# OSIRE® E3731i - Startup Guide

## Application Note

**Published by ams-OSRAM AG**
Tobelbader Strasse 30,
8141 Premstaetten Austria
Phone +43 3136 500-0
ams-osram.com

# OSIRE® E3731i - Startup Guide

**Application Note No. AN078**

**Valid for:**
OSIRE® E3731i

## Abstract

The OSIRE® E3731i is an “intelligent” multi-color LED including an embedded IC together with 3 LED-Chips. It is based on Open System Protocol that simplifies integration at customer side through an open control concept, daisy chaining of units, integrated temperature monitoring and access to calibration data for each individual unit.

This document describes the usage and communication with the OSIRE® E3731i.

# Table of contents


**1 Basic information ..... 4**

**2 Application description ..... 5**

2.1 Communication mode selection ..... 7

2.2 Connection to MCU ..... 7

2.3 Connection via CAN-FD ..... 8

2.4 Connection via LVDS ..... 9

2.5 Component selection guide ..... 9

**3 Basic operation ..... 11**

3.1 Example of basic operating sequence ..... 11

3.2 Device states ..... 12

3.3 Startup behavior ..... 13

3.4 Diagnostics ..... 14

**4 Color control and temperature stabilization ..... 14**

4.1 Calibration data ..... 14

4.2 PWM calculation ..... 15

4.3 Temperature stabilization ..... 16

4.3.1 Temperature sensor ..... 17

4.3.2 Temperature prediction ..... 17

4.3.3 Simple temperature compensation - I ..... 17

4.3.4 Simple temperature compensation - II ..... 18

4.3.5 Optimized temperature compensation functions ..... 19

**5 Appendix ..... 21**

5.1 Glossary ..... 21

5.2 OSIRE® E3731i OSP specification ..... 22

5.2.1 Message encoding ..... 22

5.2.2 Message frame format ..... 23

5.2.3 Message handling ..... 24

5.2.4 Cyclic redundancy check ..... 26

5.2.5 Commands ..... 27

5.3 OTP to PWM example ..... 36

5.4 Code examples ..... 40
5.4.1 Manchester coding ..... 40
5.4.2 Message cyclic redundancy check ..... 42
5.4.3 OTP cyclic redundancy check ..... 45
5.5 Example chain configurations ..... 46
5.6 Message timing estimations ..... 48

# 1 Basic information

The OSIRE® E3731i is an “intelligent” multi-color LED for automotive illumination tasks that simplifies integration at customer side through an open control concept, daisy chaining of units, integrated temperature monitoring and access to calibration data for each unit. It includes an embedded IC together with 3 LED-Chips (e.g., red, green, and blue) in one package.

Daisy chaining allows the customer to control a large number of LEDs in serial connection. It is possible to control a single unit via a microcontroller with minimal wiring effort. Communication is based on a dedicated master-slave serial bus protocol, called open system protocol (OSP), which is detailed in chapter 5.2 "OSIRE® E3731i OSP specification".

Refer to chapter 3 "Basic operation" for a quick start guide.

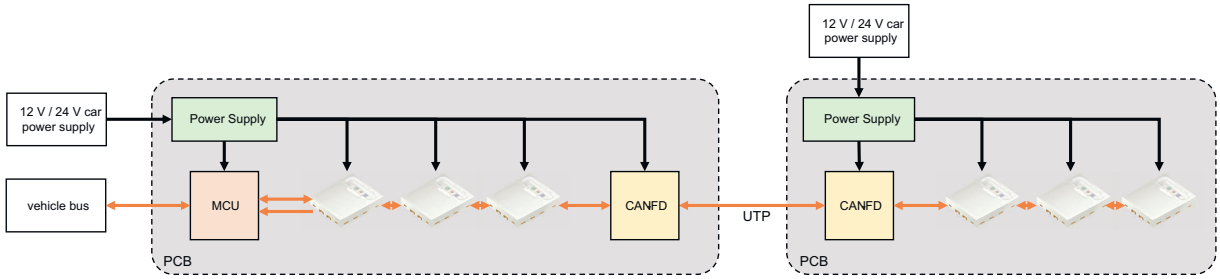
The product benefits are:

- High color setpoint accuracy over full gamut
- Flexible & large color gamut selectable via binning
- Highly accurate optical calibration data for all LEDs included
- Dynamic color control and animations
- PWM frequency up to 1 kHz
- High brightness over full gamut to allow visibility at daytime
- Temperature sensor included for full compensation of all three colors
- Completely open protocol, no license costs
- Auto-addressable included
- Diagnostics and error detection included in IC
- Bidirectional half-duplex and unidirectional loop-back modes
- Single-ended, LVDS and CAN-transceiver modes
- Less components on PCB needed, small PCB designs & high density of LEDs possible
- Single layer (flex) PCB designs possible
- Possibility to include other slave microcontrollers or IC components (e.g. sensing)

# 2 Application description

Figure 1 shows a typical application for dynamic ambient lighting with up to 1000 LEDs arranged on one or multiple PCBs. In this example, the first device in a daisy chain communicates with the master MCU via a dedicated single-ended interface. The downstream data is Manchester coded. The communication between two OSIRE® E3731i uses Manchester-coded LVDS for increased robustness and EMC performance.

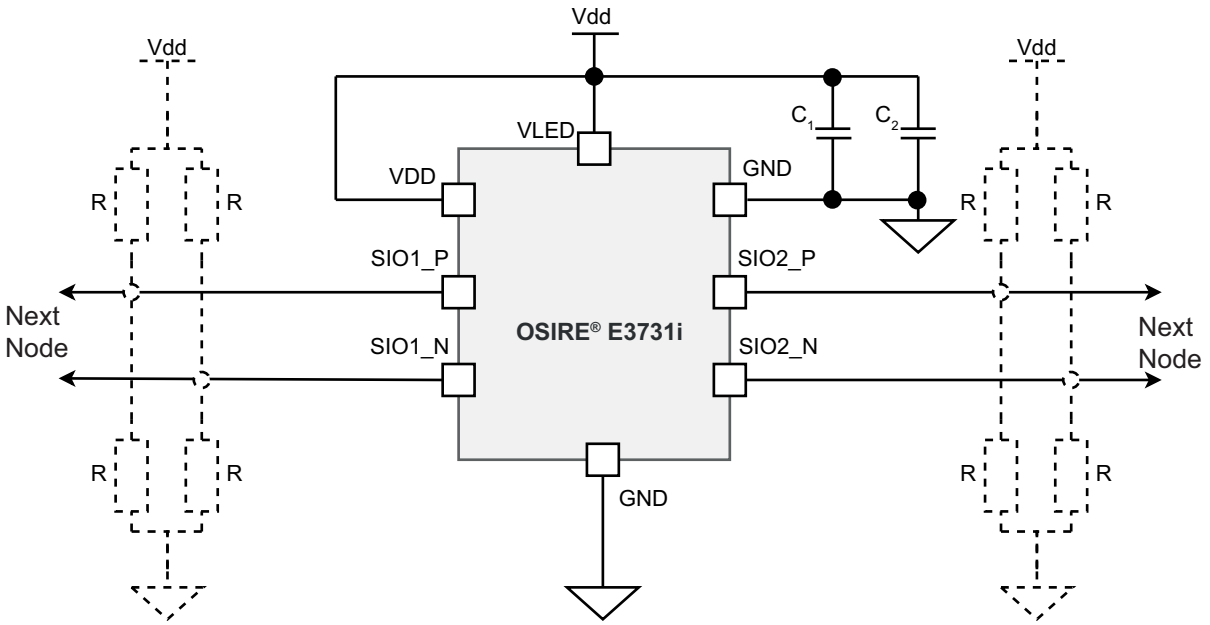
Figure 1: A typical application scenario (dynamic ambient lighting)




When the daisy chain extends over more than one PCB, the PCBs can be connected by using standard CAN-FD transceivers and UTP cables in a point-to-point fashion (no bus). The OSIRE® E3731i provides a dedicated CAN mode for communication with all standard CAN-FD transceivers supporting at least 4.8 Mbit/s. To allow a communication over cable between RGBi devices a connectivity to transceiver e.g. CAN FD via single ended Manchester coded communication is an option. The protocol will not change to a CAN protocol. Each PCB would require a dedicated on-board power supply.

The application diagram in Figure 2 shows mandatory and optional (dashed) components. The dashed resistors are optional and need to be chosen according to the communication mode. The two capacitors are needed for stabilization and HF rejection. Refer to chapter "Passive Components" for a selection guide for the passive components.

Figure 2: Application diagram



## 2.1 Communication mode selection

Each of the two I/O interfaces of the OSIRE® E3731i supports 4 different modes, that are selected through pull-up and pull-down resistors to GND and VDD, respectively. The communication mode selection will be done after a reset command or POR see chapter 3.3 "Startup behavior".

Table 1: Communication mode selection

| Mode | R @ SIOx_P | R @ SIOx_N | SIOx_P Input | SIOx_P Output | SIOx_N Input | SIOx_N Output |
|---|---|---|---|---|---|---|
| LVDS | (down) | (down) | ME | ME | ME | ME |
| MCU | up | down | ME | DATA | - | CLK |
| EOL | down | up | - | DATA | - | CLK |
| CAN | up | down | - | ME | ME | - |

ME --> Manchester encoded data
DATA --> Data only
CLK --> Clock

**LVDS mode:**

This mode is intended for communication between two OSIRE® E3731i. The two pull down resistors for LVDS are optional due to built-in 100 kΩ pull down resistors.

**MCU mode:**

This mode intends to simplify communication with standard MCUs by providing synchronous data and clock signals for upstream communication.

**EOL (end-of-line) mode:**

This mode signals a unit that it is the last in the chain which is needed for proper initialization. See chapter 5.2.5 "Commands" for more details.

**Note:** It is possible to connect an MCU in read-only fashion as needed for loop-back communication.

**CAN mode:**

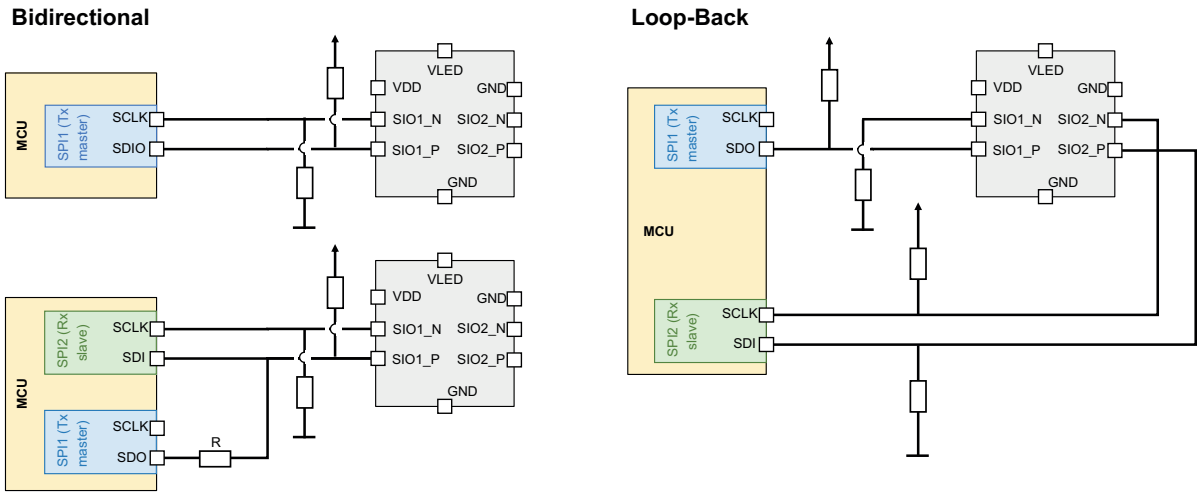
This mode is intended for communication with standard CAN-FD transceivers as is needed for extending the daisy chain over several PCBs.

## 2.2 Connection to MCU

Figure 3 illustrates basic connection options between the OSIRE® E3731i and an MCU. The two figures on the left show connection options suitable for bidirectional communication using a single SPI port (top) and using two SPI ports (bottom). If the single port option is used (top figure), the MCU should be sufficiently fast to switch between master and slave mode in order to be able to receive responses from the first unit in the chain (see chapter "Responses and status requests" for timing requirements). The resistor between SDO and SIOx_P in the bottom figure

has to be chosen together with the total capacitance and the pull-up and -down resistors to ensure voltage levels and signal rise- and fall times are within specification (see chapter "Passive Components"). Ideally, the SDO pin should be pulled low or set to a high impedance state while receiving data from the OSIRE® E3731i.

Figure 3: MCU connection schemes for bidirectional (left) and unidirectional (right) communication.




Some MCUs require a chip select (CS) signal for receiving messages in slave mode. A possible solution is to drive the CS pin of the SPI port with another GPIO pin of the same MCU. The OSIRE® E3731i expects the MCU the transmit data over SIOx_P using Manchester encoding. Any signals received on SIOx_N are ignored (Figure 9).

In the upstream direction, the OSIRE® E3731i transmits data synchronously with data on SIOx_P and clock on SIOx_N and a special "start" and "stop" sequence (Figure 10). The clock polarity can be inverted if needed (see chapter 5.2.5 "Commands"). Before and after the message, the SIOx_P and SIOx_N levels should be actively pulled to their idle states determined by the resistor configuration (MCU or EOL mode).

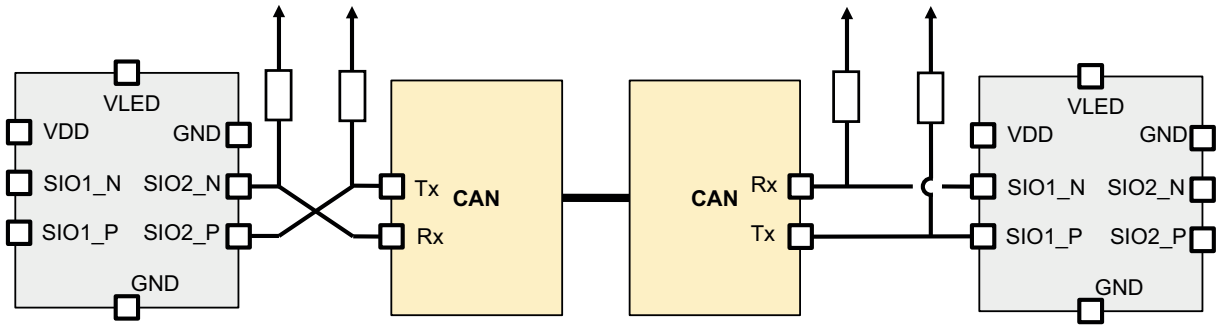
All single-ended connections should be kept as short as possible to ensure signal integrity and to lower EMC concerns.

## 2.3 Connection via CAN-FD

The OSIRE® E3731i has a dedicated communication mode for standard CAN-FD transceivers (Figure 4). This allows to extend the daisy chain over multiple PCBs in a point-to-point fashion using the physical layer of CAN-FD. This mode is selected by two pull-up resistors connected to the SIOx_P and SIOx_N pins. SIOx_P is used for output only and transmits data as Manchester encoded signals to the Tx-pin of the CAN-FD transceiver (Figure 12). During transmission, any signals received on SIOx_N are ignored. SIOx_N is used for input only and receives Manchester encoded data from the Rx-pin of the CAN-FD transceiver (Figure 11).

Due to the voltage rating of the SIOx pins, the use of CAN-FD transceivers is strictly necessary to comply with automotive transient stability requirements. As the OSIRE® E3731i does not support the CAN-FD protocol, a bus topology is not supported.

Figure 4: Connection scheme via CAN-FD transceivers.




While it is in principle possible to program the MCU in a way to directly communicate with the CAN-FD transceiver (see Figure 12), the MCU needs to implement a Manchester decoder.

All single-ended connections should be kept as short as possible to ensure signal integrity and to lower EMC concerns. The CAN-FD transceivers need to support a data rate of at least 4.8 Mbit/s. Additional components may be needed depending on the exact type of transceiver and the application requirements.

## 2.4 Connection via LVDS

Two OSIRE® E3731i should be connected using the LVDS mode, which is selected through two pull-down resistors. Due to the built-in 100-kΩ resistors, the external pull-down resistors are not necessary. The OSIRE® E3731i includes the termination resistor required for LVDS, so no external resistors are required.

The PCB layout should follow the usual layout guidelines for LVDS traces:

- Keep the wires parallel and close to minimize interference with radiated electromagnetic fields and noise injection.
- The wire lengths for the SIOx_P and SIOx_N pair should be balanced.
- The differential LVDS line impedance should be close to 200 Ω.
- Avoid or minimize stubs and junction taps to avoid reflections.
- Avoid right-angle traces to minimize radiated emission.

## 2.5 Component selection guide

The following list is intended only as a quick start guide and does not claim to be complete or suitable for automotive applications.

## Power Supply

The OSIRE® E3731i requires a typical supply voltage of 5 V and a variable supply current up to approximately 170 mA depending on the device state, current mode, brightness, and color point settings. Fore details, please refer to the OSIRE® E3731i datasheet.

In addition, wire resistance along the PCB can cause a severe voltage-drop between the first and the last unit in the chain. Care must be taken to ensure that the trace thickness is designed well enough for the application. The current flow depends on the number of devices and driving modes. A power supply concept on different positions to the application can be helpful.

## Passive Components

The supply filtering capacitors should be placed as closely as possible to the VDD pad.

**Table 2: Passive components**

| Parameter | Min | Typ | Max | Unit | Note |
|---|---|---|---|---|---|
| Stabilization capacitor on VDD | 1 | | | µF | C2 |
| HF filtering capacitor on VDD | 100 | | | nF | C1, low ESR recommended |
| Pullup & pulldown resistors* | 2.2k | 10k | 25k | Ω | |
| MOSI decoupling resistor | | 2.2k | | Ω | optional |

* Higher values up to 60 kΩ are possible when 3.3-V logic components are used.

## Microcontrollers

The OSIRE® E3731i supports microcontrollers and FPGAs that are 5 V tolerant and use 3.3 V or 5 V logic levels for I/O.

Examples include (but not restricted to):

- NXP® S32K144
- STMicroelectronics® STM32F446
- Microchip® ATSAMC21J18A
- Infineon® PSoCTM4

## Transceivers

The OSIRE® E3731i supports all standard CAN-FD transceivers that are 5 V tolerant and use 3.3 V or 5 V logic levels.

Examples include (but not restricted to):

- NXP® TJA1057GT/3
- NXP® TJA1442
- Infineon® TLE9251V_AEC

# 3 Basic operation

## 3.1 Example of basic operating sequence

The following shows an exemplary sequence for setting a static or dynamic, temperature-controlled color (gradient) on a chain of 100 LEDs:

**Setup Phase**

1. Connect supply voltage and wait until device is ready (~ 1 ms).
2. Send setup configuration via broadcast (e.g.enable crc check, SPI clock polarity)
3. Send INIT command (bidirectional or loop-back) and wait for response from last unit.
4. For every unit:
   - Read status and check for errors
   - Send basic configuration (diagnostic options, PWM frequency, ...)
   - *Read calibration data and store in the MCU* (only once after assembly)
   - Read temperature and store in the MCU
   - Go to ACTIVE mode

**Main Loop (execute every time a new input from the BCU is received)**

5. Get new input from body control unit (color target for every unit)
6. Calculate new PWM settings for every unit based on the stored data
7. Update PWM settings of every unit

**Stabilization loop (execute continuously)**

8. Read status and temperature from one unit
9. Check for errors
10. Calculate new PWM settings for this unit based on the new temperature
11. Update PWM settings of this unit
12. Proceed with next unit

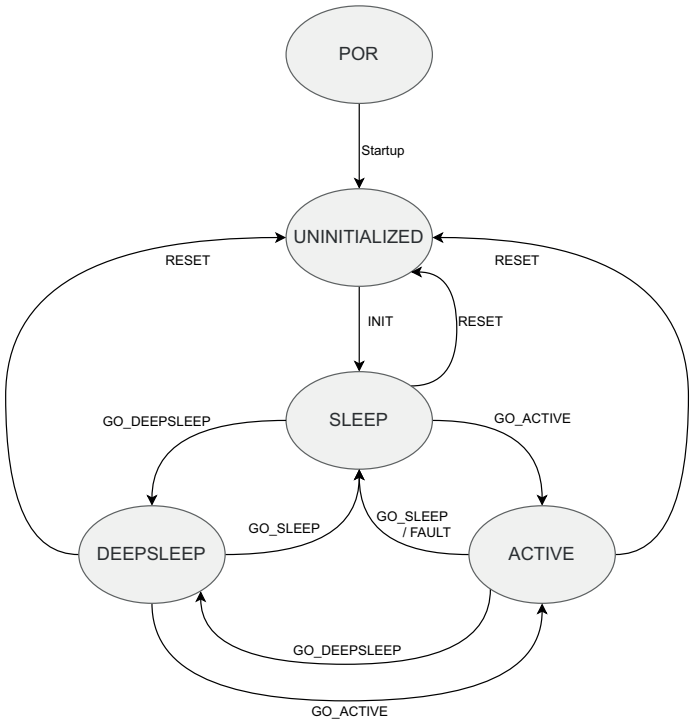
**Shutdown Phase**

13. Go to SLEEP mode
14. Disconnect the supply voltage

## 3.2 Device states

Figure 5 shows a simplified device state diagram with relevant transitions.

Figure 5: Device state diagram




**POR:** The IC is in an off state.

**UNINITIALIZED:** The OSIRE® E3731i will reach this state automatically after power-up. PWM values are 0 and LED drivers disabled. In this state, the OSIRE® E3731i only responds to a limited set of commands:

- INIT (bi-directional or loop back)
- RESET (broadcast)
- CLRERROR (broadcast)
- SETOTTH (broadcast)
- SETSETUP (broadcast)
- SETPWM (broadcast)

Other messages are ignored. Broadcast messages are fast forwarded and addressed messages are forwarded after the message has been fully received.

**SLEEP:** The OSIRE® E3731i enters SLEEP mode after receiving an INIT command, after a failure event (if selected) or via GO_SLEEP. In this state, the LED drivers are disabled but the

last PWM parameters are stored, i.e., not reset to 0. An update of the PWM parameters is possible.

**ACTIVE:** The OSIRE® E3731i enters this mode after receiving the GO_ACTIVE command if no fault conditions are present. In this state the LED drivers are enabled with the respective current setting, PWM frequency and duty cycle for each channel. An update of the PWM parameters is possible. After entering the ACTIVE state, the open/short detection is executed once for every channel with a PWM value exceeding the minimum value. See chapter 3.4 "Diagnostics".

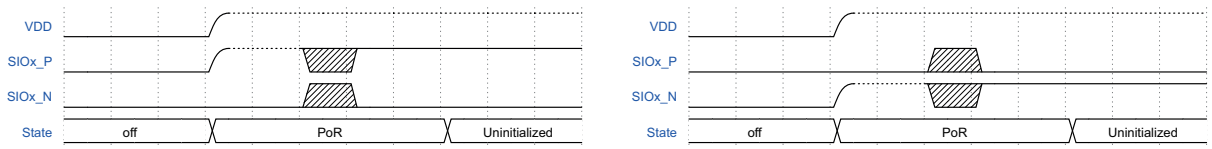
**DEEPSLEEP:** The OSIRE® E3731i is in a power save mode with reduced current consumption. This will be a power save mode for future IC's. The command is implemented in the OSIRE® E3731i too but the current consumption is similar to the sleep mode.

## 3.3 Startup behavior

When the supply voltage is connected and VDD increases above the VDDgood level (see datasheet), or the RESET command is received, the OSIRE® E3731i undergoes the following startup sequence (Figure 6):

1. All registers are initialized to their default state (POR in Figure 6).
2. Trim settings are loaded and checked for integrity. If a mismatch is detected, the OTPCRC flag in the STATUS register is set.
3. The voltage levels at each SIOx pin are checked to detect the communication modes (shaded signals in Figure 6). The SIOx_P and SIOx_N pins are shorted together for a period of ~ 40 µs.
4. The OSIRE® E3731i enters the UNINITIALIZED mode and is now ready to receive the first command (Uninitialized in Figure 6).

**Figure 6: Illustration of the startup behavior of the OSIRE® E3731i. The example shows the MCU (left) and EOL (right) modes. After a RESET command, the device starts in the POR state.**




The whole sequence takes approximately 1 ms.

**Note:** The IO detection could be interpreted as clock edge on the MCU input if the SPI interface is already enabled.

## 3.4 Diagnostics

The OSIRE® E3731i provides built-in diagnostic functions for automatic detection of fault conditions. If a failure is detected, the respective failure flag bit is set. The failure flag bit can only be cleared when the STATUS register has been read by the MCU (either directly or via status request bit), all flag bits have been cleared using CLRERROR, or the unit has received a RESET command. See chapter 5.2.5 "Commands" for more details.

If selected, the OSIRE® E3731i will automatically turn off the LED drivers when a failure is detected and change to SLEEP state to prevent damage to the part. This helps to ensure that communication is always possible through the entire daisy-chain. See chapter 5.2.5 "Commands" for more details.

**Overtemperature:** This error occurs when the IC junction temperature rises above the overtemperature threshold. It can only be cleared once the temperature has fallen below the overtemperature release threshold. See chapter 5.2.5 "Commands" for details on controlling the threshold values.

**Undervoltage:** This error occurs when the supply voltage of the OSIRE® E3731i drops below the undervoltage threshold for a period longer than the deglitch time. Refer to the OSIRE® E3731i datasheet for more details.

**Driver open / short:** This error occurs when the LED driver pin voltage remains above the LED-open threshold voltage or falls below the LED-short threshold voltage for a period longer than the deglitch time. These error conditions are tested individually for each channel, but only once after the GOACTIVE command has been received and the PWM value of the channel is set to a value higher than the minimum value. In active modes, a PWM change below the minimum value and then back to a value higher than the minimum value, leads to a new open/short check. Please refer to the OSIRE® E3731i datasheet for more details.

**Note:** Due to the phase shift of the LED driver and depending on the PWM frequency setting, the open/short detection of R,G,B needs some time (<1ms) after the GOACTIVE command. It is recommend to set open/short flag will lead to sleep in the setup register.

**Communication:** See chapter "Communication errors" for communication errors.

# 4 Color control and temperature stabilization

The OSIRE® E3731i provides built-in calibration data for every LED at both day- and nighttime current settings as well as a built-in temperature sensor in the IC. Together, this allows a highly accurate color control over the full color gamut and operating temperature range.

## 4.1 Calibration data

Calibration data is stored as encoded (u', v', Iv) triplets for every LED and both current settings in the built-in nonvolatile memory. Once the encoded values, $X_{Iv}$, $X_u$, and $X_v$, have been loaded

by the MCU, the physical color coordinates can be calculated as followed with the current mode settings from Table 3.

$I_v = LSB_{Iv}\ (M) \times X_{Iv}$

$u' = LSB_u\ (M) \times X_u$

$v' = LSB_v\ (M) \times X_v$

Table 3: Current mode settings

| Current Mode (M) | $LSB_{IV}$ | $LSB_u$, $LSB_v$ |
|---|---|---|
| Nighttime Mode | 0.44 mcd | 0.0025 |
| Daytime Mode | 1.25 mcd | 0.0025 |

See chapter 5.2.5 "Commands" on how to retrieve the data from the memory and chapter 5.3 "OTP to PWM example" for an example of the decoding procedure.

## 4.2 PWM calculation

Any color within the accessible gamut for an RGB triplet can be realized by a linear combination of the individual spectra of the three LEDs. Scaling of the intensity of the single LEDs is conveniently done by changing the effective duty cycle of the LEDs through the PWM.

In order to create a mixed color with tristimulus values $X_T$, $Y_T$ and $Z_T$, the PWM duty cycles may be calculated as outlined below.

1. Create the target tristimulus vector from the Cx/Cy or u'/v' color target:

$$T = \begin{bmatrix} X_T \\ Y_T \\ Z_T \end{bmatrix} = I_{v,\,target} \begin{bmatrix} \frac{C_x}{C_y} \\ 1 \\ \frac{1 - C_x - C_y}{C_y} \end{bmatrix} = I_{v,\,target} \begin{bmatrix} \frac{9u'}{4v'} \\ 1 \\ \frac{12 - 3u' - 20v'}{4v'} \end{bmatrix}$$

2. Calculate the tristimulus matrix of the RGB triplet using the correct calibration data for the used current mode setting:

$$A = \begin{bmatrix} X_R & X_G & X_B \\ Y_R & Y_G & Y_B \\ Z_R & Z_G & Z_B \end{bmatrix} = \frac{1}{4} \cdot \begin{bmatrix} 9I_{v,R}\frac{u'_R}{v'_R} & 9I_{v,G}\frac{u'_G}{v'_G} & 9I_{v,B}\frac{u'_B}{v'_B} \\ 4I_{v,R} & 4I_{v,G} & 4I_{v,B} \\ I_{v,R}\frac{12 - 3u'_R - 20v'_R}{v'_R} & I_{v,G}\frac{12 - 3u'_G - 20v'_G}{v'_G} & I_{v,B}\frac{12 - 3u'_B - 20v'_B}{v'_B} \end{bmatrix}$$

3. Solve the linear equation

Ax = T

for the solution vector x = $(D_r, D_g, D_b)^T$, which yields the duty cycles (in a range from 0 to 1) needed for each LED:

$$D_r = \frac{det(A1)}{det(A)}$$

$$D_g = \frac{det(A2)}{det(A)}$$

$$D_{br} = \frac{det(A3)}{det(A)}$$

A1, A2, and A3 are 3x3 matrices obtained by replacing column 1, 2, or 3, respectively, by the target vector and the determinant of a 3x3 matrix is generally given by

$det(A) = A_{11} A_{22} A_{33} + A_{12} A_{23} A_{31} + A_{13} A_{21} A_{31} - A_{12} A_{21} A_{33} - A_{13} A_{22} A_{31} - A_{23} A_{32} A_{11}$

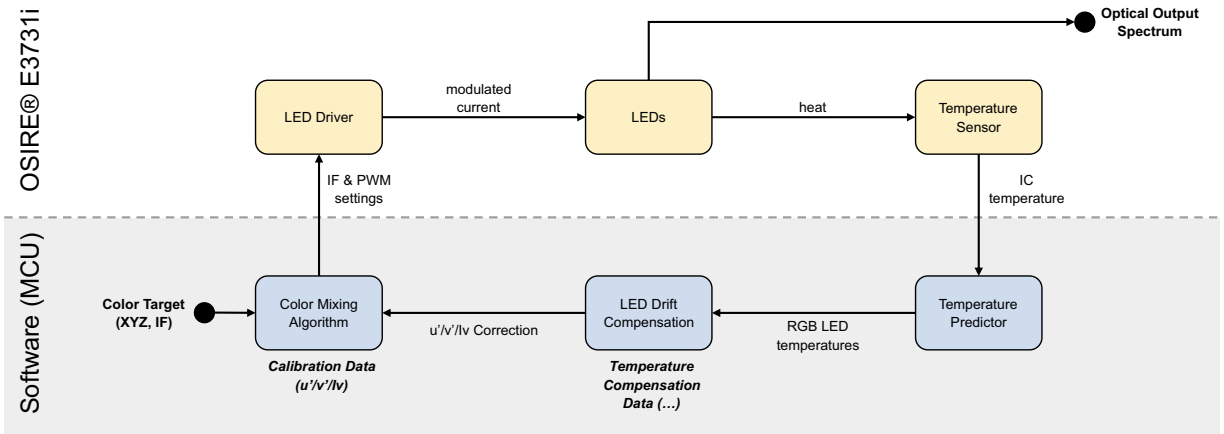
Finally, for the SETPWM command, these duty cycles have to be multiplied by the maximum PWM value for the chosen PWM frequency: $2^{15} - 1 = 32767$. Please use the same PWM setting for 1kHz PWM frequency mode, because it will be changes internally to a $2^{14}$ bit resolution (see PWM_F in chapter 5.2.5 "Commands"). See chapter 5.2.5 "Commands" on how to program the PWM and current values and chapter 5.3 "OTP to PWM example" for a worked example.

## 4.3 Temperature stabilization

The built-in temperature sensor of the OSIRE® E3731i allows various ways to realize a highly accurate color control over the full color gamut and operating temperature range. Two examples are provided in the following.

The basic idea of a temperature stabilization loop is given in Figure 7. Yellow boxes are implemented in the OSIRE® E3731i, blue boxes have to be implemented on customer side.

Figure 7: Basic idea of temperature stabilization loop



### 4.3.1 Temperature sensor

The OSIRE® E3731i features a built-in temperature sensor with a resolution of 8 bit that offers accurate and stable reading over the full temperature range.The return value, ADC, can be converted to a chip temperature in °C by

$T_{IC}$ [°C] = 1.08 x ADC - 126

**Note:** As the calibration temperature, $T_{cal}$, corresponds to a fixed ADC value of 140, the equations for the temperature compensation are given in terms of ADC counts in the following.

The temperature sensor can be read out during SLEEP and ACTIVE modes.

### 4.3.2 Temperature prediction

The returned temperature reading corresponds to the junction temperature of the IC. In a first approximation, this value can be used instead of the LED temperatures for the temperature compensation functions (applies also to ADC counts):

$T_x = T_{IC}$

For a more accurate color stabilization, however, the junction temperatures of the LEDs are required, as they may differ by a few °C from the IC temperature depending on the current mode and chosen color point. The temperatures have to be converted between °C and ADC counts using the equation given in chapter 4.3.1 "Temperature sensor"

### 4.3.3 Simple temperature compensation - I

A good temperature stabilization can be achieved by just compensating the loss of brightness of the red LED, either using a simple linear factor (these values still represent an early status and might be subject to change)

$$PWM_{comp} \approx [1 + 0.005\ (T - T_{cal})] \times PWM(T_{cal})$$

or a look-up table

$$PWM_{comp} \approx A(I_F, T) \times PWM(T_{cal})$$

where $A(I_F, T)$ has to be interpolated from Table 4

Table 4: Temperature compensation look-up table

| Temperature (ADC counts) | A(T) for nighttime mode | A(T) for daytime mode |
|---|---|---|
| 73 | 0.84 | 0.79 |
| 93 | 0.88 | 0.85 |
| 103 | 0.90 | 0.87 |
| 113 | 0.92 | 0.91 |
| 140 | 1.00 | 1.00 |
| 163 | 1.11 | 1.13 |
| 198 | 1.33 | 1.40 |
| 213 | 1.42 | 1.51 |
| Note that these values still represent an early status and might be subject to change | | |

### 4.3.4 Simple temperature compensation - II

Better compensation is achieved by using the look up Table 5 and Table 6 to compensate all three LEDs. These tables are optimized for D65.

Table 5: Nighttime mode

| Temperature (ADC counts) | A(T) for Red | A(T) for Green | A(T) for Blue |
|---|---|---|---|
| 73 | 0.71 | 0.85 | 0.90 |
| 93 | 0.78 | 0.89 | 0.93 |
| 103 | 0.82 | 0.92 | 0.94 |
| 113 | 0.86 | 0.94 | 0.96 |
| 140 | 1.00 | 1.00 | 1.00 |
| 163 | 1.18 | 1.07 | 1.05 |
| 198 | 1.57 | 1.18 | 1.16 |
| 213 | 1.83 | 1.23 | 1.21 |
| Note that these values still represent an early status and might be subject to change | | | |

Table 6: Daytime mode

| Temperature (ADC counts) | A(T) for Red | A(T) for Green | A(T) for Blue |
|---|---|---|---|
| 73 | 0.69 | 0.90 | 0.89 |
| 93 | 0.76 | 0.92 | 0.93 |
| 103 | 0.81 | 0.93 | 0.94 |
| 113 | 0.85 | 0.95 | 0.96 |
| 140 | 1.00 | 1.00 | 1.00 |
| 163 | 1.20 | 1.06 | 1.05 |
| 198 | 1.64 | 1.17 | 1.13 |
| 213 | 1.93 | 1.22 | 1.22 |
| Note that these values still represent an early status and might be subject to change | | | |

### 4.3.5 Optimized temperature compensation functions

The temperature dependence of the calibration parameters is very well described by the quadratic function:

$f(T) = [a(T - T_{cal})^2 + b(T - T_{cal})+1] \times f(T_{cal})$

where

f = placeholder for u', v', and $I_v$

T = junction temperature of the LED

$T_{cal}$ = calibration temperature (ADC code = 140)

a, b = constants

Together with the stored calibration data, $f(T_{cal})$, this allows to calculate the calibration data needed for the calculation of the PWM values at any temperature. The values for the constants are given in Table 7.

Table 7: PWM values

| Current mode | Chip | Parameter | a | b |
|---|---|---|---|---|
| nighttime | red | I_v_T | 1.030E-05 | -7.397E-03 |
| | | u_prime_T | -1.653E-06 | 4.029E-04 |
| | | v_prime_T | 2.567E-07 | -6.984E-05 |
| | green | I_v_T | -5.864E-07 | -2.306E-03 |
| | | u_prime_T | 4.555E-06 | 2.081E-03 |
| | | v_prime_T | -8.166E-08 | -1.921E-05 |
| | blue | I_v_T | -2.679E-06 | 7.200E-04 |
| | | u_prime_T | -1.060E-06 | -6.739E-04 |
| | | v_prime_T | 5.774E-06 | 2.434E-03 |
| daytime | red | I_v_T | 1.122E-05 | -7.808E-03 |
| | | u_prime_T | -1.466E-06 | 3.727E-04 |
| | | v_prime_T | 2.475E-07 | -6.384E-05 |
| | green | I_v_T | -5.946E-06 | -1.813E-03 |
| | | u_prime_T | 5.878E-06 | 2.241E-03 |
| | | v_prime_T | -1.897E-07 | 1.981E-05 |
| | blue | I_v_T | -2.226E-06 | 7.734E-04 |
| | | u_prime_T | -1.036E-06 | -6.427E-04 |
| | | v_prime_T | 6.519E-06 | 2.423E-03 |
| Note that these values still represent an early status and might be subject to change | | | | |

Example:

Corrected $I_v$ for the red LED in nightmode at a temperature of 87.8°C (ADC = 198):

$$I_v\,(85°C)=[1.030\times10^{-5}\,(198-140)^2 - 7.397 \times 10^{-3}\,(198-140) + 1] \times I_{(v,cal.)}=0.606\times I_{(v,cal.)}$$

# 5 Appendix

## 5.1 Glossary

Table 8 shows the glossary for this documentation and explains the terms with a description.

Table 8: Glossary

| Term | Description |
|---|---|
| BCU | body control unit |
| CAN (FD) | controller area network (flexible data-rate) |
| DMA | direct memory access |
| EMC | electromagnetic compatibility |
| ESR | equivalent series resistance |
| EOL | end of line |
| ASIC | application-specific integrated circuit |
| LED | light emitting device |
| LSB | least significant bit; also used as unit for the quantization step size |
| MCU | microcontroller (unit) |
| MSB | most significant bit |
| NOP | no operation |
| NRZ | non return to zero |
| OSP | open system protocol |
| PCB | printed circuit board |
| POR | power-on reset |
| PWM | pulse-width modulation |
| RGB | red, green, blue |
| SPI | serial peripheral interface |
| UTP | unshielded twisted pair |
| 0x”value” | Number in hexadecimal format, e.g., 0xff = 255 |
| 0b”value” | Number in binary format, e.g., 0b101 = 5 |

## 5.2 OSIRE® E3731i OSP specification

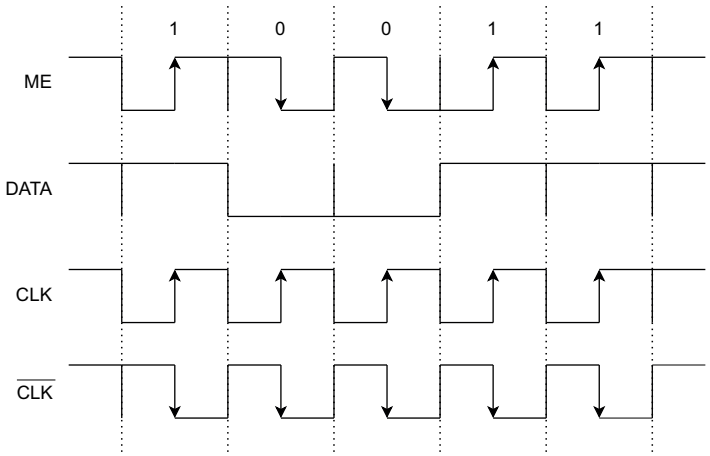
Communication is based on a dedicated master-slave serial bus protocol, called open system protocol (OSP). In addition to this base protocol, the OSIRE® E3731i implements further dedicated commands. This section provides a short overview over the OSP and the supported commands.

### 5.2.1 Message encoding

#### Manchester encoding

Figure 8 shows the different encodings used by the OSIRE® E3731i. Where applicable, Manchester encoding is based on the IEEE 802.3 standard, where a logical "1" is encoded by a rising edge and a logical "0" by a falling edge.

Figure 8: Different encodings used by the OSIRE® E3731i




In the case of single-ended communication from the OSIRE® E3731i to the MCU, the clock polarity can be selected by a command (see chapter 5.2.5 "Commands"). The default setting uses the rising edge. Refer to chapter 5.4 "Code examples" for examples.

#### Example waveforms

The internal signals are for reference only. The internal clock indicates the length of a data bit.

Figure 9: Example showing communication from MCU to OSIRE® E3731i

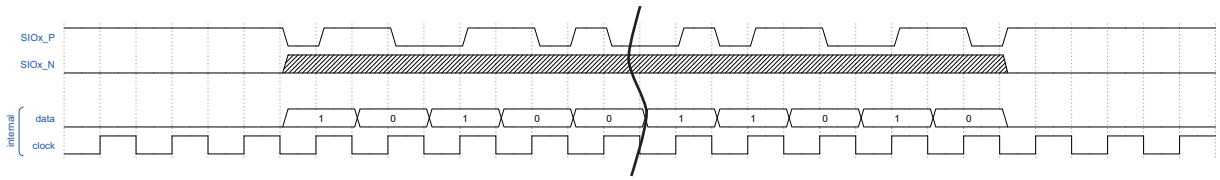


Figure 10: Example showing communication from OSIRE® E3731i to the MCU with normal and inverted clock polarity.

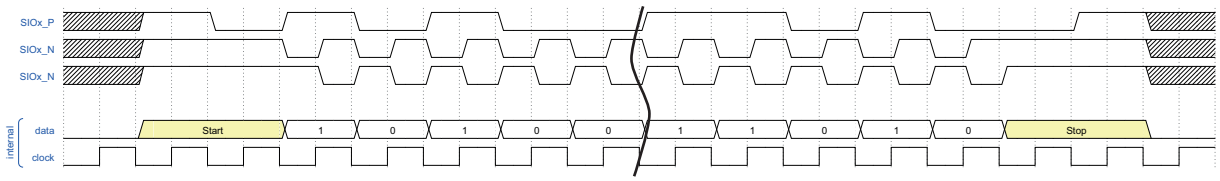



Figure 11: Example showing communication from CAN to OSIRE® E3731i

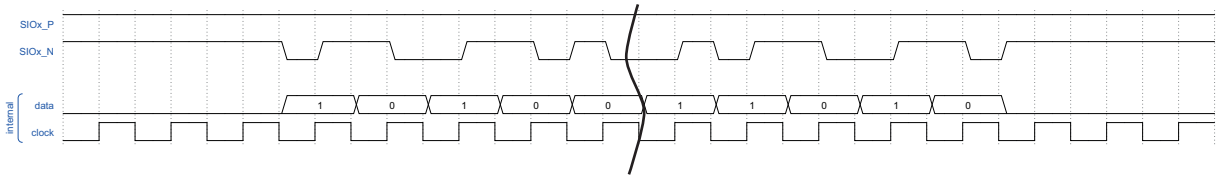



Figure 12: Example showing communication from OSIRE® E3731i to CAN

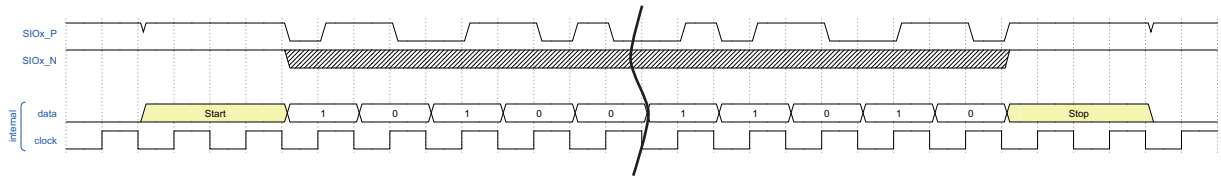
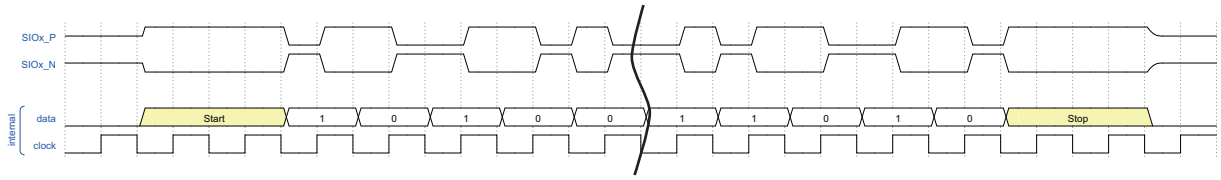


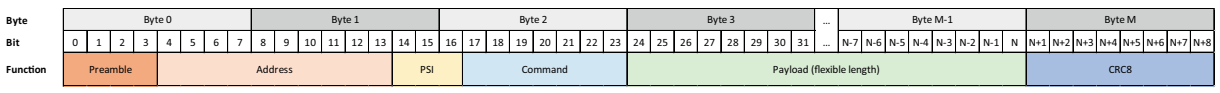

Figure 13: Example showing communication between two OSIRE® E3731i using LVDS




## 5.2.2 Message frame format

The OSP is a message-based protocol with the general message frame format as shown in Figure 14.

Figure 14: general message frame format




The bit order is MSB first. The format is the same for upstream and downstream communication.

**Preamble (4 bit):** Fixed to 0b1010.

Avoid messages with incorrect preamble as this could lead to forwarding of garbage messages along the chain and to raising the CE_FLAG.

**Address (10 bit):** Allowed values for downstream messages are 0x000 (broadcast) and 0x001 to 0x3ef (individual nodes). Upstream messages always contain the address of the sender in the address field. Not all commands are allowed in a broadcast message. See chapter 5.2.5 "Commands".

**PSI (3 bit):** The PSI field indicates the length of the payload in bytes. Its value must match the expected value for each command (may be different for up- and downstream directions). See chapter 5.2.5 "Commands".

Table 9: Allowed values for PSI field

| PSI | 0b000 | 0b001 | 0b010 | 0b011 | 0b100 | 0b101 | 0b110 | 0b111 |
|---|---|---|---|---|---|---|---|---|
| Payload | 0 | 1 | 2 | 3 | 4 | reserved | 6 | 8 |

The total message length is PSI + 4 bytes.

**Note:** The PSI 0b101 is not allowed and will give rise to a communication error even when the unit is not directly addressed.

**Note:** A PSI value larger than the payload will potentially lead to the payload being filled up with garbage data from previous messages. In this case, no CE_FLAG is raised.

**Command (7 bit):** See chapter 5.2.5 "Commands".

**Payload (0 to 64 bit):** Variable payload field. The length must match the value given in the PSI field.

**CRC (8 bit):** See chapter 5.2.4 "Cyclic redundancy check"

### 5.2.3 Message handling

#### Broadcast

An address of 0x000 initiates a broadcast, i.e., the same message is sent and received by all nodes on the chain. Whether a command can be broadcast is detailed in chapter 5.2.5 "Commands".

Commands that trigger a response and status requests are not allowed in broadcast messages.

If a response is requested nevertheless (READxx or SR), only the last unit in the chain (EOL) sends out a response message. This behavior is not supported and subject to change without notice.

#### Forwarding and fast forwarding

Message forwarding refers to the direct forwarding of a message without modification.

Messages are forwarded to the next device if the address in the message frame is the broadcast address, or it is different from the node's address.

If a message is to be forwarded, forwarding starts exactly one cycle after the preamble and address field are fully received, i.e., after 15 bits. This is referred to as fast-forwarding.

When a node is uninitialized, broadcast messages are fast-forwarded and addressed messages (0x001 to 0x3fe) are forwarded only after the full message has been received.

### Responses and status requests

Some commands trigger a response from the addressed node. In this case, the address field contains the address of the sending node to indicate the origin of the message. The response is typically sent out approximately 3-4 bit-periods after the end of the command message has been received by the node.

Some commands do not send a response but allow a status request. The status request is done by setting bit 5 in the command field ("SR" bit):

```
uint8_t commandSR = command | (1 << 5);
```

If a valid "SR" bit has been received, the node responds by sending the TEMPSTAT register to the MCU after the original command has been executed. Status bits are cleared after responding to the status request. In the case of the CLRERROR command, the status bits are cleared by the command before the status request is sent.

Commands supporting a status request are marked in the command description. See chapter 5.2.5 "Commands".

**Note**: If the SR bit is set in commands officially not supporting the status request, the command is executed but the TEMPSTAT register is returned to the MCU instead of the expected response of the command. This does not apply to the INITxx commands.

**Note:** The OSP does not allow simultaneous up- and downstream messages on the same node, as this could lead to dropped or corrupted messages. Therefore, commands that trigger a response and status requests are not allowed in broadcast messages.

If a response is requested nevertheless (READxx or SR), only the last unit in the chain (EOL) sends out a response message. This behavior is not supported and subject to change without notice.

### Waiting time between messages

The OSIRE® E3731i is not accepting a new message while a message is being received or transmitted on the other port. Likewise, simultaneous upstream and downstream messages are not supported.

Therefore, a waiting time has to be added by the MCU between sending two messages in order to ensure that all receiving and transmitting activities have been concluded.

The minimum waiting time between two downstream messages should be 8.5 µs (edge to edge) and applies to write-only and broadcast commands without response request. If an answer has been requested, the next message should only be sent after the response has been received by the MCU.

### Communication errors

The OSP requires certain communication errors to be detected by each unit in a daisy chain. Table 10 summarizes all relevant error conditions.

Table 10: Relevant error conditions

| Unit addressed | Correct preamble | PSI matches payload | CRC is correct* | PSI matches command | CE-FLAG raised |
|---|---|---|---|---|---|
| - | no | - | - | - | yes |
| yes | yes | - | - | no | yes |
| yes | yes | - | no | - | yes |
| yes | yes | no | - | - | yes |
| no | yes | - | - | no | no |
| no | yes | - | no | - | yes |
| no | yes | no | - | - | yes |

- value not relevant
* applies if CRC check is enabled
** applies to broadcast messages that address another unit

The OSIRE® E3731i performs some error detection even for messages that are to be forwarded (CRC and PSI). If an error is detected in such a case, the failure flag is set and the message is still forwarded, i.e., communication is not stopped. Errors due to unknown commands or PSI-command-mismatch are only detected if the unit is directly addressed.

Note that messages with a PSI to payload mismatch not addressed to a unit will be forwarded with a payload matching the PSI value leading to either truncation or extension with undefined data. For example, a missing or very bad GND connection between two devices could lead to such behavior.

## 5.2.4 Cyclic redundancy check

The CRC field contains the CRC8 check word calculated over the full message excluding the CRC field.

CRC Implementation: x^8 + x^5 + x^3 + x^2 + x + 1.

Polynomial: 0x2F (normal notation)

Initial value: 0x00

XOR value: 0x00

Direction: MSB first

**Note:** This implementation uses the same polynomial but is not compatible with the AUTOSAR implementation because of the different initial and XOR values of 0xFF.

**Note:** The calculation direction for the CRC field in the non-volatile memory is LSB first.

Refer to section 0 for examples.

### 5.2.5 Commands

The OSIRE® E3731i supports the following commands.

Table 11: Table of commands

| Command | Encoding | Broadcast possible | Payload size (bytes) | Response payload size (bytes) | Status request allowed* |
|---|---|---|---|---|---|
| RESET | 0x00 | yes | 0 | 0 | no |
| CLRERROR | 0x01 | yes | 0 | 0 | yes |
| INITBIDIR | 0x02 | no | 0 | 2 | no |
| INITLOOP | 0x03 | no | 0 | 2 | no |
| GOSLEEP | 0x04 | yes | 0 | 0 | yes |
| GOACTIVE | 0x05 | yes | 0 | 0 | yes |
| GODEEPSLEEP | 0x06 | yes | 0 | 0 | yes |
| IDENTIFY | 0x07 | no | 0 | 4 | no |
| READSTATUS | 0x40 | no | 0 | 1 | no |
| READTEMPST | 0x42 | no | 0 | 2 | no |
| READCOMST | 0x44 | no | 0 | 1 | no |
| READLEDST | 0x46 | no | 0 | 1 | no |
| READTEMP | 0x48 | no | 0 | 1 | no |
| READOTTH | 0x4A | no | 0 | 3 | no |
| SETOTTH | 0x4B | yes | 3 | 0 | yes |
| READSETUP | 0x4C | no | 0 | 1 | no |
| SETSETUP | 0x4D | yes | 1 | 0 | yes |
| READPWM | 0x4E | no | 0 | 6 | no |
| SETPWM | 0x4F | yes | 6 | 0 | yes |
| READOTP | 0x58 | no | 1 | 8 | no |

* See chapter "Responses and status requests". The payload size of a status request is 2 bytes. Not allowed with broadcast.

If the OSIRE® E3731i receives a command that is not listed above with PSI > 0, a communication error is raised. If the PSI is zero, the message is silently ignored.

**Note:** If the SR bit is set in commands officially not supporting the status request, the command is ignored and the TEMPSTAT register is returned to the MCU instead.

### RESET

Performs a complete reset of one or all devices. The effect is identical to a power cycle (see chapter 3.3 "Startup behavior").

All register values are set to their default values, all error flags are cleared, the communication mode detection is restarted, LED drivers are turned off, and the address is set to 0x3ff. The device enters the UNINITIALIZED mode.

After issuing the RESET command, wait for at least 150 µs before sending out the next command.

### CLRERROR

Clears all error flags in one or all devices. If an error condition still persists, the error flag is automatically set again.

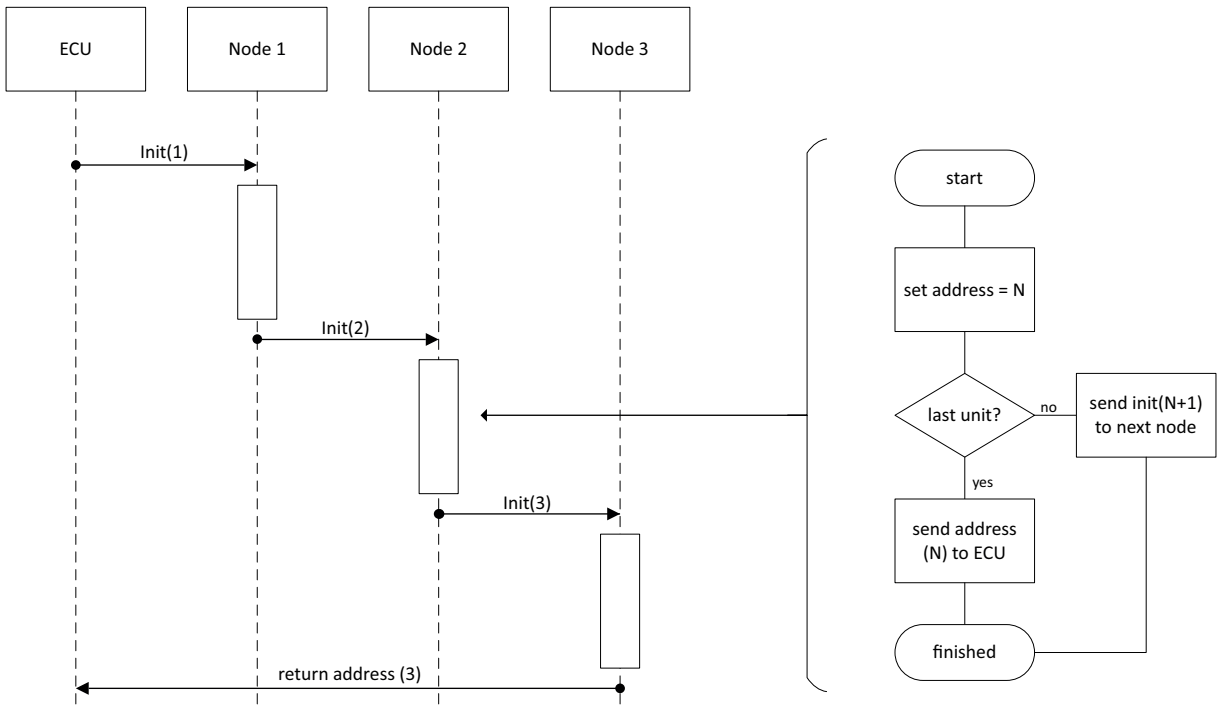
If the SR bit is set, the device returns the TEMPSTAT register after the error flags have been cleared.

### INITBIDIR

Initiates the automatic addressing of the chain and sets the communication direction to bidirectional.

The command shall be addressed to the first unit in the chain always with address 0x001. The last unit in the chain (indicated by the EOL mode, see chapter 2.1 "Communication mode selection") returns its address to the master. Subsequent commands should only be sent after the response has been received by the MCU.

Figure 15: Illustration of the automatic addressing feature



If the maximum number of node addresses has been reached (0x3fe), the unit stops the initialization procedure and returns its address (0x3fe) to the master.

If a node is already initialized, the command is ignored but the message is forwarded with incremented address and the last unit returns its address. This can be used to selectively initialize individual units in the chain after a RESET or POR. In this case, do not initialize the chain with an address different from 0x001 as this could cause undefined behavior.

**Note:** When the last unit is not in EOL mode (e.g., CAN-FD), it will not recognize itself as last unit. Then, the application needs to ensure that the chain length is correctly received by the master.

Uninitialized nodes interpret this command.

**Note:** This command is not allowed as broadcast. If the broadcast address is used nevertheless, all units in the chain will be initialized to address 0x000 and no response message will be sent. Proper operation can be restored with a broadcast RESET followed by INITxx with address 0x001.

**Note:** If the command is sent using the broadcast address 0x000 into an already addressed chain, all units will set the CE_FLAG and the last unit will return the TEMPSTAT register with address 0x000.

#### INITLOOP

Same as INITBIDIR but sets the communication mode to loop-back. The response to the master is sent in the forward direction.

**Note:** If the command is sent using the broadcast address 0x000 into an already addressed chain, the last unit will return the TEMPSTAT register with address 0x000. No CE_FLAG is raised in this case.

#### GOSLEEP

Sends one or all devices into SLEEP state. See chapter 3.2 "Device states".

#### GOACTIVE

Sends one or all devices into ACTIVE state. See chapter 3.2 "Device states".

#### GODEEPSLEEP

Sends one or all devices into DEEPSLEEP state. See chapter 3.2 "Device states".

#### IDENTIFY

Returns the 32-bit node type identification code (= 0x00000000).

#### READSTATUS

Returns the STATUS register as shown in Table 12.

Table 12: READSTATUS register

| Bit | <7> | <6> | <5> | <4> | <3> | <2> | <1> | <0> |
|---|---|---|---|---|---|---|---|---|
| Name | STATE [1:0] | | OTPCRC | COM | CE_FLAG | LOS_FLAG | OT_FLAG | UV_FLAG |
| Default | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

where

Table 13: READSTATUS description

| Status | Description |
|---|---|
| STATE [1:0] | Device state:<br>0b00 - UNINITIALIZED<br>0b01 - SLEEP<br>0b10 - ACTIVE<br>0b11 - DEEPSLEEP |
| OTPCRC | OTP error flag set during startup when a mismatch between the OTP values and the internal CRC is detected. Cleared after readout. |
| COM | Communication direction:<br>0b0 - bidirectional<br>0b1 - loop-back |
| CE_FLAG | Communication fault flag. Cleared after readout. |
| LOS_FLAG | LED fault flag. Cleared after readout. See LEDST register. |
| OT_FLAG | Overtemperature fault flag. Cleared after readout. |
| UV_FLAG | Undervoltage fault flag. Cleared after readout. |

The status flags are cleared after the response is sent.

### READTEMPST

Returns the STATUS + TEMP registers in a single 2-byte payload. See READSTATUS and READTEMP.

### READCOMST

Returns the COM STATUS register as shown in Table 14

Table 14: COM STATUS register

| Bit | <7> | <6> | <5> | <4> | <3> | <2> | <1> | <0> |
|---|---|---|---|---|---|---|---|---|
| Name | reserved | | | | SIO2_STATE[1:0] | | SIO1_STATE[1:0] | |
| Default | 0 | | | | X | X | X | X |

where

Table 15: COM STATUS description

| Status | Description |
|---|---|
| SIO2_STATE [1:0] | Communication mode of SIO2:<br>0b00 - LVDS<br>0b01 - EOL<br>0b10 - MCU<br>0b11 - CAN<br>Value is set automatically at startup and after reset. |
| SIO1_STATE [1:0] | Communication mode of SIO1:<br>0b00 - LVDS<br>0b01 - EOL<br>0b10 - MCU<br>0b11 - CAN<br>Value is set automatically at startup and after reset. |

## READLEDST

Returns the LED STATUS register as shown in Table 16.

Table 16: LED STATUS register

| Bit | <7> | <6> | <5> | <4> | <3> | <2> | <1> | <0> |
|---|---|---|---|---|---|---|---|---|
| Name | reserved | RO | GO | BO | reserved | RS | GS | BS |
| Default | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

where

Table 17: LEDSTATUS description

| Status | Description |
|---|---|
| RO | RED channel open fault flag. Cleared after readout. |
| GO | GREEN channel open fault flag. Cleared after readout. |
| BO | BLUE channel open fault flag. Cleared after readout. |
| RS | RED channel short fault flag. Cleared after readout. |
| GS | GREEN channel short fault flag. Cleared after readout. |
| BS | BLUE channel short fault flag. Cleared after readout. |

The flags are cleared after the response is sent.

## READTEMP

Returns the TEMP register as shown in Table 18

Table 18: TEMP register

| Bit | <7> | <6> | <5> | <4> | <3> | <2> | <1> | <0> |
|---|---|---|---|---|---|---|---|---|
| Name | TEMP[7:0] | | | | | | | |
| Default | X | | | | | | | |

where

Table 19: TEMP register description

| Status | Description |
|---|---|
| TEMP[7:0] | IC Temperature: T(°C) = 1.08 x TEMP - 126 |

## READOTTH

Returns the OTTH register as shown in Table 20.

Table 20: OTTH register

| Bit | <23> | <22> | <21> | <20> | <19> | <18> | <17> | <16> |
|---|---|---|---|---|---|---|---|---|
| Name | reserved | | | | | | OR_CYCLE[1:0] | |
| Default | 0 | | | | | | 1 | 1 |
| **Bit** | **<15>** | **<14>** | **<13>** | **<12>** | **<11>** | **<10>** | **<9>** | **<8>** |
| Name | OT_LOW_VALUE[7:0] | | | | | | | |
| Default | 0xE6 | | | | | | | |
| **Bit** | **<7>** | **<6>** | **<5>** | **<4>** | **<3>** | **<2>** | **<1>** | **<0>** |
| Name | OT_HIGH_VALUE[7:0] | | | | | | | |
| Default | 0xF0 | | | | | | | |

where

Table 21: OTTH register description

| Status | Description |
|---|---|
| OR_CYCLE[1:0] | Overtemperature detection low pass filter cycle length:<br>0b00 - 1 cycle<br>0b01 - 2 cycles<br>0b10 - 3 cycles<br>0b11 - 4 cycles |
| OT_LOW_VALUE[7:0] | Overtemperature fault release threshold in ADC counts. The OT_FLAG can be cleared only when the temperature is below this value. |
| OT_HIGH_VALUE[7:0] | Overtemperature fault threshold in ADC counts. The OT_FLAG is raised automatically once the temperature increases above this value. |

### SETOTTH

Writes the OTTH register. See READOTTH for the payload format.

**Note:** It takes at least one update cycle of the temperature sensor (TEMPCK) for the changes to take effect (see READSETUP).

### READSETUP

Returns the SETUP register as shown in Table 22.

Table 22: SETUP register

| Bit | <7> | <6> | <5> | <4> | <3> | <2> | <1> | <0> |
|---|---|---|---|---|---|---|---|---|
| Name | PWM_F | CLK_INV | CRC_EN | TEMPCK | CE_FSAVE | LOS_FSAVE | OT_SAVE | UV_FSAVE |
| Default | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

where

Table 23: SETUP register description

| Status | Description |
|---|---|
| PWM_F | PWM frequency and dynamic range<br>0b0 - 586 Hz / 15 bit<br>0b1 - 1172 Hz / 14 bit |
| CLK_INV | CLK polarity for MCU mode |
| CRC_EN | CRC check<br>0b0 - disabled<br>0b1 - enabled |
| TEMPCK | Update rate of the temperature sensor<br>0b0 - 19.2 kHz<br>0b1 - 2.4 kHz |
| CE_FSAVE | Device behavior when a communication error is detected<br>0b0 - Error flag is raised<br>0b1 - Error flag is raised, and the device is put into SLEEP mode |

Table 23: SETUP register description

| Status | Description |
|---|---|
| LOS_FSAVE | Device behavior when an open/short error is detected<br>0b0 - Error flag is raised<br>0b1 - Error flag is raised, and the device is put into SLEEP mode |
| OT_SAVE | Device behavior when an overtemperature error is detected<br>0b0 - Error flag is raised<br>0b1 - Error flag is raised, and the device is put into SLEEP mode |
| UV_FSAVE | Device behavior when an undervoltage error is detected<br>0b0 - Error flag is raised<br>0b1 - Error flag is raised, and the device is put into SLEEP mode |

### SETSETUP

Writes the SETUP register. See READSETUP for the payload format.

**Note:** It is recommend to enable crc check e.g. via broadcast command before init.

### READPWM

Returns the PWM register as shown in Table 24.

Table 24: PWM register

| **Bit** | **<47>** | **<46>** | **<45>** | **<44>** | **<43>** | **<42>** | **<41>** | **<40>** |
|---|---|---|---|---|---|---|---|---|
| Name | IRED | RED_PWM[14:8] | | | | | | |
| Default | 0 | 0 | | | | | | |
| **Bit** | **<39>** | **<38>** | **<37>** | **<36>** | **<35>** | **<34>** | **<33>** | **<32>** |
| Name | RED_PWM[7:0] | | | | | | | |
| Default | 0 | | | | | | | |
| **Bit** | **<31>** | **<30>** | **<29>** | **<28>** | **<27>** | **<26>** | **<25>** | **<24>** |
| Name | IGREEN | GREEN_PWM[14:8] | | | | | | |
| Default | 0 | 0 | | | | | | |
| **Bit** | **<23>** | **<22>** | **<21>** | **<20>** | **<19>** | **<18>** | **<17>** | **<16>** |
| Name | GREEN_PWM[7:0] | | | | | | | |
| Default | 0 | | | | | | | |
| **Bit** | **<15>** | **<14>** | **<13>** | **<12>** | **<11>** | **<10>** | **<9>** | **<8>** |
| Name | IBLUE | BLUE_PWM[14:8] | | | | | | |
| Default | 0 | 0 | | | | | | |
| **Bit** | **<7>** | **<6>** | **<5>** | **<4>** | **<3>** | **<2>** | **<1>** | **<0>** |
| Name | BLUE PWM[7:0] | | | | | | | |
| Default | 0 | | | | | | | |

where

Table 25: PWM register description

| Status | Description |
|---|---|
| IRED | Red peak current<br>0b0 - "nighttime" mode<br>0b1 - "daytime" mode |
| RED_PWM[14:0] | Red PWM value. If PWM_F = 0b1, only bits [13:0] are interpreted. |
| IGREEN | Green peak current<br>0b0 - "nighttime" mode<br>0b1 - "daytime" mode |
| GREEN_PWM[14:0] | Green PWM value. If PWM_F = 0b1, only bits [13:0] are interpreted. |
| IBLUE | Blue peak current<br>0b0 - "nighttime" mode<br>0b1 - "daytime" mode |
| BLUE_PWM[14:0] | Blue PWM value. If PWM_F = 0b1, only bits [13:0] are interpreted. |

### SETPWM

Writes the PWM register. See READPWM for the payload format.

**Note:** The changes take effect immediately after the command has been received.

### READOTP

Reads OTP values from address OTPADDR as given in the payload are shown in Table 26

Table 26: OTP register

| Bit | <7> | <6> | <5> | <4> | <3> | <2> | <1> | <0> |
|---|---|---|---|---|---|---|---|---|
| Name | reserved | | OTPADDR[5:0] | | | | | |

The response contains a payload of 8 bytes consisting of the 8 following bytes starting from OTPADDR ordered from highest to lowest byte.

For example, OTPADDR = 0x0A returns bytes 0x11 to 0x0A as shown in Table 27

Table 27: Example READOTP

| Header | Payload | | | | | | | | CRC |
|---|---|---|---|---|---|---|---|---|---|
| | 0x11 | 0x10 | 0x0F | 0x0E | 0x0D | 0x0C | 0x0B | 0x0A | |

If any of the 8 bytes exceeds the range of valid addresses, these bytes contain random data and should be ignored.

The response is sent out after a delay of approximately 55 µs.

**Note:** The data readout of all devices could take a while depending on number of devices. Our recommendation is a readout one time and storing data within MCU.

The memory map of the OSIRE® E3731i is shown in Figure 16.

**Note:** u and v refer to u' and v', respectively. See chapter 4.1 "Calibration data" for more details.

Figure 16: Memory map

| OTP Address | Bits | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 00 | Chip_ID[7:0] (position) | | | | | | | |
| 01 | Chip_ID[15:8] (position) | | | | | | | |
| 02 | Reserved | | | Chip_ID[20:16] (wafer number) | | | | |
| 03 | Reserved | | | | | | | |
| 04 | | | | | | | | |
| 05 | | | | | | | | |
| 06 | | | | | | | | |
| 07 | | | | | | | | |
| 08 | | | | | | | | |
| 09 | | | | | | | | |
| 0A | R_u_night[7:0] | | | | | | | |
| 0B | R_v_night[7:0] | | | | | | | |
| 0C | R_lv_night[7:0] | | | | | | | |
| 0D | R_lv_day[11:8] | | | | R_lv_night[11:8] | | | |
| 0E | R_u_day[7:0] | | | | | | | |
| 0F | R_v_day[7:0] | | | | | | | |
| 10 | R_lv_day[7:0] | | | | | | | |
| 11 | B_u_night[7:0] | | | | | | | |
| 12 | B_v_night[7:0] | | | | | | | |
| 13 | B_lv_night[7:0] | | | | | | | |
| 14 | B_lv_day[11:8] | | | | B_lv_night[11:8] | | | |
| 15 | B_u_day[7:0] | | | | | | | |
| 16 | B_v_day[7:0] | | | | | | | |
| 17 | B_lv_day[7:0] | | | | | | | |
| 18 | G_u_night[7:0] | | | | | | | |
| 19 | G_v_night[7:0] | | | | | | | |
| 1A | G_lv_night[7:0] | | | | | | | |
| 1B | G_lv_day[11:8] | | | | G_lv_night[11:8] | | | |
| 1C | G_u_day[7:0] | | | | | | | |
| 1D | G_v_day[7:0] | | | | | | | |
| 1E | G_lv_day[7:0] | | | | | | | |
| 1F | CRC2<7:0> (could also be used for other data) | | | | | | | |

## 5.3 OTP to PWM example

This section presents an example for decoding calibration data from OTP values and for the calculation of PWM values necessary to reach the D65 white point in nighttime mode as given in chapter 4.1 "Calibration data" and chapter 4.2 "PWM calculation".

The table in Figure 17 illustrates an exemplary OTP loading with the byte values given in the right-most column.

Figure 17: Example OTP Map

| OTP Address | Bits | | | | | | | | Example Data: |
|---|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0A | R_u_night[7:0] | | | | | | | | 201 |
| 0B | R_v_night[7:0] | | | | | | | | 210 |
| 0C | R_lv_night[7:0] | | | | | | | | 154 |
| 0D | R_lv_day[11:8] | | | | R_lv_night[11:8] | | | | 166 |
| 0E | R_u_day[7:0] | | | | | | | | 203 |
| 0F | R_v_day[7:0] | | | | | | | | 210 |
| 10 | R_lv_day[7:0] | | | | | | | | 145 |
| 11 | B_u_night[7:0] | | | | | | | | 51 |
| 12 | B_v_night[7:0] | | | | | | | | 84 |
| 13 | B_lv_night[7:0] | | | | | | | | 198 |
| 14 | B_lv_day[11:8] | | | | B_lv_night[11:8] | | | | 50 |
| 15 | B_u_day[7:0] | | | | | | | | 57 |
| 16 | B_v_day[7:0] | | | | | | | | 73 |
| 17 | B_lv_day[7:0] | | | | | | | | 40 |
| 18 | G_u_night[7:0] | | | | | | | | 28 |
| 19 | G_v_night[7:0] | | | | | | | | 232 |
| 1A | G_lv_night[7:0] | | | | | | | | 160 |
| 1B | G_lv_day[11:8] | | | | G_lv_night[11:8] | | | | 153 |
| 1C | G_u_day[7:0] | | | | | | | | 23 |
| 1D | G_v_day[7:0] | | | | | | | | 229 |
| 1E | G_lv_day[7:0] | | | | | | | | 155 |
| 1F | CRC2<7:0> (could also be used for other data) | | | | | | | | 149 |

Specific OTP bytes are referenced in the following by their address, e.g., the value at address 0x0A is referred to by <0x0A>. In the concrete example, <0x0A> = 201.

In order to verify data integrity, the CRC check word given in <0x1F> can be used. An example implementation for the CRC algorithm (crc_otp) is given in chapter 5.4.3 "OTP cyclic redundancy check".

In this concrete example:

```
uint8_t data[] = { 201, 210, 154, 166, 203, 210, 145, 51, 84, 198, 50, 57, 73,
                   40, 28, 232, 160, 153, 23, 229, 155, 149};  // include all
                   OTP bytes from 0x0A to 0x1F (=CRC2 checkword)

uint8_t checkword = crc_otp(data, 22);
```

As the check word in the example computes to 0x00, the data is verified.

The necessary decoding steps from OTP bytes to calibration parameters are given in Figure 18 along with the decoded parameter values and units.

Figure 18: Decoding of OTP values to calibration parameters.

| Variable Name | Decoding | Value | Unit |
|---|---|---|---|
| Red u' @ night | <0x0A> * LSB_u | 0,5025 | |
| Red v' @ night | <0x0B> * LSB_v | 0,5250 | |
| Red Iv @ night | ((<0x0D> & 0x0f) << 8 \| <0x0C>) * LSB_Iv_10 | 0,7436 | cd |
| Red u' @ day | <0x0E> * LSB_u | 0,5075 | |
| Red v' @ day | <0x0F> * LSB_v | 0,5250 | |
| Red Iv @ day | ((<0x0D> & 0xf0) << 4 \| <0x10>) * LSB_Iv_50 | 3,3813 | cd |
| Green u' @ night | <0x11> * LSB_u | 0,1275 | |
| Green v' @ night | <0x12> * LSB_v | 0,2100 | |
| Green Iv @ night | ((<0x14> & 0x0f) << 8 \| <0x13>) * LSB_Iv_10 | 0,3124 | cd |
| Green u' @ day | <0x15> * LSB_u | 0,1425 | |
| Green v' @ day | <0x16> * LSB_v | 0,1825 | |
| Green Iv @ day | ((<0x14> & 0xf0) << 4 \| <0x17>) * LSB_Iv_50 | 1,0100 | cd |
| Blue u' @ night | <0x18> * LSB_u | 0,0700 | |
| Blue v' @ night | <0x19> * LSB_v | 0,5800 | |
| Blue Iv @ night | ((<0x1B> & 0x0f) << 8 \| <0x1A>) * LSB_Iv_10 | 1,0842 | cd |
| Blue u' @ day | <0x1C> * LSB_u | 0,0575 | |
| Blue v' @ day | <0x1D> * LSB_v | 0,5725 | |
| Blue Iv @ day | ((<0x1B> & 0xf0) << 4 \| <0x1E>) * LSB_Iv_50 | 3,0738 | cd |
| CRC2 | | 149 | |

The necessary coefficients for decoding are labelled LSB_Iv_xx, LSB_u, and LSB_v. The respective values are given in chapter 4.1 "Calibration data".

Following the equations given in chapter 4.2 "PWM calculation", the tristimulus matrix is given as

$$A = \begin{pmatrix} 1.6014 & 0.2944 & 0.4268 \\ 0.7436 & 1.0842 & 0.3124 \\ -0.0027 & 0.0888 & 2.7586 \end{pmatrix}$$

with a determinant of

det(A) = 4.1702

Assuming a target brightness of 1 cd at D65 (Cx = 0.3128, Cy = 0.3290 or u' = 0.1979, v' = 0.4683), the target vector is found to be

$$T = \begin{pmatrix} 0.9508 \\ 1.0000 \\ 1.0888 \end{pmatrix}$$

The three matrices A1, A2, and A3 are obtained by replacing the respective column of the tristimulus matrix by the target vector:

$$A_1 = \begin{pmatrix} 0.9508 & 0.2944 & 0.4268 \\ 1.0000 & 1.0842 & 0.3124 \\ 1.0888 & 0.0888 & 2.7586 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 1.6014 & 0.9508 & 0.4268 \\ 0.7436 & 1.0000 & 0.3124 \\ -0.0027 & 1.0888 & 2.7586 \end{pmatrix}$$

and

$$A_3 = \begin{pmatrix} 1.6014 & 0.2944 & 0.9508 \\ 0.7436 & 1.0842 & 1.0000 \\ -0.0027 & 0.0888 & 1.0888 \end{pmatrix}$$

with determinants

$\det(A_1) = 1.6393$,

$\det(A_2) = 2.2685$,

and

$\det(A_3) = 1.5745$

respectively.

The three PWM duty cycles are finally obtained as

$$D_r = \frac{det(A_1)}{det(A)} = \frac{1.6393}{4.1702} = 0.3931$$

$$D_g = \frac{det(A_2)}{det(A)} = 0.5440$$

and

$$D_b = \frac{det(A_3)}{det(A)} = 0.3775$$

Assuming the lower PWM frequency setting is to be used (PWM_F = 0b0, see SETSETUP in Section 7.5), the PWM values are encoded using 15 bits. The respective values for the SETPWM command are thus

$PWM_r = D_r \times (2^{15} - 1) = 12881$,

$PWM_g = D_g \times (2^{15} - 1) = 17825$,

and

$PWM_b = D_b \times (2^{15} - 1) = 12370$.

## 5.4 Code examples



### 5.4.1 Manchester coding

```
//--------------- Lookup table for manchester encoding -------------------- //
uint16_t manLookUpTable[256] =
  { 0xaaaa, 0xaaa9, 0xaaa6, 0xaaa5, 0xaa9a, 0xaa99, 0xaa96, 0xaa95, 0xaa6a,
      0xaa69, 0xaa66, 0xaa65, 0xaa5a, 0xaa59, 0xaa56, 0xaa55, 0xa9aa, 0xa9a9,
```

```
        0xa9a6, 0xa9a5, 0xa99a, 0xa999, 0xa996, 0xa995, 0xa96a, 0xa969, 0xa966,
        0xa965, 0xa95a, 0xa959, 0xa956, 0xa955, 0xa6aa, 0xa6a9, 0xa6a6, 0xa6a5,
        0xa69a, 0xa699, 0xa696, 0xa695, 0xa66a, 0xa669, 0xa666, 0xa665, 0xa65a,
        0xa659, 0xa656, 0xa655, 0xa5aa, 0xa5a9, 0xa5a6, 0xa5a5, 0xa59a, 0xa599,
        0xa596, 0xa595, 0xa56a, 0xa569, 0xa566, 0xa565, 0xa55a, 0xa559, 0xa556,
        0xa555, 0x9aaa, 0x9aa9, 0x9aa6, 0x9aa5, 0x9a9a, 0x9a99, 0x9a96, 0x9a95,
        0x9a6a, 0x9a69, 0x9a66, 0x9a65, 0x9a5a, 0x9a59, 0x9a56, 0x9a55, 0x99aa,
        0x99a9, 0x99a6, 0x99a5, 0x999a, 0x9999, 0x9996, 0x9995, 0x996a, 0x9969,
        0x9966, 0x9965, 0x995a, 0x9959, 0x9956, 0x9955, 0x96aa, 0x96a9, 0x96a6,
        0x96a5, 0x969a, 0x9699, 0x9696, 0x9695, 0x966a, 0x9669, 0x9666, 0x9665,
        0x965a, 0x9659, 0x9656, 0x9655, 0x95aa, 0x95a9, 0x95a6, 0x95a5, 0x959a,
        0x9599, 0x9596, 0x9595, 0x956a, 0x9569, 0x9566, 0x9565, 0x955a, 0x9559,
        0x9556, 0x9555, 0x6aaa, 0x6aa9, 0x6aa6, 0x6aa5, 0x6a9a, 0x6a99, 0x6a96,
        0x6a95, 0x6a6a, 0x6a69, 0x6a66, 0x6a65, 0x6a5a, 0x6a59, 0x6a56, 0x6a55,
        0x69aa, 0x69a9, 0x69a6, 0x69a5, 0x699a, 0x6999, 0x6996, 0x6995, 0x696a,
        0x6969, 0x6966, 0x6965, 0x695a, 0x6959, 0x6956, 0x6955, 0x66aa, 0x66a9,
        0x66a6, 0x66a5, 0x669a, 0x6699, 0x6696, 0x6695, 0x666a, 0x6669, 0x6666,
        0x6665, 0x665a, 0x6659, 0x6656, 0x6655, 0x65aa, 0x65a9, 0x65a6, 0x65a5,
        0x659a, 0x6599, 0x6596, 0x6595, 0x656a, 0x6569, 0x6566, 0x6565, 0x655a,
        0x6559, 0x6556, 0x6555, 0x5aaa, 0x5aa9, 0x5aa6, 0x5aa5, 0x5a9a, 0x5a99,
        0x5a96, 0x5a95, 0x5a6a, 0x5a69, 0x5a66, 0x5a65, 0x5a5a, 0x5a59, 0x5a56,
        0x5a55, 0x59aa, 0x59a9, 0x59a6, 0x59a5, 0x599a, 0x5999, 0x5996, 0x5995,
        0x596a, 0x5969, 0x5966, 0x5965, 0x595a, 0x5959, 0x5956, 0x5955, 0x56aa,
        0x56a9, 0x56a6, 0x56a5, 0x569a, 0x5699, 0x5696, 0x5695, 0x566a, 0x5669,
        0x5666, 0x5665, 0x565a, 0x5659, 0x5656, 0x5655, 0x55aa, 0x55a9, 0x55a6,
        0x55a5, 0x559a, 0x5599, 0x5596, 0x5595, 0x556a, 0x5569, 0x5566, 0x5565,
        0x555a, 0x5559, 0x5556, 0x5555};

/**
 * @brief Manchester encoder
```

```
 * This function codes the bytes given by p_buffer to manchester code and
 * stores the new bytes in the same buffer. The buffer must be twice as
 * large as the number of bytes given by byteCount.
 *
 * @param p_buffer Pointer to an array where the bytes of the message are
 *                 stored and the new bytes will be saved
 * @param byteCount Specifies how many bytes form p_buffer will be encoded
 *
 */
void manchester_encoding (uint8_t* p_buffer, uint8_t byteCount)
{
  p_buffer = p_buffer + byteCount - 1;
  uint16_t buf;
  for (uint8_t counter = 0; counter < byteCount; counter++)
    {
      buf = manLookUpTable[*p_buffer];
      p_buffer = p_buffer + byteCount - counter;
      *p_buffer = buf & 0x00ff;
      p_buffer--;
      *p_buffer = buf >> 8;
      p_buffer = p_buffer - byteCount + counter;
    }
}
```

### 5.4.2 Message cyclic redundancy check

**Examples:**

Sending:

CRC Input: 0xA0_04_02' CRC Checkword: 0xA9 ' Full Message: 0xA0_04_02_A9

Receiving:

CRC Input: 0xA0_04_02_A9' CRC Checkword: 0x00

### CRC without Look-Up Table

A possible implementation of the CRC algorithm is the following:

```
uint8_t crc(uint8_t *InBytes, uint8_t InSize)
{
	uint8_t crc = 0x00;
	uint8_t extract;
	for(uint8_t i = 0;i < InSize; i++)
	{
	  extract = *InBytes;
	  for (uint8_t mask = 8; mask > 0; mask--)
	  {
		if (((extract >> (mask - 1)) & 0x01) == ((crc >> 7) & 0x01))
			crc = crc << 1;
		else
			crc = (crc << 1) ^ 0x2F;
		crc &= 0xFF;
	  }
	  InBytes++;
	}
	return crc;
}
```

### CRC with Look-Up Table

The same CRC calculation with look-up table:

```
//------------------ crc lookup table for polynom 0x2F  --------------------//
uint8_t crcLookUpTable[256] = { 0x00, 0x2F, 0x5E, 0x71, 0xBC, 0x93, 0xE2, 0xCD,
0x57, 0x78, 0x09, 0x26,
      0xEB, 0xC4, 0xB5, 0x9A, 0xAE, 0x81, 0xF0, 0xDF, 0x12, 0x3D, 0x4C, 0x63,
      0xF9, 0xD6, 0xA7, 0x88, 0x45, 0x6A, 0x1B, 0x34, 0x73, 0x5C, 0x2D, 0x02,
      0xCF, 0xE0, 0x91, 0xBE, 0x24, 0x0B, 0x7A, 0x55, 0x98, 0xB7, 0xC6, 0xE9,
```

```
        0xDD, 0xF2, 0x83, 0xAC, 0x61, 0x4E, 0x3F, 0x10, 0x8A, 0xA5, 0xD4, 0xFB,
        0x36, 0x19, 0x68, 0x47, 0xE6, 0xC9, 0xB8, 0x97, 0x5A, 0x75, 0x04, 0x2B,
        0xB1, 0x9E, 0xEF, 0xC0, 0x0D, 0x22, 0x53, 0x7C, 0x48, 0x67, 0x16, 0x39,
        0xF4, 0xDB, 0xAA, 0x85, 0x1F, 0x30, 0x41, 0x6E, 0xA3, 0x8C, 0xFD, 0xD2,
        0x95, 0xBA, 0xCB, 0xE4, 0x29, 0x06, 0x77, 0x58, 0xC2, 0xED, 0x9C, 0xB3,
        0x7E, 0x51, 0x20, 0x0F, 0x3B, 0x14, 0x65, 0x4A, 0x87, 0xA8, 0xD9, 0xF6,
        0x6C, 0x43, 0x32, 0x1D, 0xD0, 0xFF, 0x8E, 0xA1, 0xE3, 0xCC, 0xBD, 0x92,
        0x5F, 0x70, 0x01, 0x2E, 0xB4, 0x9B, 0xEA, 0xC5, 0x08, 0x27, 0x56, 0x79,
        0x4D, 0x62, 0x13, 0x3C, 0xF1, 0xDE, 0xAF, 0x80, 0x1A, 0x35, 0x44, 0x6B,
        0xA6, 0x89, 0xF8, 0xD7, 0x90, 0xBF, 0xCE, 0xE1, 0x2C, 0x03, 0x72, 0x5D,
        0xC7, 0xE8, 0x99, 0xB6, 0x7B, 0x54, 0x25, 0x0A, 0x3E, 0x11, 0x60, 0x4F,
        0x82, 0xAD, 0xDC, 0xF3, 0x69, 0x46, 0x37, 0x18, 0xD5, 0xFA, 0x8B, 0xA4,
        0x05, 0x2A, 0x5B, 0x74, 0xB9, 0x96, 0xE7, 0xC8, 0x52, 0x7D, 0x0C, 0x23,
        0xEE, 0xC1, 0xB0, 0x9F, 0xAB, 0x84, 0xF5, 0xDA, 0x17, 0x38, 0x49, 0x66,
        0xFC, 0xD3, 0xA2, 0x8D, 0x40, 0x6F, 0x1E, 0x31, 0x76, 0x59, 0x28, 0x07,
        0xCA, 0xE5, 0x94, 0xBB, 0x21, 0x0E, 0x7F, 0x50, 0x9D, 0xB2, 0xC3, 0xEC,
        0xD8, 0xF7, 0x86, 0xA9, 0x64, 0x4B, 0x3A, 0x15, 0x8F, 0xA0, 0xD1, 0xFE,
        0x33, 0x1C, 0x6D, 0x42};

//------------------------------ crc  -----------------------------------//
uint8_t crc (uint8_t* p_buffer, uint8_t byteCount)
{
  uint8_t counter, crc = 0;

  //for loop for every byte which is given by parameter byteCount
  for (counter = 0; counter < byteCount; counter++)
    {
      crc ^= *p_buffer;           //xor current crc with value of pointer
      crc = crcLookUpTable[crc];  //take new crc from lookup table
      p_buffer++;                 //increase address of pointer
```

```
    }
  return crc;                    //return crc
}
```

### 5.4.3 OTP cyclic redundancy check

The OTP CRC uses the same polynomial as the message-CRC with the difference that the bit order within an input byte and the CRC output byte is reversed.

**Examples**:

Sending:

CRC Input: 0xA0_04_02 --> CRC Checkword: 0x59 --> Full Message: 0xA0_04_02_59

Receiving:

CRC Input: 0xA0_04_02_59 --> CRC Checkword: 0x00

A possible implementation of the CRC algorithm is the following (differences to chapter 5.4.2 "Message cyclic redundancy check" are highlighted by bold font):

```
uint8_t reverse_bits(uint8_t b)
{
   b = (b & 0xf0) >> 4 | (b & 0x0f) << 4;
   b = (b & 0xcc) >> 2 | (b & 0x33) << 2;
   b = (b & 0xaa) >> 1 | (b & 0x55) << 1;
   return b;
}


uint8_t crc_otp(uint8_t *InBytes, uint8_t InSize)
{
       uint8_t crc = 0x00;
       uint8_t extract;
       for(uint8_t i = 0;i < InSize; i++)
       {
         extract = *InBytes;
```

```
        for (uint8_t mask = 0; mask < 8; mask++)
        {
            if (((extract >> (mask - 1)) & 0x01) == ((crc >> 7) & 0x01))
                crc = crc << 1;
            else
                crc = (crc << 1) ^ 0x2F;
            crc &= 0xFF;
        }
        InBytes++;
    }
    return reverse_bits(crc);
}
```

## 5.5 Example chain configurations

Figure 19 shows a bidirectional chain example with CAN-FD section. The 1st OSIRE® E3731i serves as a Manchester decoder between the chain and the MCU (upstream direction).

Figure 19: Bidirectional chain example with CAN-FD section

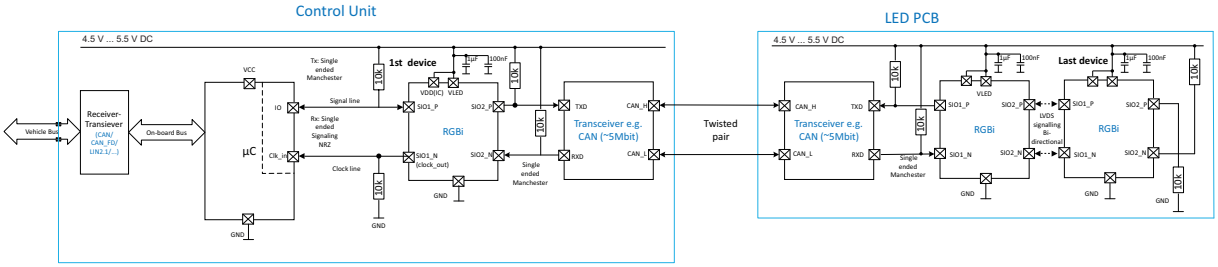



Figure 20 shows a bidirectional chain example with direct MCU - CAN-FD connection. The MCU needs to implement the Manchester decoder.

Figure 20: Bidirectional chain example with direct MCU-CAN-FD connection

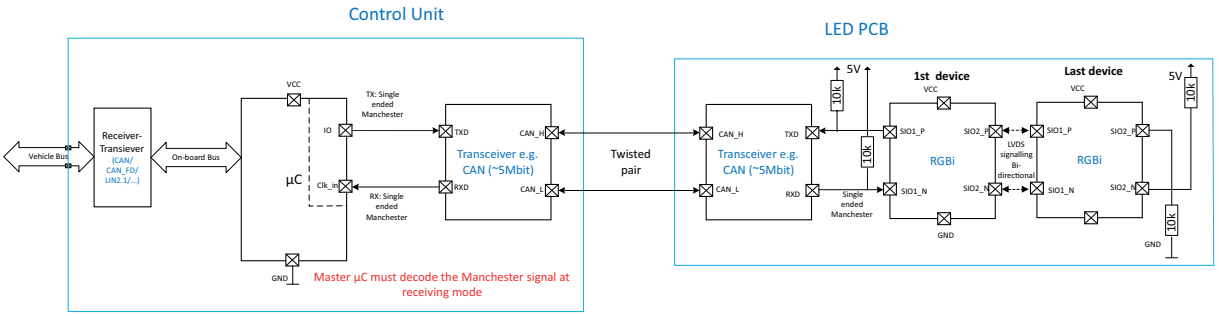
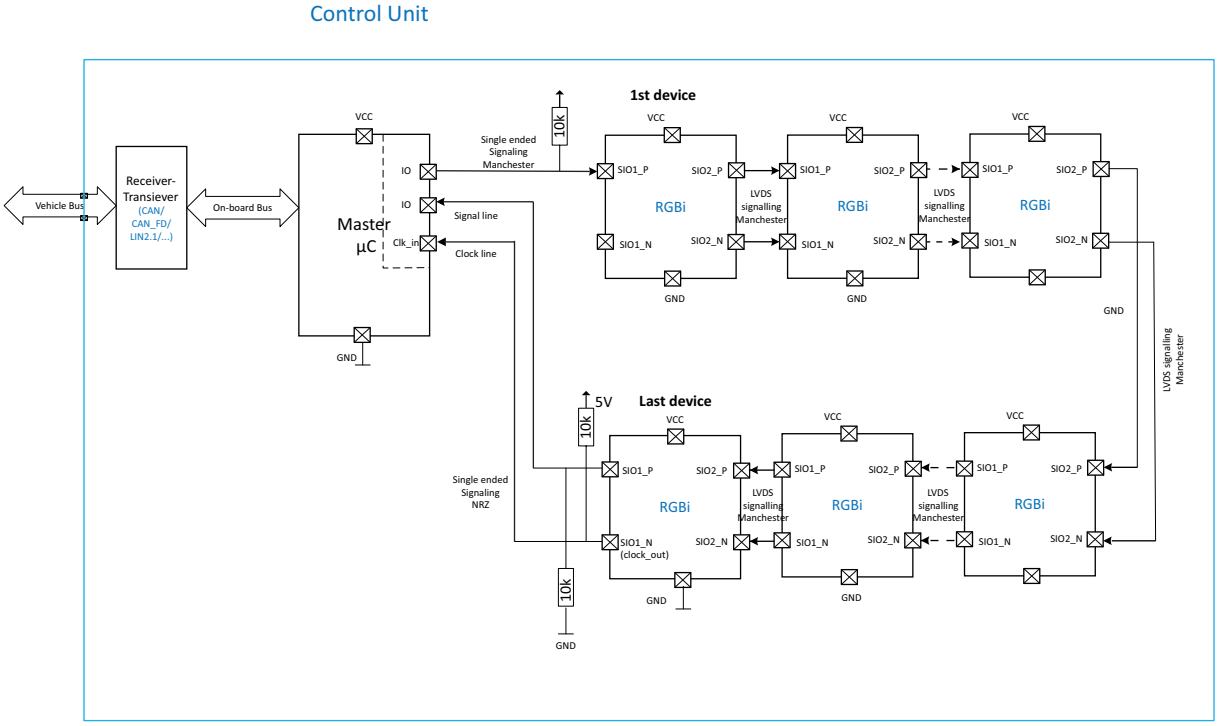


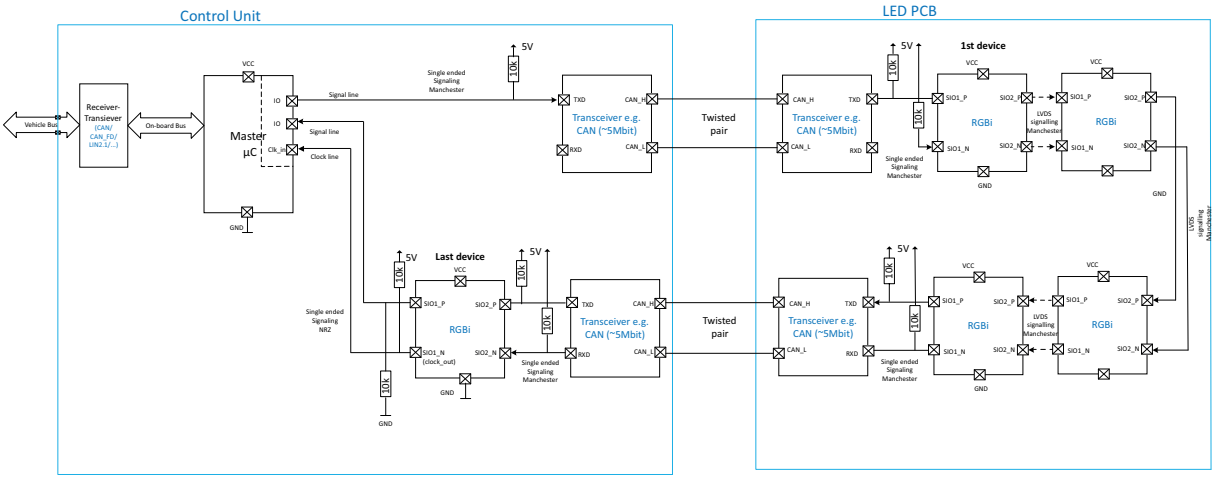

Figure 21 shows a loop-back chain example on a single PCB.

Figure 21: Loop-Back chain on a single PCB




An example of a loop-back chain on two PCBs is shown in Figure 22.

Figure 22: Loop-back chain on two PCBs




## 5.6 Message timing estimations

Table 28 and Table 29 show the message timing calculations.

Table 28: Message timing calculations

| COM speed | Bits | Min Mbps | Typ Mbps | Max Mbps |
|---|---|---|---|---|
| COM Data Rate MCU (+/- 8%) | | 2.21 | 2.4 | 2.59 |
| COM Manchester Coded | | 4.42 | 4.8 | 5.18 |
| COM Data Rate RGBi (+/- 5%) | | 2.28 | 2.4 | 2.52 |
| Duration 1 bit (µs) | 1 | 0.44 | 0.42 | 0.40 |
| Duration 16 bit (µs) | 16 | 7.0 | 6.7 | 6.3 |
| Duration 48 bit (µs) | 48 | 21 | 20 | 19 |
| Duration 80 bit (µs) | 80 | 35 | 33 | 32 |
| Duration 96 bit (µs) | 96 | 42 | 40 | 38 |

| Propagation delay (15 bit) fast-forwarding (broadcast & unmatched address) | devices | Min Mbps | Typ Mbps | Max Mbps |
|---|---|---|---|---|
| Duration (µs) | 1 | 8,7 | 8,1 | 7,5 |
| Duration (µs) | 100 | 871,7 | 807,3 | 749,0 |

Table 28: Message timing calculations

| Propagation delay (32 bit) init address or ping for error | devices | Min Mbps | Typ Mbps | Max Mbps |
|---|---|---|---|---|
| Duration (µs) | 1 | 14,9 | 14,1 | 12,1 |
| Duration (µs) | 100 | 1493,4 | 1408,3 | 1208,3 |
| Duration (µs) incl. answer from last device - bi-directional | 100 | 2365,1 | 2215,6 | 1957,3 |
| Duration (µs) incl. answer from last device - bi-directional | 300 | 7095,4 | 6646,9 | 5872,0 |
| | | | | |
| **Minimum MCU wait time between packets (µs)** | 8.3 | | | |

Table 29:

| Daisy chain total loading (ms) | 50 devices | 100 devices | 300 devices |
|---|---|---|---|
| init address | 0.7 | 1.4 | 4.2 |
| init address incl. answer from last device at bi-directional mode | 1.1 | 2.2 | 6.6 |
| init address incl. answer from last device at loop back mode | 0.7 | 1.4 | 4.2 |
| Set setup register via broadcast | 0.4 | 0.8 | 2.4 |
| Set active or sleep mode via broadcast | 0.4 | 0.8 | 2.4 |
| Set PWM & current individually for all devices | 1.92 | 4.2 | 12.5 |
| Receive status & temperature (set SR-bit) from last device at bi-directional mode | 0.4 | 0.8 | 2.4 |
| Receive status & temperature (set SR-bit) from last device at loop back mode | 0.02 | 0.02 | 0.02 |
| Receive all optical data from OTP (typ. onetime readout) | 77 | 276 | 2281 |

ABOUT ams OSRAM Group (SIX: AMS)

The ams OSRAM Group (SIX: AMS) is a global leader in optical solutions. By adding intelligence to light and passion to innovation, we enrich people's lives. This is what we mean by Sensing is Life. With over 110 years of combined history, our core is defined by imagination, deep engineering expertise and the ability to provide global industrial capacity in sensor and light technologies. Our around 24,000 employees worldwide focus on innovation across sensing, illumination and visualization to make journeys safer, medical diagnosis more accurate and daily moments in communication a richer experience. Headquartered in Premstaetten/Graz (Austria) with a co-headquarters in Munich (Germany), the group achieved over EUR 5 billion revenues in 2021. Find out more about us on https://ams-osram.com

DISCLAIMER

PLEASE CAREFULLY READ THE BELOW TERMS AND CONDITIONS BEFORE USING THE INFORMATION SHOWN HEREIN. IF YOU DO NOT AGREE WITH ANY OF THESE TERMS AND CONDITIONS, DO NOT USE THE INFORMATION.

The information provided in this general information document was formulated using the utmost care; however, it is provided by ams-OSRAM AG or its Affiliates* on an "as is" basis. Thus, ams-OSRAM AG or its Affiliates* does not expressly or implicitly assume any warranty or liability whatsoever in relation to this information, including – but not limited to – warranties for correctness, completeness, marketability, fitness for any specific purpose, title, or non-infringement of rights. In no event shall ams-OSRAM AG or its Affiliates* be liable – regardless of the legal theory – for any direct, indirect, special, incidental, exemplary, consequential, or punitive damages arising from the use of this information. This limitation shall apply even if ams-OSRAM AG or its Affiliates* has been advised of possible damages. As some jurisdictions do not allow the exclusion of certain warranties or limitations of liabilities, the above limitations and exclusions might not apply. In such cases, the liability of ams-OSRAM AG or its Affiliates* is limited to the greatest extent permitted in law.

ams-OSRAM AG or its Affiliates* may change the provided information at any time without giving notice to users and is not obliged to provide any maintenance or support related to the provided information. The provided information is based on special conditions, which means that the possibility of changes cannot be precluded.

Any rights not expressly granted herein are reserved. Other than the right to use the information provided in this document, no other rights are granted nor shall any obligations requiring the granting of further rights be inferred. Any and all rights and licenses regarding patents and patent applications are expressly excluded.

It is prohibited to reproduce, transfer, distribute, or store all or part of the content of this document in any form without the prior written permission of ams-OSRAM AG or its Affiliates* unless required to do so in accordance with applicable law..

* ("Affiliate" means any existing or future entity: (i) directly or indirectly controlling a Party; (ii) under the same direct, indirect or joint ownership or control as a Party; or (iii) directly, indirectly or jointly owned or controlled by a Party. As used herein, the term "control" (including any variations thereof) means the power or authority, directly or indirectly, to direct or cause the direction of the management and policies of such Party or entity, whether through ownership of voting securities or other interests, by contract or otherwise.)

For further information on our products please see the Product Selector and scan this QR Code.

Published by ams-OSRAM AG
Tobelbader Strasse 30, 8141 Premstaetten, Austria

**Published by ams-OSRAM AG**
Tobelbader Strasse 30,
8141 Premstaetten Austria
Phone +43 3136 500-0
ams-osram.com