropriate action, and retransmits either the same or different data within the same time slot as shown in figure 5(a). Specifically, the retransmitted commands and data are delayed by 3 bits from the received data stream as shown in figure 5(b).

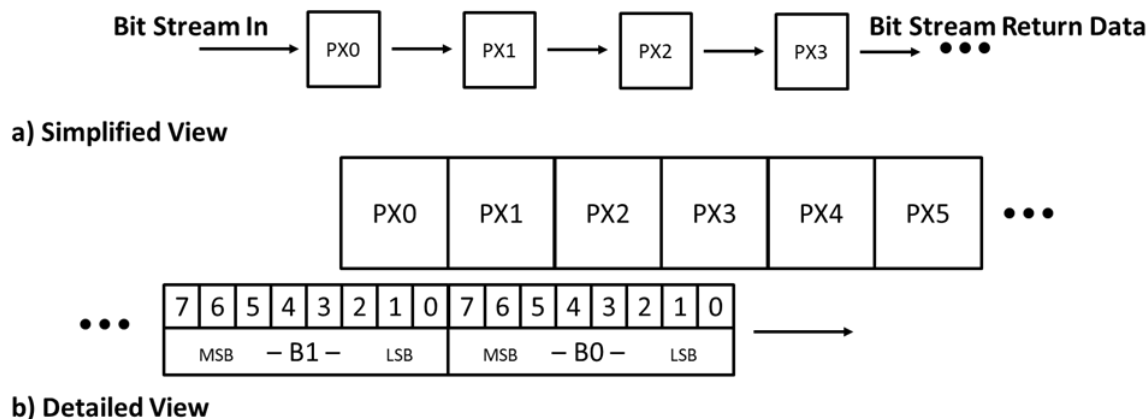


Figure 5. Data Stream

All communication is accomplished through a series of data words where each data word (command or data) is an 8-bit byte. The least significant bit, LSB, is sent first. Each byte may be immediately followed by another byte which could be supporting data for a command, another command, trailing zeros, or start frame bytes. The communication is conveyed via 1 wire (referenced to GND) and is clockless (clock is recovered from the bit stream). To start communication, at least two zero non-data zero bits must be transmitted, and another 8 bits are required to be sure the pixels are searching for an SF byte (described below), but it is recommended to start communication with at least two bytes of zeros for convenience. When transmission is to be terminated, additional zero bits must be transmitted to provide the communication clock and push the data through the entire chain as:

$$\text{Minimum Trailing Zeros} = 3 * (\text{NumPx} - 1)$$

These trailing zeros are necessary because of the 3-bit delay from pixel to pixel. If communication stops at the first pixel, all data within the string will also stop and be lost. To handle time delay such as is usual between video frames, the system designer has a choice to either stop communication after the minimum trailing zeros are transmitted, or to pad such delay times with zeros. The minimum trailing zero requirement does not apply when using zero padding since the communication is not being terminated and either padded zeros or new commands and data will push the prior data through the string.

2.2.1 START FRAME BYTES

The start frame, SF, byte has no function other than to guarantee alignment of the data and provide a means by which communication can be reestablished in the event that errors occur. The SF byte is defined as D5h (11010101), which is a pattern commonly used for ethernet communication and is commonly referred to as the start frame delimiter (SFD). Communication must be initiated by at least ten zero bits followed by four SF bytes. Commands and data should immediately follow.

The naming "Start Frame" is a carryover from the ethernet communication as it marks the beginning of a frame of data. It is important to realize that it is not necessarily aligned with the start of a video frame and has little connection to the end of frame (EOF) command. Start Frame aligns the data to follow, then does not need to be repeated as long as command/data communication continues. But, since most systems will have extra time remaining between video frames and during that time a long chain of zeros will be transmitted or communication stopped altogether, most designs will need the SF sequence at the beginning of each frame of data for realignment. So while the SF sequence doesn't necessarily relate to a video frame, most designs may use it at the beginning of the data for each video frame.

2.2.2 DATA STREAM SEQUENCE

The command protocol allows for communication between the controller and each component/pixel individually or the entire string of components at once. Figure 6 shows a flow chart of the sequence the controller should follow to properly communicate with a string of components. Communication must start with at least 10 zero bits (two to start communication and 8 more to be sure the components are searching for an SF byte). After the string of zeros four SF bytes are transmitted followed by a command byte. It is recommended that the first command be a "Reset" command which needs 1 byte of zeros to follow. The reset necessitates another four SF bytes to continue. The byte following the four SF bytes must always be a command. If a command requires data, the number of data bytes associated with that command is transmitted immediately following the command. The number of data bytes transmitted following a command must be correct, otherwise data gets interpreted as commands and vice versa resulting in an undetermined response. As shown in figure 6, communication continues with commands and corresponding data.

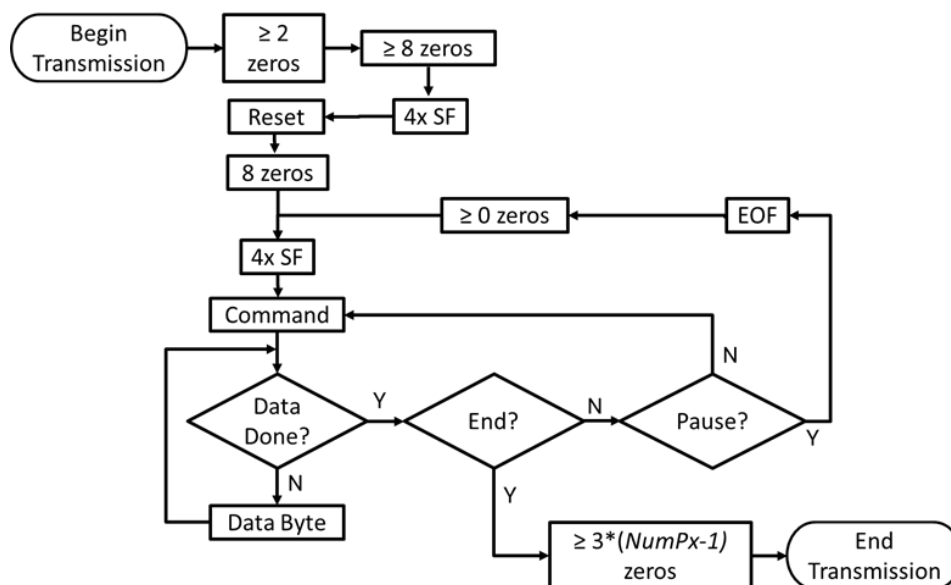


Figure 6. Data Stream Flow Chart

After a series of commands communication can either be stopped entirely or paused by transmitting zeros at the same communication frequency. Before stopping communication entirely, a string of $3 \times (\text{NumPx} - 1)$ zero bits must be transmitted to push the last portion of data through all the components. Then communication may be stopped entirely. If additional commands are expected to be sent soon (such as for the next video frame), it is generally desirable to keep communication active with any number of zeros. When it is time to send more commands and data, communication can continue by transmitting four SF bytes to align the data again.

3. COMMAND REFERENCE

3.1 COMMAND BYTE

Each bit of command byte has a purpose as summarized in the command byte elements table.

Bit	Function	Description
0	String (all)	0 - single-component command, 1 - all-component command
1	Not Executed	1 - command yet to be executed (applies to single-component commands)
2	Instruction 0	Instruction code
3	Instruction 1	
4	Instruction 2	
5	Instruction 3	
6	Error	Bit set on next single-pixel command if error occurs
7	Parity	Parity Check

The *string* bit indicates that the command is to be executed by all the components on the data stream. String commands only need to be sent once, and all components will execute the command. If the string bit is not set, it is a single-component command (aka non-string or pixel command) and will only be executed by the first component that receives it having its EOF state set. Commands such as EOF always apply to the whole string and therefore ignore this bit.

The not executed bit is used to address any individual component through single-component commands. It is not used for string commands. It indicates that the single-component command has not yet been executed by another component and should be executed by the first component that is in the EOF state to receive it. The first component that is in the EOF state to receive a single-component command with the not-executed bit set 1) retransmits the command with the not executed bit cleared, 2) clears its internal EOF state so that the pixel will not execute any more single-pixel commands until its EOF state is set again by an EOF command or timeout, and 3) executes the command. In this respect, only one component can execute a single-component command. Once the EOF state is cleared, it will remain cleared until the next EOF command or a communication timeout occurs.

The error bit is a method by which components can indicate that an error has been detected. The controller should always transmit the error bit in the cleared (zero) state. When a component detects an error, it will set the error bit on the next single-component command directed at that component. That is, when an error is detected, the component will wait until the next single-component command for which it is the target because components may only change command bytes and data for non-executed single-component commands that are received with the component in the EOF state. As discussed later, a component may also change its data portion of an SS command. Since errors are generally detected after the error bit has been retransmitted, the component waits for the next opportunity to signal the error on the next single-component command directed at that component. As such, the controller is usually not notified of errors until the next frame.

Figure 7 shows a simplified flow chart of this process by which a component decides if it should execute a command. Errors also prevent some commands from being executed (not shown).

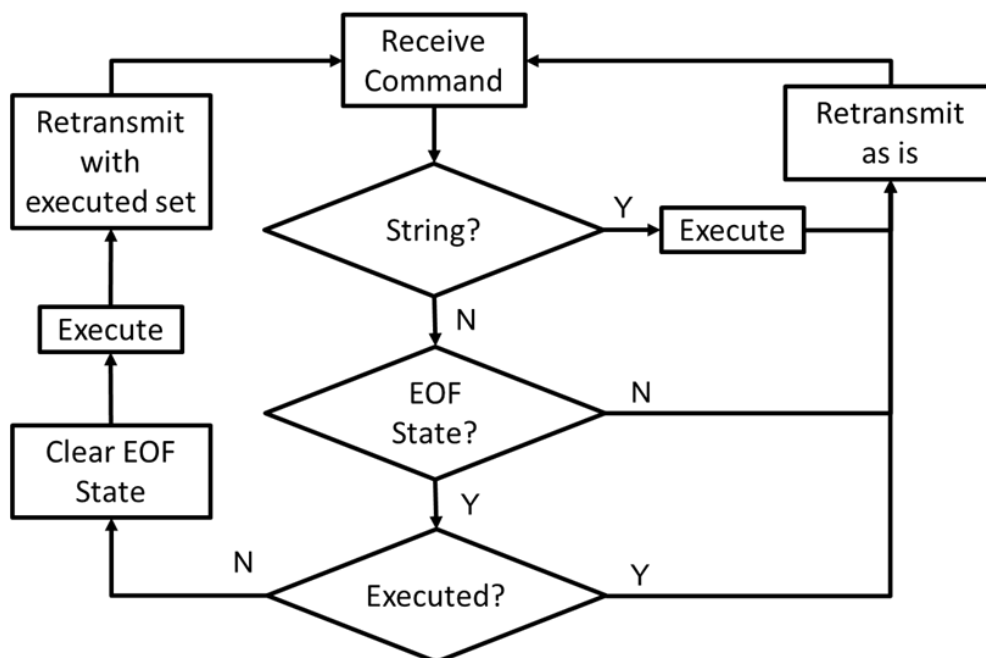


Figure 7. Simplified flow chart of component execution decision process

The EOF state is an internal state of the component that indicates it is ready to execute the next available non-string command. In the normal operation of a video screen, this would be set at the end of a frame where all components are set to receive their respective data for the next frame. The EOF state is set within the component by either power-on, reset, EOF command, or a communication timeout. The parity bit may optionally be used by the system to monitor the integrity of the communication loop. It should be set according to even parity (zero if the other 7 bits have an even number of ones, otherwise one). The receiver may take appropriate action if the parity bit is received in error. Only command bytes have a parity bit. Data bytes are 8-bit with no parity. Therefore, parity cannot be used to detect data errors. However, the CRC option (see below) is effective towards detecting data errors as well.

3.2 COMMAND SET

The command set table shows the available commands (values in decimal). Commands can either act on a single component, or the whole string. Some commands are only appropriate in one case. The number of data bytes expected to follow each command is listed. When the CRC option is selected, a CRC byte is added at the end of the data following a command. Only commands that send data use the CRC. The SF byte is listed in this table for convenience, but it is technically not a valid command.

Command Set (values in decimal)

Command	Single	String (all)	Num Data
SF	NA	213	3 (repeat 4x)
EOF	NA	129	0
Set Options	6	5	5 + CRC
Write Memory	10	9	130 + CRC
SS Brightness	NA	141	Variable + CRC
Set Brightness; Return Data	150	149	Num Color Bytes + CRC
Skip	154	NA	0
Reset	30	29	0
Measure Vf	34	33	0
Measure Vr	166	165	0
Measure T	170	169	0
SS Skip	NA	177	2
Write Events	54	53	16 + CRC
Swap Flash	58	57	0

3.2.1 EOF

The EOF command does two things: 1) sets every receiving component “EOF state” so each will execute the next non-executed, non-string command; and 2) causes the component to search for the next valid set of four SF bytes before interpreting any more commands. This is an essential part of the protocol allowing individual component addressing. Once the component receives one non-executed, non-string command, it will clear its internal EOF state until the next EOF command sets that state again (another EOF command, reset, power cycle, or timeout). When a device is not in the EOF state, it will only execute string commands. String commands do not affect the EOF state. The most common use of the EOF command is at the end of the data frame, setting all devices in the string to be ready to accept the next frame. While other events such as communication timeout restore the component to the EOF state, using the command method is recommended practice. After an EOF command, it is recommended that either a string of any number of zeros be transmitted or immediately continue with a set of four SF bytes followed by the next command.

3.2.2 Set Options

The OptiLamp options are set via the five bytes following a set options command as shown in the options table. Set options may target either one component or all the components at once. Care must be taken to be sure that options that affect the data width (color depth and enable CRC) must be the same for all pixels, otherwise the data stream may become misaligned with unexpected behavior. Therefore, it is recommended that the string version of set options be used first with those common options (when changing the color depth or CRC options), then single options commands can be used for individual components as needed, but being sure not to change the data widths. When enabling the CRC, it is immediately enabled and the options command that enables it must have a CRC byte.

The options default to zero at power-up. When the options are being changed, the OptiLamp sets the LED driver outputs to zero. As such the use of options should be part of setup sequences, but generally not part of repeated operations such as during video display to avoid undesirable interruptions.

The following is a summary of the various option categories which are described in more detail in their respective sections.

Byte	Bit	Alias	Option
1	0	NA	NA
	1	GAM	Enable Decompression/Gamma
	2	CRC	Enable CRC
	3	TC	Enable Thermal Compensation
	4	DEC	Enable Decode
	5	CAL	Enable Calibration
	6	CD0	Color Depth 0
	7	CD1	Color Depth 1
2	0	F0	Enable Flash 0
	1	F1	Enable Flash 1
	2	CO0	Color Order 0
	3	CO1	Color Order 1
	4	CO2	Color Order 2
	5	RNG0_0	Current Range 0_0
	6	RNG0_1	Current Range 0_1
	7	RNG0_2	Current Range 0_2
3	0	RNG1_0	Current Range 1_0
	1	RNG1_1	Current Range 1_1
	2	RNG1_2	Current Range 1_2
	3	RNG2_0	Current Range 2_0
	4	RNG2_1	Current Range 2_1
	5	RNG2_2	Current Range 2_2
	6	PWM0	Number PWM 0
	7	PWM1	Number PWM 1

Options (continued)

Byte	Bit	Alias	Option
4	0	CPD	ADC Power Down
	1	RT0	ADC Trim 0
	2	RT1	ADC Trim 1
	3	NA	NA
	4	NA	Must be 0
	5	NA	Must be 0
	6	NA	Must be 0
	7	NA	Must be 0
5	0	CS0	Current Select 0
	1	CS1	Current Select 1
	2	RV	Reference Voltage; 0 = VCC, 1 = 1.2V
	3	DPD	Driver Power Down; 0 = on, 1 = off
	4	NA	NA
	5	NA	NA
	6	NA	NA
	7	AN	Analog

Enables

Several features (decompression, CRC, thermal compensation, decode, calibration, flash) are enabled only when their respective bit is set. Their function is described in their respective sections. Options such as gamma, thermal compensation, and calibration require a write memory command to populate the memory with valid curve data before using.

Color Depth

Four options of the color depth are shown in the Color Depth (CD0/CD1) table. The color data spans the number of bytes according to the set option, starting LSB first for LED 0, then LSB first for LED 1, with LED 2 data at the end. The respective colors for LED 0, 1, 2 are controlled by the color order option so that the data order is user selectable (see Color Order section). **Important:** All components in a string must be set to the same color depth before transmitting brightness commands.

Color Depth	Depth Standard	Actual Depth	Num Color Bytes	Num Bits		
				LED 0	LED 1	LED 2
0	24-bit	24	3	8	8	8
1		40	5	14	14	12
2	48-bit	48	6	16	16	16
3		72	9	24	24	24

On power-up, the component defaults to a zero (0) color depth setting – the lowest setting of 24-bit color (3 data bytes following a brightness command). Only brightness commands have the number of data bytes adjusted for color depth. To promote less data traffic (and therefore more components in each string at a given frame rate) lower color depth can be used in combination with decompression curves and/or decode to achieve high dynamic range. Since a color depth value of 3 (72-bit) represents the full raw (linear) dynamic range, it is recommended not to use the decode option in that case and limit the use of the decompression curves as they reduce the resolution of the raw 72-bit data.

Color Order

The Color Order (CO2/CO1/CO0) table shows how the three (3) color order option bits allow the user to specify the data order for the three LED chips. The OptiLamp family of packages can have different chip-to-ASIC connections, the specific package color order is provided in the specific product datasheets.

Color Order	Data Order		
	LED 0	LED 1	LED 2
0	0	1	2
1	0	2	1
2	1	2	0
3	1	0	2
4	2	1	0
5	2	0	1

Current Select and Current Range

The Current Select (CS0/CS1) works with the Current Range (RNG#_#) options to set the maximum current for the LEDs. Current Select should be considered first as it affects all three LEDs with one value.

The individual LED currents can be ranged downward from the Current Select value with their respective 3-bit current range options. Each value of current range represents a factor of two steps in the range as it represents a bit shift of the data before transferred to the PWM driver. The Current Select / Current Range table lists the typical values for the maximum current for the various combinations. Note that the dynamic range will be reduced from 24-bit per color for Current Range values below 6 (blue shaded region). This is because current range simply defines a bit shift of the data whereas current select is the analog driver output setting and does not affect dynamic range. Current Range is not compatible with the analog driver mode and must be set to 7 if the analog option is used.

Current Range	Dynamic Range	Current Select			
		0	1	2	3
0	18-bit	0.01	0.03	0.05	0.06
1	19-bit	0.03	0.05	0.10	0.13
2	20-bit	0.06	0.10	0.20	0.25
3	21-bit	0.11	0.21	0.40	0.50
4	22-bit	0.23	0.42	0.79	1.00
5	23-bit	0.45	0.84	1.58	2.00
6	24-bit	0.91	1.68	3.17	4.00
7	24-bit	1.81	3.35	6.34	8.00

Number PWM

The Number PWM (PWM0/PWM1) option specifies the number of PWM periods (1-3) to be displayed after a brightness update. If Number PWM is zero, the PWM output will remain on until the next brightness update or another command that turns off the output. When 1-3 is selected and the frame period is longer than the 1-3 PWM periods, the PWM phase will be preserved from frame to frame. When 0 is selected, the PWM will generally be different from frame to frame as the PWM counter continues in the next frame where it left off in the previous.

In some applications it is desirable to only display an integer number of PWM periods (1-3) for each frame. This also provides for control of the PWM phase (see the PWM Phase section).

For other applications, it is more desirable to continue the PWM output (by setting Number PWM to zero) so that the output remains on until next frame. One such case is when the PWM period is longer than the Frame period. In this scenario, frames with identical brightness value may not have identical integrated output, but the difference will be very small and generally unnoticeable. The output accuracy across several frames is not diminished because the PWM counter is not reset between frames.

ADC Power Down

The ADC provides voltage measurements for temperature and LED health. While the default value at power up (CPD = 0) provides for the ADC to be on, setting this bit to 1 (power down) may conserve power.

ADC Trim

The ADC is a simple sigma-delta design. It provides 16-bit resolution, but the accuracy and linearity depend on the sampling frequency and filter time constant. The 2-bit value for ADC Trim (RT1/RT0) may help optimize the linearity of the ADC. A value of 3 is recommended.

Reference Voltage

The ADC Reference Voltage (RV) should be set to 0 (3.3V) when measuring LED voltages, and 1 (1.2V) when measuring temperature.

Analog

When the Analog (AN) option is set, the driver produces a non-pulsed brightness output. This mode improves recording with unsynchronized photography. When using the AN option, the Current Range must be set to 7 for all three LEDs. Any of the four Current Select options may be used.

3.2.3 Write Memory

The write memory command loads 130 bytes of decompression curves, thermal compensation curves, calibration parameters, and timing parameters to the component. It may be directed to single components, or the whole string at once. Typically, the write memory command is used at power-up. As with the options parameters, the memory is volatile and is reset to zero values at power-up.

The Write Memory map shows the position of the various elements which are discussed further in their respective sections.

Write Memory Map

Command Data Bytes	Value	Function	Notes
0-13		Thermal Comp 0, bytes 2-15	First two bytes transmitted near end
14-29		Thermal Comp 1, bytes 0-15	
30-45		Thermal Comp 2, bytes 0-15	
46-47	CalFctr 0	Calibration Factor for LED 0	
48-49	CalFctr 1	Calibration Factor for LED 1	
50-51	CalFctr 2	Calibration Factor for LED 2	
52-53	TempShift	Temperature shift value	
54-55	CalShift 0	Calibration Offset Factor for LED 0	
56-57	CalShift 1	Calibration Offset Factor for LED 1	
58-59	CalShift 2	Calibration Offset Factor for LED 2	
60-61	TempScale	Temperature multiplier value	
62-77		Gamma 0, bytes 0-15	
78-93		Gamma 1, bytes 0-15	
94-109		Gamma 2, bytes 0-15	
110-124		120-bit events	
125		Not used	
126-127		Thermal Comp 0, bytes 0-1	
128-129		PWM Phase	

3.2.4 Write Events

The write events command loads a subset of the write memory command. It is provided as a more efficient means to load all components with their specific data in the case that all of them have the same calibration and curve data and only the event parameters vary. It requires 16 bytes of data with the last byte not used.

3.2.5 Set Brightness; Return Data

The set brightness command may be directed to single components, or the whole string at once. In both single and string commands, the controller transmits the number of bytes according to the color depth option as shown in the color depth table.

For string commands, the components don't edit the data. i.e. there is no return data for string commands. String commands request all components on the string to go to the same brightness level. String commands are useful for setting all pixels to one color or the zero-brightness level with one short command string.

For single component commands, the target component will replace the data with temperature, voltage, leakage, or the error code. The data returned depends on the state of the component as covered in detail in the return data section.

Important: All components in a string must be set to the same color depth before transmitting any brightness commands.

3.2.6 SS Brightness

SS (super string) brightness is a data-efficient version of the set brightness command which addresses all components in the string with one command and header rather than resending a brightness command for each component. It may therefore be the preferred method to transmit brightness data to the whole string, particularly for long strings at high-frame-rate video applications.

The header includes the zero-indexed number of data bytes (*Num Data*) and the number of *Data Units*. Both numbers are sent with the least significant byte first. The table below shows the structure of the command as the controller sends it. As each component receives and repeats the command to the next component, it decrements the Data Units counter, receives the data for which it is target, and replaces that data with measurement and error data in the same way as is done with the standard brightness command according to the return data section.

Num Data remains unchanged and is used to tell all components how much data to expect for the command.

Data Units tells each component which unit of brightness data to accept for that component and replace that data with return data according to the return data rules described below. A data unit is 3-9 bytes of color data according to the color depth selected. Each component will decrement this value by one for the next component and load the value internally and count down to zero. Once it reaches zero, it will accept the brightness data as the target component for that data. In this manner, the data for the first component on the string is transmitted last while the data for the last component is transmitted first.

Important: All components in a string must be set to the same color depth before transmitting any brightness commands.

	Num Bytes	Value	Description
Header	1	141	Command Code
	2	<i>Num Data</i> -1	Zero-indexed number of data bytes, $n * [\text{Num Color Bytes}] - 1$
	2	<i>Data Units</i> -1	Zero-indexed number of pixels, $n - 1$
Data	Num Color Bytes		Brightness data for last pixel, n
	Num Color Bytes		Brightness data for pixel $n - 1$
	...		
	Num Color Bytes		Brightness data for pixel 1
	1		Optional CRC

3.2.7 Skip

Normal operation generally doesn't require the use of the skip command. The skip command clears the EOF state of the target component the same way any other single component command does. Since a single component command will be executed by the first component with a set EOF state, this command provides a method to target any single component within the string without affecting the components before it first. For example, if it is desired to target the fifth component with an options command without first targeting the first four components, the controller may first send an EOF to set the EOF state of all components, then send four skip commands followed by the options command with respective data for the fifth component. The first four components must be targeted before the fifth component can be targeted. The skip command allows this to take place without any other effect.

3.2.8 SS Skip

The SS skip command is a data-efficient version of the skip command allowing many components to be skipped with one command and header. The header is a zero-based 2-byte number of components to skip, sent with the least significant byte first. Each component will decrement the number by 1 even after it reaches zero at which time the counter will roll over from 0 to 65535 which is considered as a negative number because its MSB is set. Once the counter is negative, receiving components will no longer be skipped. The maximum number of components that can be skipped within a string using SS Skip is 32768. Since strings of that length are not expected, the skip value will remain negative and not affect the remainder of the string.

Num Bytes	Value	Description
1	177	Command Code
2	Num Skip -1	Zero-indexed counter to skip (n-1)

3.2.9 Reset

The reset command implements a soft reset of relatively few things and may not be needed but is recommended when starting communication after power-up or timeout. The only things impacted by the reset command are the output (turning it off), resetting the error codes, and restarting the ADC. It doesn't clear the memory or options.

3.2.10 Measure Vf

Measure Vf initiates the measurement of the voltage at the cathode of each LED at the current settings selected by the options. The measurement is conducted one LED at a time. A noticeable flash may occur. By making this measurement at very low current, the flash may not be noticeable.

The SD-ADC is considerably slow and measures one LED at a time. The measurements require about 270k cycles at the frequency selected by the ADC clock option for each of three measurements for each LED. So if the ADC clock is at 3 MHz, the three measurements finish and are available in about 270 ms. After that, the three 2-byte measurements can be read back by one of the brightness commands (see Return Data section). Previous Vf or Vr data should be read back before requesting another measurement. For the forward bias measurement, the LED anode is near Vcc and the measurement is taken at the cathode. The forward voltage of the LED chip is approximated by:

$$V_f = V_{RV} \left(1 - \frac{\text{Reading}}{65535} \right)$$

where V_{RV} is the comparator voltage selected by the reference voltage option which should be V_{CC} .

3.2.11 Measure Vr

Measure Vr initiates the measurement of the voltage across a resistor in series with the LED biased in reverse, thus providing insight into the LED health, sometimes before failure is otherwise observed. It is made via the voltage measurement at the cathode. During the measurement cycle, the anode voltage is inverted (effectively GND), the sink driver is turned off, and the cathode node is connected to V_{cc} through a pull-up resistance of approximately 22 kohm. This allows the reverse current for each color LED chip to be approximated by:

$$I_{rev} = V_{RV} \left(1 - \frac{Reading}{65535} \right) / 22000$$

where V_{RV} is the comparator voltage selected by the reference voltage option which should be V_{cc} . As with Measure Vf, the three LEDs are measured one at a time and the three 2-byte measurements can be read back later. Previous V_f or V_r data should be read back before requesting another measurement.

3.2.12 Measure T

Measure T initiates the measurement of the temperature. All other measurements must have finished (the ADC must be idle), otherwise the command will be ignored. The temperature is only measured by command, so if thermal compensation is used, it is up to the user to send the Measure T command at regular intervals such as 1/100 or 1/1000 frames. During normal operation such as video frames, the return data that replaces the brightness data consists of only the temperature and an error code. The first two bytes are the temperature reading with the least significant byte first, and the last byte is the error code (usually zero). Depending on the color depth, there may be intermediate bytes which are repeated temperature readings and can be ignored. The voltage of the temperature sensor V_{TC} is:

$$V_{TC} = V_{RV} \left(\frac{ADC_{TC}}{65535} \right)$$

where ADC_{TC} is the raw 16-bit reading from the A_{DC} and V_{RV} is the comparator voltage selected by the reference voltage option which should be 1.2V. The temperature may be approximated by:

$$T = 1000 * [0.832 - V_{TC}] / 1.76$$

3.3 CRC

OptiLamp provides for cyclic redundancy check (CRC). If the CRC option is selected, a CRC byte is added at the end of the commands that contain data (except SS Skip) as shown in the commands table. Commands that don't have data such as EOF don't get CRC bytes added to them. They depend only on the parity bit for error detection. If a CRC or parity error is detected, the component will ignore most commands and communicate the error to other components and the controller as described in the error reporting section. The CRC used is the standard CRC-8/ROHC algorithm which operates on 8-bit data using the 8-bit polynomial, $1+x+x^2+x^8$, otherwise referred to with its hexadecimal notation, 0x07. An initial value of 0xFF is used. The CRC is reinitialized between commands and is calculated from the command code with its header and data bytes.

3.4 Return Data

The target component of the single component set brightness command replaces the brightness data with temperature and error status, voltage, or leakage. The SS brightness command also returns the same for the data segment of the respective target component. The replaced data is returned as it gets repeated through the string of components, eventually returning to the controller. There is no requirement for the controller to process this data. In simple systems, it may be ignored. The returned data are predetermined values stored in the component that are available at the time the serial command is received. As a component receives brightness data, there are five possibilities depending on the state of the component:

- 1) No replacement – repeat data as received. This is the response if the command is a string command or the component is not the target component (command already executed or component in EOF state).
- 2) Temperature for the first two bytes and error status for the last byte. This is the normal return for a target component of a non-string command. If the color depth has more than 3 bytes, the other bytes may be duplicates and should be ignored.
- 3) Up to 6 bytes of reverse leakage data. When the reverse leakage data becomes available after executing a Measure Vr command, reverse leakage is returned as the return data for the first occurrence of a non-string brightness command for which the component is target. Enough time must elapse for the measurement to be available, otherwise the return data will be deferred to the next time it is the target for brightness. Since the last byte is always the error code, 72-bit color depth should be selected prior to reading the data. In that case, the first 6 data bytes returned contain the leakage data and the 9th byte contains the error code. Since lower color depth can't transmit all the return data, if lower color depth is being used it may be advised to change the options to 72-bit color for the purpose of reading the data, send a zero-brightness command to read the data, then change the options back to the desired color depth settings for normal operation.
- 4) Up to 6 bytes of forward voltage data. Forward voltage data is read back the same way as the leakage data (see above).
- 5) Zeros. If no measurement has finished, the target component will return zeros with the last byte being the error code.

3.5 Error Reporting

OptiLamp has 4 mechanisms for detecting/reporting errors:

- 1) Parity bit in command byte
- 2) Error bit in command byte
- 3) Error byte returned in brightness command
- 4) CRC at the end of command data (if enabled)

Most errors (such as CRC errors) cannot immediately be detected until after the opportunity to report the error has passed. This is because the error byte is sent before the CRC with commands and data being repeated in real time with only a 3-bit delay. Furthermore, components are unable to create a new data stream as they rely on the controller to provide brightness commands for which it can replace data. Therefore, when an error is detected, the component stores the error internally and transmits it on the next occurrence of a brightness command for which it is the target component.

Another level of complexity is that the communication isn't peer-to-peer, but a lengthy string of repeaters. If an error occurs, we would like to know the location where it was first detected. Even with these complexities, it is possible to determine the error type and the location where it occurred.

Code	Bit	Description
PT	0	Parity Error At Target
CT	1	CRC Error At Target
PAE	2	Parity Error After Execution
PBE	3	Parity Error Before Execution
CAE	4	CRC Error After Execution
CBE	5	CRC Error Before Execution
SF	6	SF extra ones
BNR	7	Brightness Not Ready

When a communication error is detected, if it is a brightness command, it is ignored, but it will still be retransmitted according to the Error Communication Rules table, as shown below:

Command Type	Command Status	EOF State	Target	Error	Codes Generated	Actions
SS/String	NA	0/1	Yes	Parity	PT, PBE	None
	NA	0/1	Yes	CRC	CT, CBE	Transmit Corrupt CRC
Pixel	Executed	0/1	No	Parity	PAE	Retransmit with corrected parity
	Executed	0/1	No	CRC	CAE	None
	Not Executed	1	Yes	Parity	PI	- Clear Not Executed bit - Set Error bit on next pixel command
	Not Executed	1	Yes	CRC	CI	- Transmit corrected CRC - Set Error bit on next pixel command
	Not Executed	0	No	Parity	PBE	None
	Not Executed	0	No	CRC	CBE	- Set Error bit on next pixel command - Transmit Corrupt CRC

If a controller is to respond to errors appropriately, some algorithm is required. The above rules provide the details on how errors are communicated between components and ultimately back to the controller. The following is a list of observations to aid development of such algorithms:

- 1) When SS and string commands have parity errors, the parity errors are retransmitted to the end without correction.
- 2) Pixel commands fix parity when received by the target pixel and set the error bit in the next pixel command. i.e. the error bit indicates that an error happened before the target for that command.
- 3) The error bit is not used for SS/string commands.
- 4) Error byte codes are generally returned in the next frame.
- 5) When a parity, error code, or corrupt CRC are received by the controller, the simplest way to determine the location and cause is to send an SS brightness command and receive all the error codes at once. In many cases, this could be implemented simply by continuing to send the next frame and looking at the return error codes.
- 6) When errors occur, the internal registers of target pixels may still get changed. For brightness, the error prevents the component from updating the PWM output. But this cannot be done for options and writing memory because the error flag may not occur until the end (CRC) and the internal registers have already been updated at that point. It is the responsibility of the controller to respond appropriately in those cases.
- 7) Since Options and Memory commands are static and must be correct for even basic functions, it is advisable for the controller to be sure they are communicated without error before sending brightness commands.

3.6 Calibration and Compensation

OptiLamp provides extensive calibration and temperature compensation. By default, no calibration or compensation is used at power-up and the device may be used in a simple system by only providing brightness levels through the brightness command. The memory may be loaded with the calibration coefficients (see the write memory command) to utilize this extended capability.

Depending on which options are selected, the intermediate output to each LED may generally be described as:

$$\text{intermediate output} = \text{CalFctr} * g(d(\text{brightness})) * f(\text{scaledTemp})$$

Where:

CalFctr is the 2-byte calibration factor.

g is the 17-coefficient piecewise continuous transfer function defined in memory as “Gamma” as defined in the Gamma/decompression section.

d is the decode function as defined in the Decode section.

brightness is the data input by one of the brightness commands.

f is another 17-coefficient piecewise continuous transfer function defined in memory as “Thermal Comp.”

scaledTemp is value received by the temperature sensor after being scaled by the temperature scaling values according to:

$$\text{scaledTemp} = \text{TempScale} * (\text{ADC}_{TC} - \text{TempShift})$$

ADC_{TC} is the 16-bit raw data reading from the temperature sensor.

scaledTemp is not an actual temperature value, but rather a shift and expansion of the ADC reading to utilize the full 16-bit range in preparation to be used with the thermal compensation curves.

The general equation for *intermediate output* above applies to the case when all four transformation options are enabled. As various options are disabled, the equation is still valid by assuming the following:

If calibration is not enabled, $\text{CalFctr} = 1$

If decompression is not enabled, $g(d(\text{brightness})) = d(\text{brightness})$

If decode is not enabled, $d(\text{brightness}) = \text{brightness}$

If thermal compensation is not enabled, $f(\text{scaledTemp}) = 1$

All calculations are conducted as 24-bit (full 72-bit color) regardless of color depth. As a result, the output has 24-bit dynamic range, particularly when gamma and/or decode are used. Using an appropriate transfer function for gamma expansion allows for near-24-bit dynamic range while transmitting only 8 bits per color.

The final calculation step for the *PWM Output* is to apply the user-defined *CalShift* to the *intermediate output* according to

$$\text{PWM Output} = \text{intermediate output} + \begin{cases} 0 & \text{if } \text{intermediate output} = 0 \\ \text{CalShift} & \text{otherwise} \end{cases}$$

This functionality is provided because in some cases the user may want to select the PWM level for the first non-zero brightness. In particular, the first several levels may not be visible, and the user may want a better match at the very lowest brightness levels. According to the equation, the zero level is not affected by the shift so that zero always produces zero output.

3.6.1 Gamma/Decompression Curves

OptiLamp supports user-defined decompression curves with gamma being the most common type. (Some use of the term “gamma” in this document may refer to any compression/decompression curve.) Figure 8 demonstrates the use of the transfer function and the coefficients to implement a gamma of 2 with a piecewise linear function. There are 16 coefficients that define the output values of the endpoints of 16 segments with the 17-th endpoint hardcoded to be at the maximum value of 255. The coefficients themselves are truncated to 8-bit (8 most significant bits of a 24-bit word), but the interpolated values are 24-bit. Truncating the coefficients in this way is not detrimental to the fidelity because the resulting transformation is smooth with 16-bit precision and the truncated nodes match the respective decompression curve. Since the piecewise linear function is not an exact representation of a true gamma curve, the following may be considered:

- The piecewise linear function is close enough and cannot be distinguished in most demanding requirements.
- The controller that uses gamma for compression can also use the piecewise linear function for encoding the compression so that the decompression in the component is exact.

The transfer function logic receives the input (brightness sent by the controller), determines the correct pair of coefficients to use, and linearly interpolates the output to 24-bit per color precision. Figure 8 shows typical compression/decompression gamma curves: compression with gamma = 0.5 which could be implemented on the controller side before truncating higher color depth to 24-bit, and gamma = 2 which is loaded in the component memory to decompress the color data. The result is much improved grey levels, closely replicating the original raw color depth. Any other transfer function can be used since 16 of the 17 vertices are user-defined. See the section Curve Transformation for the procedure to transform the curve data for the Write Memory command.

CoefNum	Gamma 2	Gamma 0.5
0	0	0
1	1	64
2	4	90
3	9	110
4	16	128
5	25	143
6	36	156
7	49	169
8	64	180
9	81	191
10	100	202
11	121	211
12	143	221
13	168	230
14	195	239
15	224	247
16	255	255

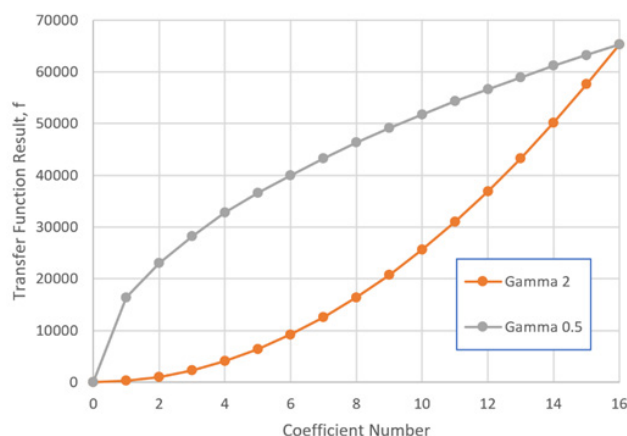


Figure 8. Transfer function and coefficient examples.

3.6.2 Decode

Decode is a separate hard-coded decompression scheme that provides more dynamic range extension than the decompression curves. Since it is hard-coded, its use doesn't require any parameters to be provided via Write Memory. This scheme uses a proprietary exponential/log curve given by

$$result = int \left(\begin{cases} m2^{(13-nb)} & , if e = 0 \\ (m + 2^{(nb-4)})2^{(e+12-nb)} & , otherwise \end{cases} \right)$$

Where nb is the number of color bits (note that LED 2 is different for color depth 1),

$$e = int \left(\frac{input}{2^{(nb-4)}} \right),$$

and

$$m = input \text{ Mod } 2^{(nb-4)}$$

input is the number as received by the brightness command because decode is the first step in calculating the final output – implementing the $d()$ function in the overall calibration and compensation calculation above. *result* is the decoded value used for the next step of the calculation which is determined by the remaining options (i.e. $result = d(input)$).

Figure 9 shows how data are mapped into the internal 24-bit registers when using 8-bit color depth without decode and all four color depth options with decode. 8-bit data without decode simply maps into the upper 8 bits of the 24-bit register and because the PWM output is linear, the 8-bit dynamic range is less than 3 orders of magnitude (255). Using 8-bit color depth with the decode options expands the dynamic range of the 255 levels to achieve 19-bit dynamic range. This option pair may be suitable for most applications. While there are still only 255 grey levels, they are spread out well so that adjacent grey levels are difficult for the human eye to notice. While the 8-bit with decode options only utilize 19 of the 24 available bits of dynamic range, some of the extra 5 bits of range can be used for calibration and thermal compensation as both adjust the curve down according to their respective values.

The higher bit depth options in figure 9 show that the 12-bit color depth option with decode (color option 1 has both 12 and 14-bit) utilizes the full 24-bit dynamic range. 14, 16, and 24-bit options also map to the full 24-bit internal register, but the usefulness of the decode option diminishes as the input color depth increases. Since the higher input bit depths map beyond 24-bit, many of the lower input levels map to duplicate output levels. Using the 24-bit input depth with decode changes the transfer characteristic at a cost of resolution. It is expected that designers using 24-bit would choose to utilize the direct raw access to the internal register without using the decode option.

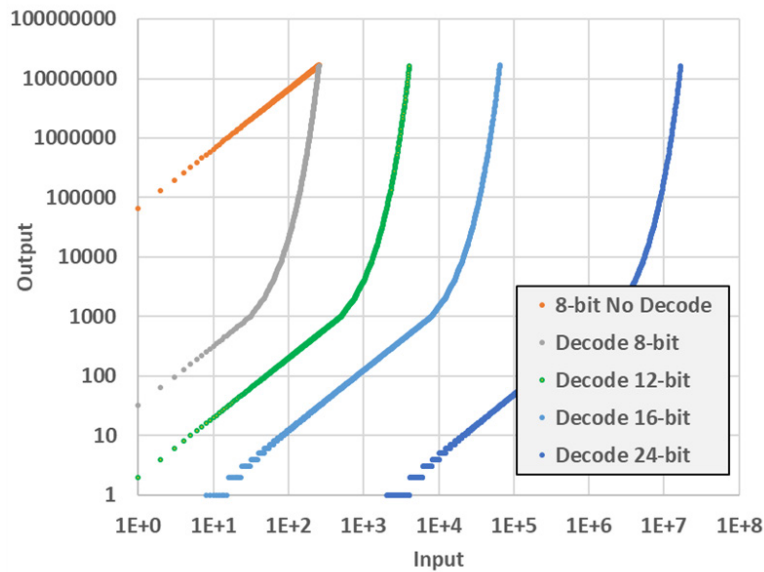


Figure 9. Decode Curves

3.6.3 Mapping with Both Decompression Curves and Decode – OLY6D-FKC Examples

Figure 10 shows several examples of the various decompression options as they apply to an 8-bit input. As before, 8-bit with no decompression or decode options gives a very small dynamic range with 255 linear levels (zero is always zero and not included for dynamic range).

The black curve of figure 10 is not an available option with the OLY6D-FKC but shows what we would get if the ASIC could calculate a gamma of 3 exactly. Gamma of 3 was chosen as it is required to be that high to decompress 8-bits to 24-bits of dynamic range. Using such high gamma is not a problem because a preprocessor is responsible to encode the 8-bit data according to the color curve that matches the content. If the goal were to have the OLY6D-FKC do all the color curve calculation without the remapping of the preprocessor, higher bit depth should be considered. But in this example, the goal is to get the most out of 8-bit data with good resolution (as perceived by the human eye) across a large dynamic range. As we see, from the granularity of the black curve for the lower 4 orders of magnitude, an exact calculation of gamma doesn't produce good resolution over the full 24-bit dynamic range using 8-bit input.

The red curve of figure 10 shows the closest representation of the exact gamma=3 curve available with the OLY6D-FKC user-defined piecewise function decompression curves (without decode). We see that it expands the 8-bit data to 16-bits and has good resolution except for the lowest few levels.

The grey curve of figure 10 shows the decode-only option with 8-bit input which gives 19-bit dynamic range. As discussed previously, this may be the preferred option in many applications as it yields high dynamic range with a lot of range remaining for calibration and thermal compensation.

The purple curve of figure 10 shows how both decompression and decode can be used together to achieve 24-bit dynamic range from 8-bit input while also having good resolution. Since the controller would have a one-to-one mapping (either by calculation or lookup table), the shape of the curve can be accounted for by proper controller encoding. The benefit is that 24-bit dynamic range can be achieved with 8-bit data while preserving good resolution with respect to the log response of the human eye over the full range.

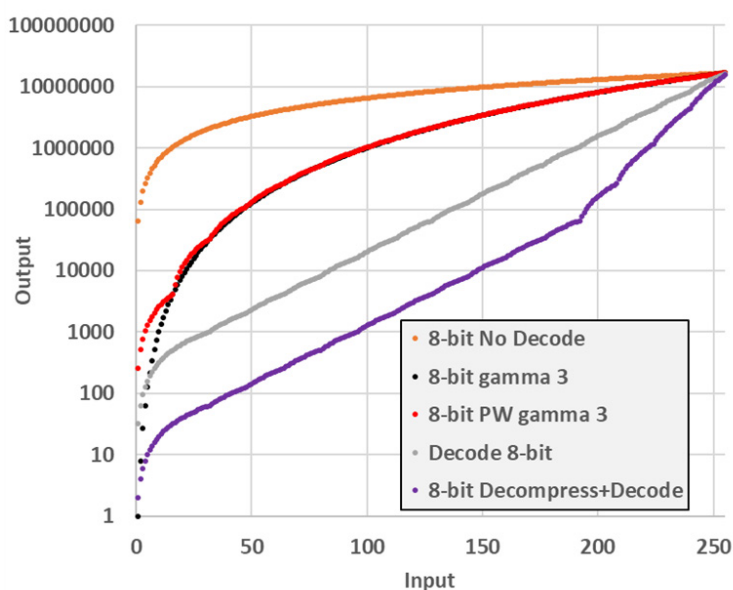


Figure 10. Combination curve examples

3.6.4 Thermal Compensation Curves

The thermal compensation, f , is implemented the same way as for the decompression transfer function, g , except its input is the scaled temperature value, $scaledTemp$, and 17th endpoint is hard coded to zero.

Figure 11 shows an example of thermal compensation curves. The grey temperature curve shows the input to function f over the temperature range. These input values are after a shift and scaling are applied to the raw temperature sensor output. By scaling the raw temperature first, more of the coefficients are utilized over the useful range and the hardcoded 17th endpoint doesn't affect the result because it is out of range.

The coefficients shown in Figure 11 represent the average response of RGB LEDs over temperature and are therefore recommended for general use. The blue curve can be used for green as well because their temperature dependence is very similar. More precise curves may be obtained through calibration over temperature, but this step is unnecessary in most applications.

Adj Reading	T[°C]	Blue Coef	Red Coef
0	150.0	0	0
4095	137.0	0	0
8191	124.0	0	0
12287	111.0	255	255
16383	97.9	246	220
20479	84.9	237	189
24575	71.9	230	161
28671	58.9	223	136
32767	45.9	217	114
36863	32.8	211	95
40959	19.8	206	80
45055	6.8	202	67
49151	-6.2	198	58
53247	-19.2	194	52
57343	-32.3	191	49
61439	-45.3	188	49
65535	-58.3	0	0

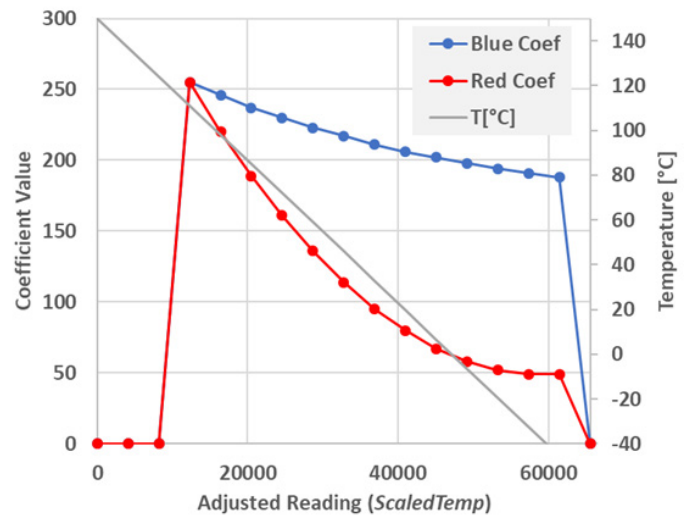


Figure 11. Example thermal compensation curves

Figure 11 shows that the f function has much more effect on the red because red is much more temperature sensitive. The first three coefficients for all colors are set to zero to implement thermal shutdown at high temperature. Above 110°C the PWM output is reduced such that it becomes zero above about 125°C. This is the choice of the designer to determine the values appropriate for the system.

3.6.5 Curve Transformation

Figure 12 illustrates a method to transform the decompression and thermal compensation coefficients to be stored in memory by the Write Memory command. There are other methods to achieve the same result, but Figure 12 shows that the transformation can be achieved by writing out the values in reverse-order binary, rotate the array of bits clockwise, and convert each row to its hexadecimal equivalent. As with all other multi-byte values, the least significant byte is sent first. For the example of figure 12, the byte order would be AA, 5A, 00, 50, 44,

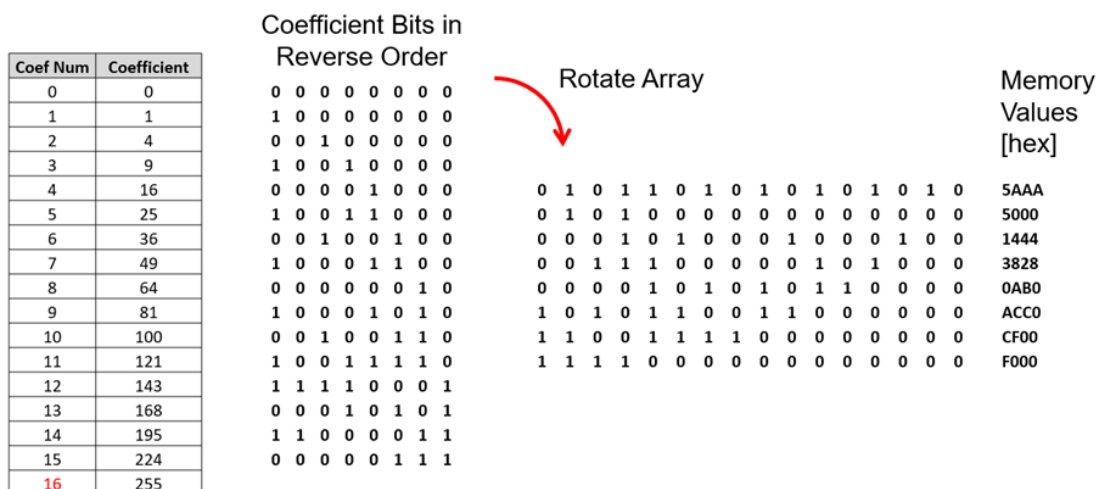


Figure 12. Curve Transformation

3.7 Timing

3.7.1 Events: Blanking and Flash

OptiLamp offers precise control over the timing of the LED output for synchronization within the display as well as with external systems. A blanking period where all components are off simultaneously can be achieved. Having all components start and end each frame at or near the same time in this way may eliminate the perception of flickering that can exist when adjacent components receive their data at significantly different times due to the string routing or module placement of the display. The timing can be tailored so that the component outputs are energized at different times to reduce power fluctuations. Furthermore, pulses at different times for left and right frames within the blanking period can be configured to be compatible with common 3D shutter glasses.

When new brightness data is received and the next PWM value is calculated, the OptiLamp ASIC delays conveyance of the new value to the driver output until after a series of 6 times elapse. The times are defined by 6 20-bit values (bytes 110-124 in Write Memory or Write Event data). These values are set to zero at power up in which case new data is conveyed to the output in the shortest possible time. The timer counts to these values at the communication frequency.

Important: The events timing begins when each component receives its respective data. The controller should not stop communication until after the events have enough time to finish. Otherwise timing between components may vary. This is because the components rely on the clock of the controller to accurately time the event process. As per the Data Stream Sequence section, it is recommended that the controller maintains communication continuously by sending zeros between frames for other reasons as well.

The Timer Event States table shows how the LED Output depends on the various timer states. The timer remains in the idle state until the component receives a new brightness, at which time it proceeds through all six timer states (1-6) in order. The PWM is off for all states except the first which is the delay for blanking to begin. The blanking time is the sum of times 2-6. Flash energizes the LEDs for the time of the respective timer state if either of the flash bits are set in the options.

Timer State	Output State	Notes
Idle	Display	PWM output
1	Display	PWM output continues with previous value
2	Off	LED off – Blanking begins
3	Flash[0]	LED On continuously if Flash[0] = 1
4	Off	LED off – Between two flash times
5	Flash[1]	LED On continuously if Flash[1] = 1
6	Off	LED off – Blanking ends

As an example, the following table lists values that are known to work with 3D shutter glasses with communication speed of 3 MHz:

Timer	Value	Time μ s
1	372	124
2	2601	991
3	72	1015
4	327	1124
5	72	1148
6	4230	2558

3.7.2 PWM Phase

The PWM is formed by comparison between the desired output and the PWM counter. To reduce the pulse period, the counter is re-sorted so that there are many pulses within a PWM period. The PWM phase is determined by the initial value of the PWM counter which is a 2-byte number set by the write memory command. At power up, it is initialized to zero. In many cases it is unnecessary to change the PWM phase. But the option is available when precise control is required.

4. EXAMPLE

The text box below shows a simple example of a command/data sequence for controlling a string of pixels (OLY6D-FKC).

```

00 00 // Start with some zeros

D5 D4 D5 D5 // SF
1D      // Reset; sng-1E all-1D
00      // Follow reset with zero

D5 D5 D5 D5 // SF
81      // EOF

D5 D5 D5 D5 // SF

05 // options; sng-06 all-05
//      80  40  20  10  08  04  02  01
10 // 0001_0000 CD1  CD0  CAL  DEC  TC   CRC  GAM  na
E4 // 1110_0100 RNG0_2 RNG0_1 RNG0_0 CO2  CO1  CO0  F1  F0
3F // 0011_1111 PWM1  PWM0  RNG2_2 RNG2_1 RNG2_0 RNG1_2 RNG1_1 RNG1_0
00 // 0000_0000 na   na   na   na   na   RT1  RT0  CPD
00 // 0000_0000 AN   na   na   na   na   DPD  RV   CS1  CS0

95      // Brightness; sng-96 all-95
00 00 90 // (blue)

00 00 00 00 00 00 00 00 // zeros to allow process time
D5 D5 D5 D5 // SF

8D      // SSBrightness
0B 00    // NumData
03 00    // NumDataUnits
FF FF FF // 4 (white)
00 00 FF // 3 (blue)
00 FF 00 // 2 (green)
FF 00 00 // 1 (red)

00 00 00 00 00 00 00 00 // trailing zeros

```

The data sent to the pixels is represented as hexadecimal numbers and comments are added with “//” for ease of reading. Transmission begins with some zeros to synchronize the pixel clocks. Then a set of four SF bytes (D5h) starts the communication. Using a reset command at the beginning of communication is recommended (1Dh) which should be followed with a byte of zeros. Since it is reset, another set of four SF bytes needs to be used to reinstate communication. An EOF (81h) may be sent next to be sure that all pixels are ready to receive individual commands, but it is unnecessary in this example because it has already been reset and no individual commands are used. As such, this step is not required in this case and is included only to demonstrate the protocol. After an EOF or any series of non-data zeros, the pixels require another set of four SF bytes to resume communication.

In this example, we set the options for all pixels at once with the full-string option command (05h). In this case we’re using color depth 0 (24-bit color); but with the DEC option, we greatly extend the dynamic range. The color order is set to 1 for R-G-B transmission order with this device (OLY6D-FKC). All three color range values are set to the maximum (7). With PWM set to zero, the pixels will retain their output until commanded to do otherwise or power down.

The first brightness command (95h) sets all pixels to blue. In this example, we wish to change only the first four pixels with an SS brightness command. This is an uncommon sequence, but useful for demonstration. Since all pixels just received a command to display blue, they are not immediately ready to display a different color because the first brightness command must finish being processed first (calculation of gamma, decode, calibration, compensation, etc.) and are not yet ready for another brightness value. Some delay is required. This could be accomplished by stopping communication and restarting at a later time, but in this example, we insert enough zeros to provide the clock cycles necessary to process the calculations. Since we padded the time with zeros, we need to resynchronize with four SF bytes again to start communication again.

The second brightness command (SS Brightness, 8Dh) sends brightness data to four pixels within one command so that the first pixel is red, then blue, then green, then white (data in reverse order). The four bytes following the command provide the header for the data that follows as described in the SS Brightness section. While this could have been accomplished just as efficiently with four individual brightness commands, the data efficiency comes when SS Brightness controls longer strings.

Finally, after all communication is finished, ample zeros need to be transmitted (3 bits per pixel) so that the data sent from the controller makes its way through the entire string.

5. ACRONYMS

Acronym	Definition
ADC	Analog to Digital Converter
ASIC	Application-Specific Integrated Circuit
cps	Cycles Per Second
CRC	Cyclic Redundancy Check
DIO	Digital Input/Output
EOF	End of Frame (video frame)
ESD	Electrostatic Discharge
FPGA	Field Programmable Gate Array
FPS	Frames Per Second
HDMI	High-Definition Multimedia Interface
LED	Light Emitting Diode
LSB	Least Significant Bit
MSB	Most Significant Bit
NC	No Connection
PCB	Printed Circuit Board
RGB	Red, Blue, and Green
SD-ADC	Sigma Delta Analog to Digital Converter
SF	Start Frame (byte)
SMD	Surface Mounted Device
SS	Superstring (command)

6. CLOSING REMARKS

For answers to questions related to these high-brightness devices, please contact your local Cree LED salesperson or the High Brightness dedicated support team at HBHelp@cree-led.com.

This application note provides guidance and does not imply any specific warranty or guarantee.