TLD7002-16ES OTP programming procedure

Application note

LITIX™ Pixel Rear
Multi-channel LED driver

About this document

Scope and purpose

The application note is intended to show the procedures to write and emulate the TLD7002-16ES OTP memory, to check the OTP content after the writing procedure and to perform the CRC calculations used to protect the OTP memory registers.

Intended audience

HW/SW designers, LED system architects and engineers for LED lighting applications

Table of contents

Table of contents

1 Introduction

The TLD7002-16ES implements an internal 40-byte OTP memory to store the device configurations.

The OTP memory is loaded into the device configuration at the state/mode transition IDLE to INIT and the integrity of the safety-relevant contents is checked with a 16-bit CRC protection word.

The OTP memory can be written only once but because of the emulation procedure, it is possible to test the desired configuration before permanently writing into the device. The emulated configuration can be reset via a power cycle.

The TLD7002-16 can be used in 2 different levels of emulation, complete emulation and partial emulation. In the complete OTP emulation, OTP registers has to be emulated and the CRC has to be correctly calculated. Then the device can be moved into the INIT and ACTIVE mode like during normal operation.

In the partial OTP emulation, the device has to remain in OTP mode, default register values are used as initial configuration values and the CRC is not checked. Channels can be switched ON and diagnostic is functional. Limitations for partial OTP mode are: no watchdog is applied, fault management cannot send the device into INIT mode and the safety mechanism is not available. It is suggested to not use this mode in application.

The complete OTP emulation and writing procedures are quite similar. Both need access to GPIN0 pin of the device and to the HSLI interface.

The main differences between the two procedures are related to the supply voltage (V_S) and to the protection key. The writing procedure needs a supply voltage (V_S) between 15.5 V and 20 V, while the emulation procedure only needs $V_S \geq 6$ V. The appropriate protection key is to be written in an HSLI register to enable the writing or emulation procedures.

2 OTP programming procedure

The OTP programming procedure can be divided in two parts which include the programming phase and the verification phase.

The OTP configuration array has to be prepared in advance. This can be easily achieved by clicking on **save configuration** in the OTP wizard tool.

The configuration array is located at the end of the .ocfg saved file: in the **HEX_DATA_16BIT** field. The file is a txt format, and can be open with a text editor.

Programming phase

The programming phase can be traced in step 1 to step 9 in Figure 1.

In this phase the user programs the desired OTP configuration. During this phase, the device uses the default OTP configuration with the following parameters:

- Interframe delay = 1 ms
- LP_INIT = disabled and the maximum baudrate is 2 Mbit/s

Verification phase

The verification phase is depicted in step 10 to step 16 in Figure 1.

In this phase the user checks if the device is programmed with success. During this phase the device uses the written OTP configuration if the programming procedure ended with success. The following parameters shall be taken into consideration for the proper communication with the device:

- Interframe delay
- LP_INIT: if the LP_INIT is activated, the maximum baudrate in INIT mode is 500 kbit/s

The OTP memory can be written with one of the following procedures:

- ***Register by register***: the OTP writing procedure, presented in the state diagram in Figure 1, can be repeated a number of times corresponding to the number of registers to be written
- ***One time - register by register***: The procedure, presented in the state diagram, Figure 1, can be done once but writing register by register instead of using only two WRITE_REG HSLI frames to write the entire OTP
- ***One time - two WRITE_REG frames***: With only two WRITE_REG HSLI frames, as presented in the state diagram in Figure 1, it is possible to write all 40 registers of the OTP device memory

Regardless of the adopted procedure, the following concepts shall be taken into consideration:

- A register can be written only once. If a register is written twice, the device reports a fault
- The CRC protects the OTP registers from address 0x83 to address 0xA5 and shall be calculated with the values written on these registers
- A partially written OTP device will process only broadcast commands and commands to address 0x01
- A device fully written but with corrupted CRC protection, processes only broadcast commands

2.1 OTP programming steps

The fastest procedure to write the OTP memory (*One time - two WRITE_REG frames)* uses only two WRITE_REG HSLI frames and will be illustrated in the following section.

2.1.1 Prerequisites

- GPIN0 shall be accessible and controllable with a voltage between 0 V and 5 V (see the product datasheet for the absolute maximum rating)
- HSLI bus shall be available
- VS pin shall be supplied with the OTP programming voltage (15.5 V ≤ V_S ≤ 20 V)
- A complete OTP configuration array available

- Device not emulated
- If multiple devices are connected on the same HSLI bus during the programming phase, the device with address 0x01, if present, should be the last (temporal) device programmed. This is because 0x01 is the default address for non-programmed devices, so step 11 and step 16 will cause all non-programmed devices to reply

2.1.2 OTP writing procedure

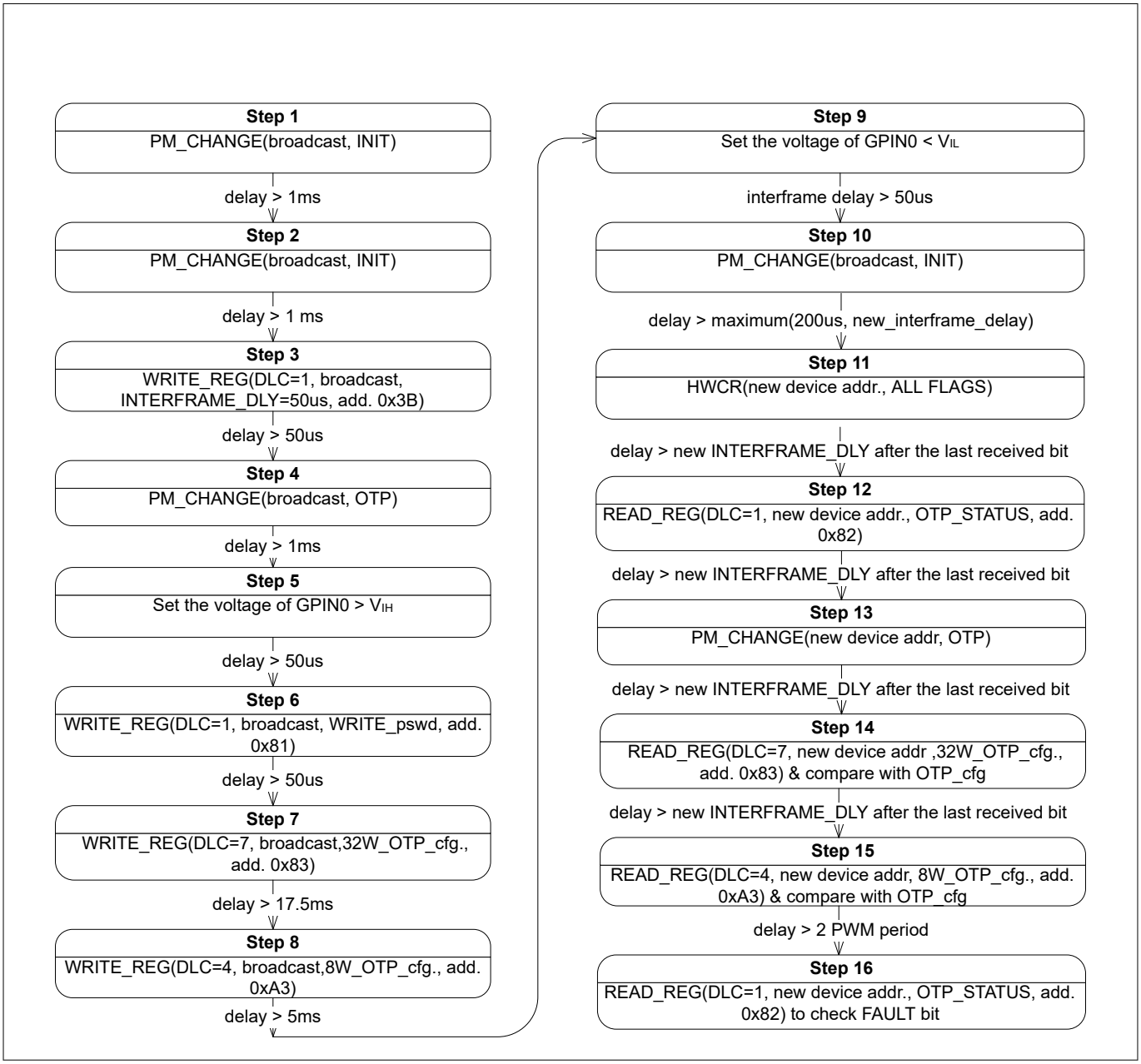

Figure 1 OTP writing procedure sequence

Delay and interframe delay: All delays and interframe delays, start from the last bit transmitted on the HSLI bus after an HSLI bus operation.

Step 1 and Step 2: The device is in INIT mode to enter into OTP mode. The PM_CHANGE frame is sent twice (Step 1 and Step 2) to ensure the synchronization of the master rolling counter. Broadcast address is used since all the not-programmed devices have HSLI address 0x01 by default.

2 OTP programming procedure

Step 3: The default interframe delay (1 ms) is changed to 50 µs writing the HSLI register 0x3B.

Step 4: The device is moved in OTP mode to write the OTP memory.

Broadcast address is used.

Step 5: The GPIN0 voltage of the device being programmed is be raised above the V_{IH} minimum voltage to enable the procedure (see the product datasheet [1] for the V_{IH} value).

Step 6: Once raised the GPIN0 voltage, the “write” password is written in the register OTP_WRITE (address 0x81).

The “write” password is 0xA47B.

Broadcast address is used.

Step 7 and Step 8: After Step 6, the device is in the programming mode and, if the VS is supplied with a voltage between 15.5 V and 20 V and T_J < 85°C, the OTP memory is written (If V_S is outside this range, this can generate a reliability issue on programmed data).

The fastest way to write the 40 registers of the OTP memory is to use a WRITE_REG(DLC=7) plus a WRITE_REG(DLC=4) (32 words plus 8 words, a word is 16 bits) corresponding to the Step 7 and Step 8.

The time required to write 32 words is 17 ms, while 5 ms to write 8 words is required.

Broadcast address is used in Step 7 and Step 8.

Step 9: The GPIN0 voltage of the device under programming can be set below the V_{IL} maximum voltage (see the product datasheet [1] for the V_{IL} value).

Step 10:

The device is moved to INIT mode by using the PM_CHANGE(broadcast, INIT) command.

If the OTP writing procedure ends successfully then the device is configured by the new OTP array.

After the PM_CHANGE HSLI frame the control unit waits a time higher than the maximum between 200 µs and the new interframe delay written in the OTP memory before proceeding to the next step.

Broadcast address is used to move the device to INIT mode.

From now on the control unit uses the new OTP parameters (e.g. interframe delay and address) for the communication with the device.

- Interframe delay written into the OTP: 50 µs
- The delay after the Step 10 shall be higher than 200 µs
- Interframe delay written in the OTP: 2.5 ms
- The delay after the Step 10 shall be higher than 2.5 ms

Step 11:

Send HWCR frame to delete the FAULT produced at startup by the unwritten device.

New device address is used for this step.

Step 12:

The register OTP_STATUS (address 0x82) is read to check if the writing operation is successful.

The register is read using the READ_REG HSLI frame with DLC set to 1 and starting address set to 0x82.

The OTP_STATUS is 0x0003.

If the OTP_STATUS register is not as expected or the device does not reply, it means that OTP programming procedure has failed, and the procedure should stop here.

Step 13:

The device shall be moved to OTP mode to read back the OTP configuration array, by using the PM_CHANGE frame .

If the device does not reply, the OTP programming procedure has failed, and the procedure should stop here.

New device address is used for this step.

Step 14 and Step 15

Read back and compare the entire OTP array.

2 OTP programming procedure

The fastest way to read back the 40 registers of the OTP fuses is to use READ_REG(DLC=7) and READ_REG(DLC=4). Data length code, DLC=7 means 32 words and DLC=4 means 8 words. Each word is 16 bits.

The read back OTP configuration array matches with the one written in steps 7 and 8, otherwise the OTP programming has not performed successfully and the device is discarded.

New device address is used for these steps.

Step 16

Read a register in order to receive the OUTPUT STATUS byte, alternatively a DC_UPDATE frame with DLC0 can be sent, this would also move the device to ACTIVE mode.

The OUTPUT STATUS byte bit 0 (FAULT), in the slave response, is 0.

If the FAULT bit is not 0 or the device does not reply, the OTP programming procedure has failed.

New device address is used for this step.

Note: In case the OTP programming procedure has failed at step 11 or 12, this could mean that the device is now corrupted or that the V_S voltage was below 15.5 V before a write batch (so the device is still unwritten or partially, but correctly written). In this last case (device unwritten or partially, but correctly written) the OTP procedure could be repeated in the attempt to save the LED driver module.

Note: The only method to understand if a device is unwritten is to have only this device on the HSLI bus (or only one with address 0x01), to apply 6 V < V_S < 13 V and try to perform an OTP write following steps 1 to 11. The read back of the OTP_STATUS register for an unwritten device at step 12 is 0x011B.

Note: When the device is VIRGIN or Partially programmed, it replies to address 0x01, if fully programmed, it answers to its own address. If fully programmed but with a wrong CRC, the device does not reply and will accept only broadcast command, therefore it can be emulated.

3 OTP emulation procedure

3.1 Prerequisites

- GPIN0 shall be accessible and controllable with a voltage between 0 V and 5 V (see the product datasheet for the absolute maximum rating)
- HSLI bus shall be available
- VS pin shall be supplied with a voltage higher than 6 V
- Device not already emulated/written with GPIN0 set as analog input (the emulation procedure needs the GPIN0 as digital input to activate the OTP emulation mode)

3.2 Complete OTP emulation procedure

The complete emulation procedure is almost the same as the programming procedure with a few minor differences. In the steps below, only the differences are explained.

3 OTP emulation procedure

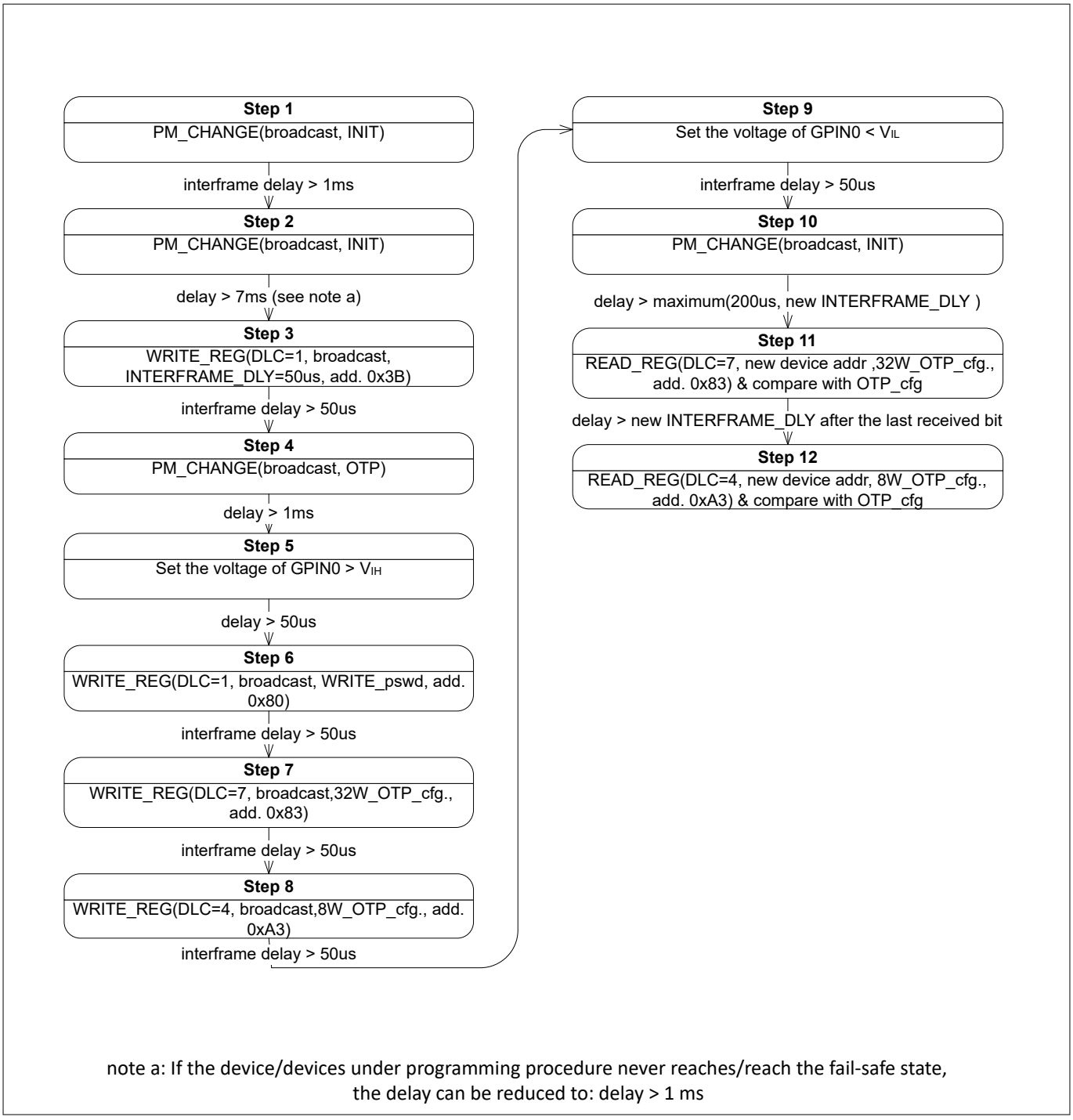

Figure 2 OTP complete emulation sequence

- **Step 6**: Once the GPIN0 voltage rises, the “emulate” password must be written in the register OTP_EMULATION (address 0x80). The “emulate” password is 0x3BD2.

  Broadcast address is used to write the register.
- The delay after Step 7 and Step 8 can be set to 50 µs.
- **Step 11:** After step 11, check if the device replies to the new address and if OTP_STATUS bits 1 and 0 are equal to 0 and 1.

  The emulation procedure is successfully performed if the read back of the 40 words OTP registers from 0x83 to 0xAA is identical to the configuration file that has just been written.

3.3 Partial OTP emulation procedure

In the partial emulation procedure it is not necessary to write all the registers nor the CRC. The CRC error will cause the OST Fault bit set to 1, but this is ignored by the device (INIT transition is not triggered).

Channels can be switched ON and diagnostic is functional. Limitations for partial OTP mode are: no watchdog is applied (only in INIT and in ACTIVE), and Diag management (load Fault will not send the device into INIT mode). All safety mechanisms are disabled.

The partial OTP procedure is shorter than the complete OTP writing procedure, and it is shown in Figure 3.

In order to remain in OTP partial emulation mode the device must not be moved to ACTIVE mode or INIT mode.

If the device is moved to INIT mode , the partial emulation will stop and the OTP CRC check will be performed.

In the partial OTP emulation, the device has to remain in OTP mode, default register values are used as initial configuration values. Partial OTP emulation can be used only for debug purposes, not in real application.

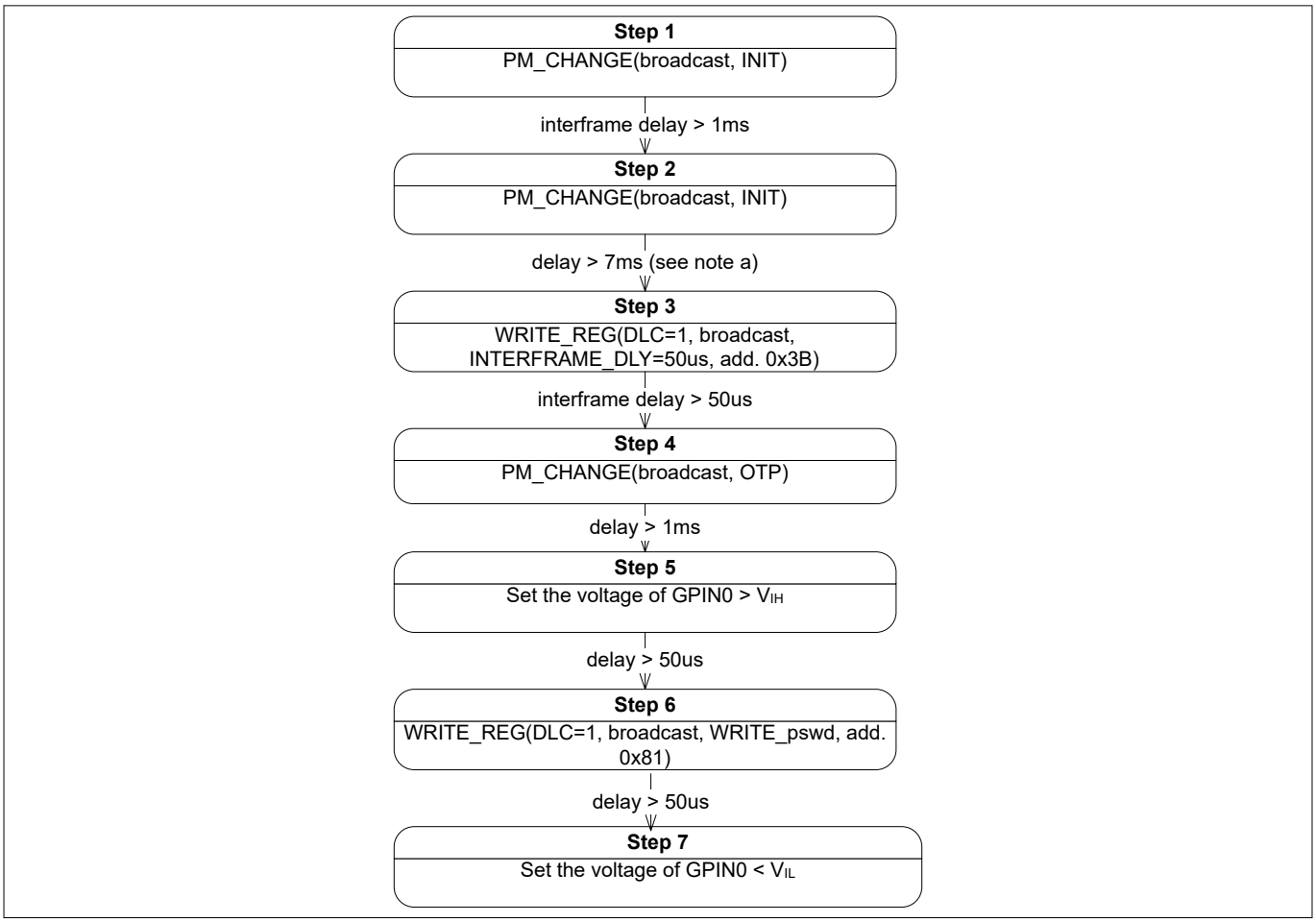

Figure 3 OTP partial emulation procedure

4 CRC16 calculation

The CRC16 protects the OTP data integrity from the OTP register address 0x83 to the address 0xA5 and it is stored in the register 0xA6.

The CRC16 has seed 0xFFFF and polynomial 0x1021.

The LUT ***crc_table_CRC_OTP_DATA*** and the function ***TLD7002_calculate_crc16_OTP*** reported below are an example of the CRC16 calculation.

CRC16 LUT and TLD7002_calculate_crc16_OTP function

```
/**
* LUT for the OTP Memory CRC16 calculation
*/
static int16_t crc_table_CRC_OTP_DATA[256] = { 0, 4129, 8258, 12387, 16516, 20645, 24774,
28903, 33032, 37161, 41290, 45419, 49548, 53677, 57806, 61935, 4657, 528, 12915, 8786, 21173,
17044, 29431, 25302, 37689, 33560, 45947, 41818, 54205, 50076, 62463, 58334, 9314, 13379,
1056, 5121, 25830, 29895, 17572, 21637, 42346, 46411, 34088, 38153, 58862, 62927, 50604, 54669,
13907, 9842, 5649, 1584, 30423, 26358, 22165, 18100, 46939, 42874, 38681, 34616, 63455, 59390,
55197, 51132, 18628, 22757, 26758, 30887, 2112, 6241, 10242, 14371, 51660, 55789, 59790, 63919,
35144, 39273, 43274, 47403, 23285, 19156, 31415, 27286, 6769, 2640, 14899, 10770, 56317, 52188,
64447, 60318, 39801, 35672, 47931, 43802, 27814, 31879, 19684, 23749, 11298, 15363, 3168, 7233,
60846, 64911, 52716, 56781, 44330, 48395, 36200, 40265, 32407, 28342, 24277, 20212, 15891,
11826, 7761, 3696, 65439, 61374, 57309, 53244, 48923, 44858, 40793, 36728, 37256, 33193, 45514,
41451, 53516, 49453, 61774, 57711, 4224, 161, 12482, 8419, 20484, 16421, 28742, 24679, 33721,
37784, 41979, 46042, 49981, 54044, 58239, 62302, 689, 4752, 8947, 13010, 16949, 21012, 25207,
29270, 46570, 42443, 38312, 34185, 62830, 58703, 54572, 50445, 13538, 9411, 5280, 1153, 29798,
25671, 21540, 17413, 42971, 47098, 34713, 38840, 59231, 63358, 50973, 55100, 9939, 14066, 1681,
5808, 26199, 30326, 17941, 22068, 55628, 51565, 63758, 59695, 39368, 35305, 47498, 43435,
22596, 18533, 30726, 26663, 6336, 2273, 14466, 10403, 52093, 56156, 60223, 64286, 35833, 39896,
43963, 48026, 19061, 23124, 27191, 31254, 2801, 6864, 10931, 14994, 64814, 60687, 56684, 52557,
48554, 44427, 40424, 36297, 31782, 27655, 23652, 19525, 15522, 11395, 7392, 3265, 61215, 65342,
53085, 57212, 44955, 49082, 36825, 40952, 28183, 32310, 20053, 24180, 11923, 16050, 3793, 7920};
/**
*
* This functions calculates the CRC16 used to protect the data integrity of the OTP memory
* data_array: uint8* pointer to data_array
* data_len: uint8 length of data array
*
* @return uint16 calculated CRC16
*
*/
uint16_t TLD7002_calculate_crc16_OTP(uint8* data_array , uint8 data_len) {
uint16_t crc = 0xFFFF; /* CRC seed = initial value */
uint8_t index = 0;
uint8_t table_index = 0;
/* for all elements of the data array */
for (index=0; index<data_len; index++) { table_index=( ((uint8_t)(crc>>8)&0xFF) ^
data_array[index]) ; crc=crc_table_CRC_OTP_DATA[table_index]^(crc<<8); } return crc; }
```

4 CRC16 calculation

The OTP content vector used in input to the function `TLD7002_calculate_crc16_OTP` shall have the following structure:

OTP content vector

```
uint8_t data_array[70];
data_array[0] = Content of OTP register address 0x83 from bit 15 to bit 8
data_array[1] = Content of OTP register address 0x83 from bit 7 to bit 0
data_array[2] = Content of OTP register address 0x84 from bit 15 to bit 0
data_array[3] = Content of OTP register address 0x84 from bit 7 to bit 0
…
data_array[68] = Content of OTP register address 0xA5 from bit 15 to bit 0
data_array[69] = Content of OTP register address 0xA5 from bit 7 to bit 0
```

The pseudocode below is an example of the function `TLD7002_calculate_crc16_OTP` usage:

Example of CRC16 calculation with pseudocode

```
/*Definition of the variable to store the calculated CRC16*/
uint16_t calculated_crc;
/*Input data vector definition, OTP register 0x83 to 0xA5*/
/*0x83_bit15-8, 0x83_bit7-0, 0x82_bit15-8, 0x82_bit7-0 , …, 0xA5_bit15-8, 0xA5bit7-0*/
uint8_t data_array[70] = {0x00, 0x00 ,…, 0x02};
/*CRC Calculation*/
calculated_crc = TLD7002_calculate_crc16_OTP(data_array , 70);
```

5 OTP Wizard Tool

The OTP Wizard, available in the Infineon toolbox, is a useful tool for the development phase of a project and it is designed to read, emulate and write the OTP memory of the TLD7002-16ES.

The tool is based on a GUI and on the OTP programmer board used to access to the HSLI bus and to control the GPIN0 pin of the TLD7002-16ES.

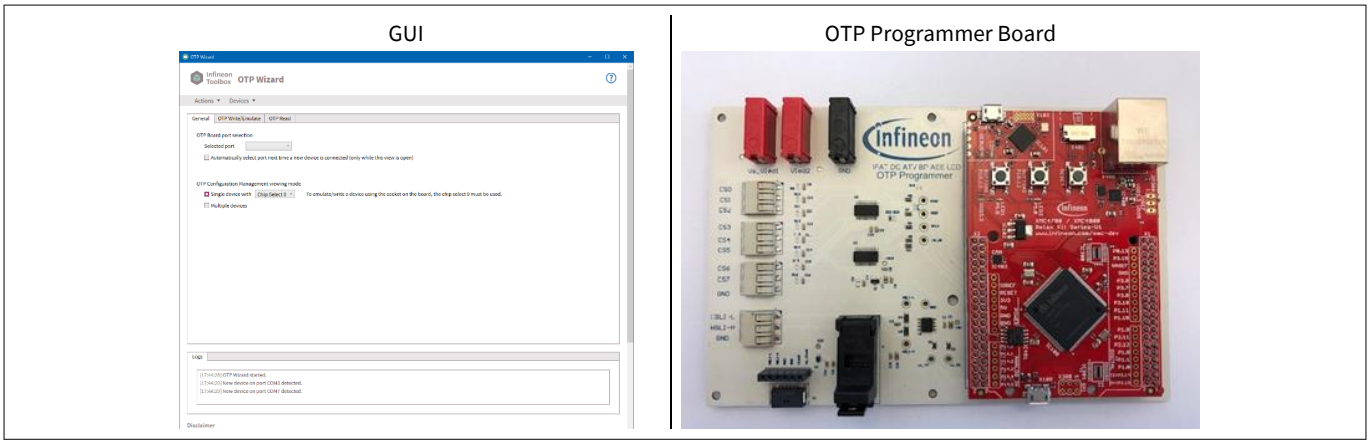

Figure 4 OTP configuration tools GUI and programmer board

5.1 General overview

The GUI is interfaced to the OTP programmer board using an USB cable. The board is detected by the operating system as a COM port that must be selected on the GUI tab, **General**, before proceeding to read, write or emulate the device.

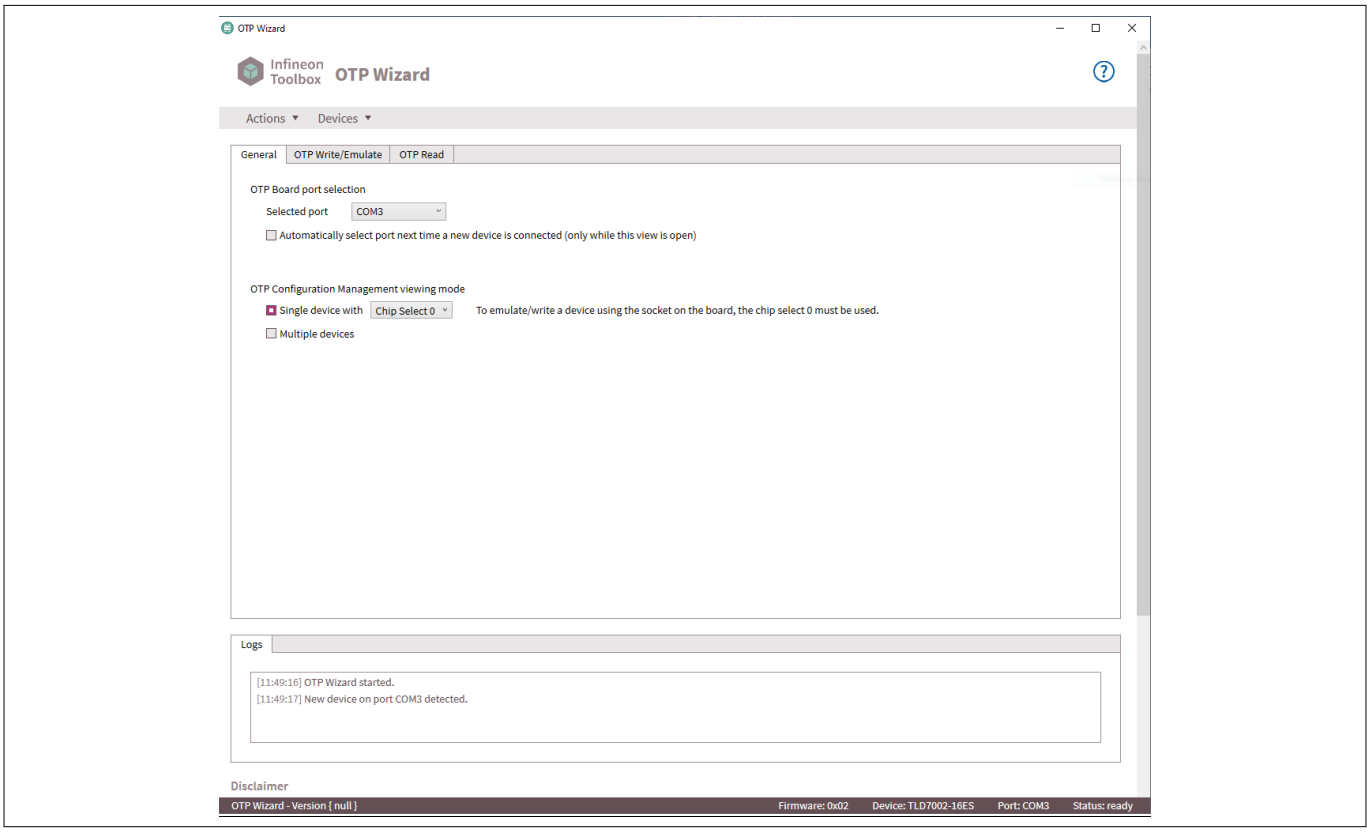

Figure 5 GUI general tab

On the GUI tab, **General**, it is possible to select whether to proceed with the programming/emulation of a single device or multiple devices.

With the multiple devices mode, it is possible to program/emulate up to eight devices in bulk mode using the eight chip select connections (GPIN0 connection) available on the OTP programmer board.

More details about the GUI and the OTP programmer board are available in the user guide of the tool. The user guide is also accessible using the question mark symbol in the upper right-hand corner of the tool.

5.2 OTP read

Use **OTP Read** tab to read OTP.

Prerequisites:

- COM port selected on the **General** tab
- Device supplied and connected to the OTP programmer board (more information available on the user guide of the tool)
- Device address defined in the property **SLAVE ID**

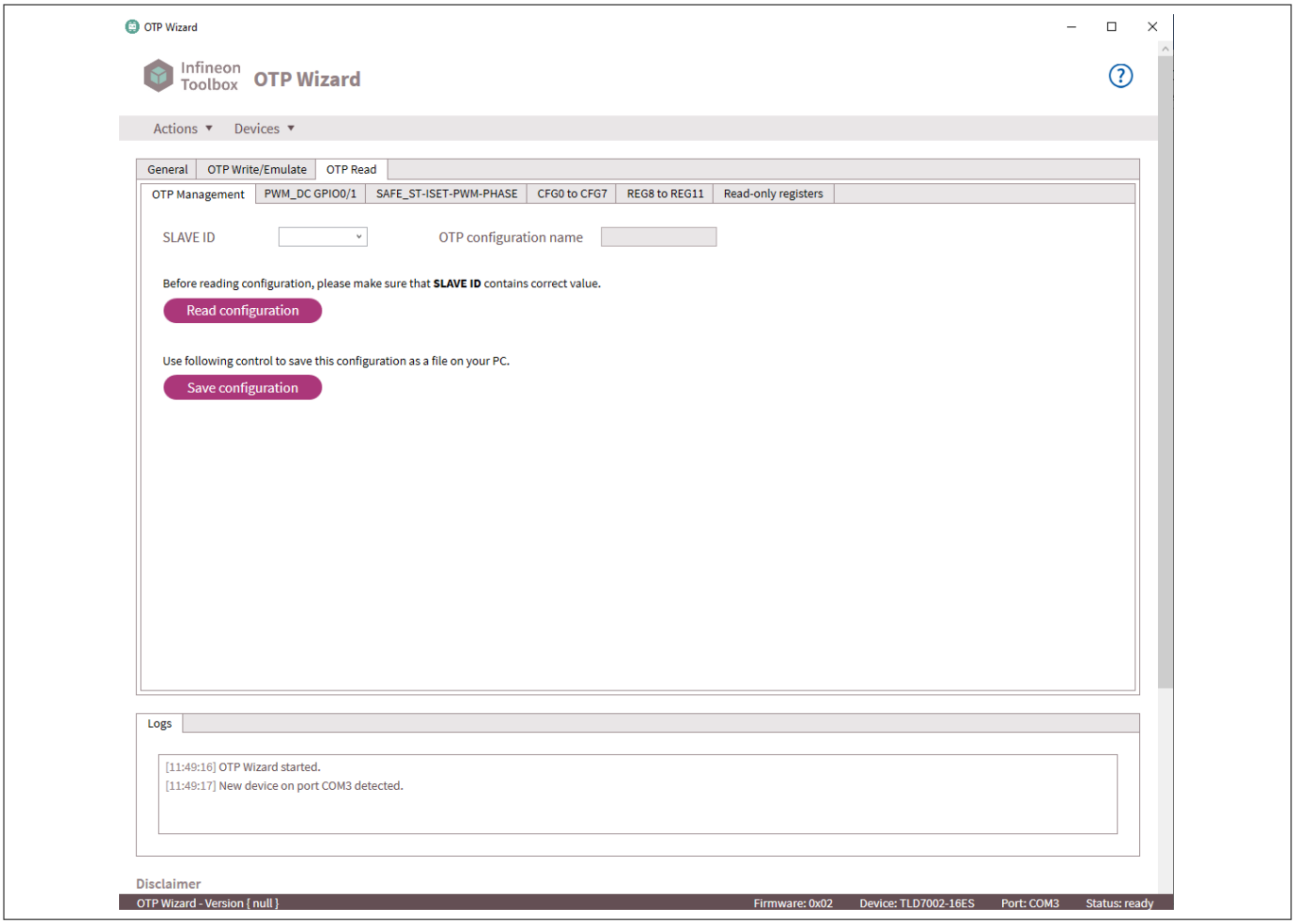

Figure 6 GUI - OTP read

Click on the **Read configuration** button to read the OTP configuration of the device.

5.3 OTP write/emulate

The OTP memory can be written or emulated using the tab **OTP Write/Emulate**.

Prerequisites:

- COM port selected on the **General** tab
- Device/Devices supplied and connected to the OTP programmer board (more information available on the user guide of the tool)
- Device/Devices configuration already defined or loaded

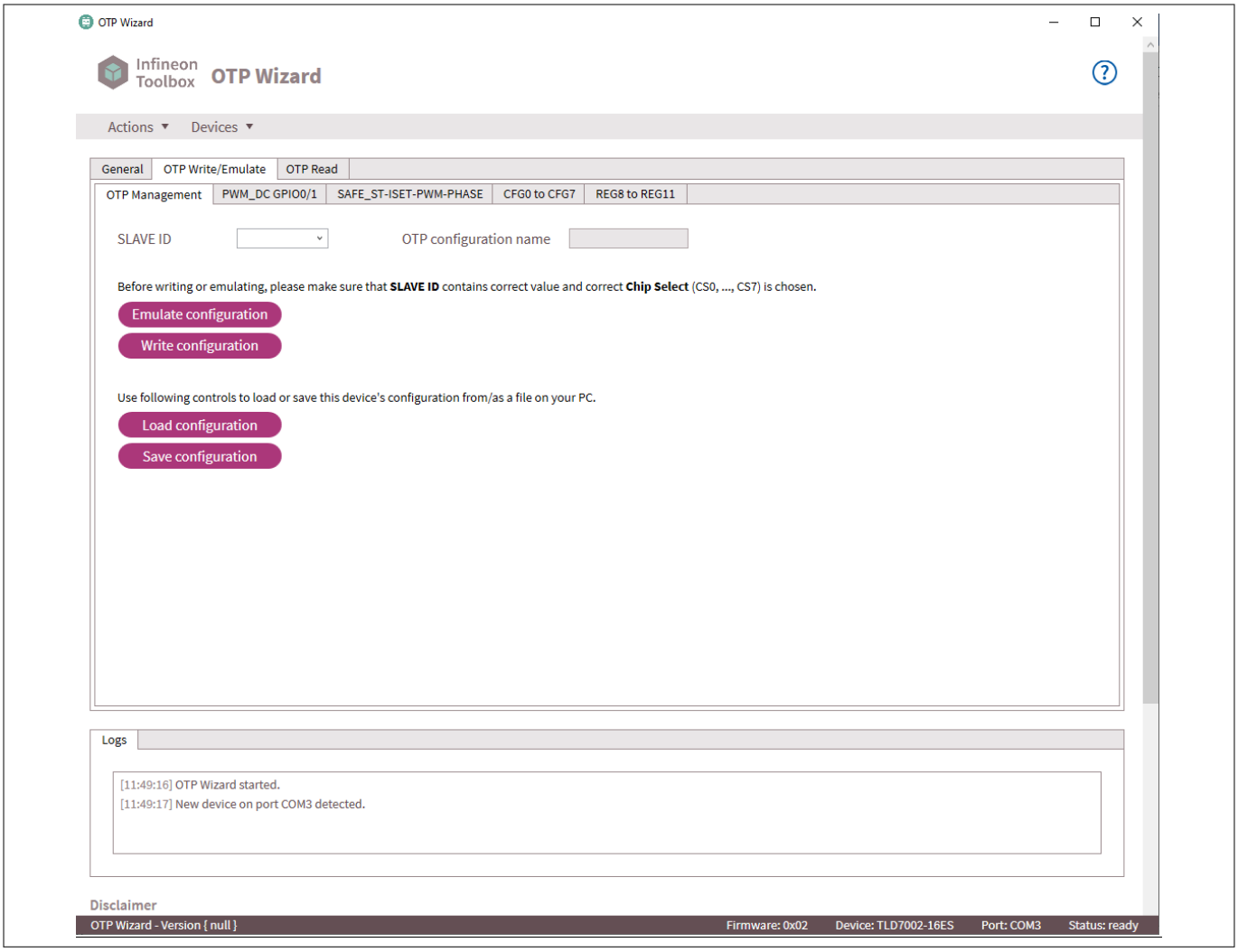

Figure 7 GUI - OTP write and emulate

5 OTP Wizard Tool

On the sub tabs **OTP Management** to **REG8 to REG11** it is possible to define the configuration of the device if the **Single device** mode is selected.

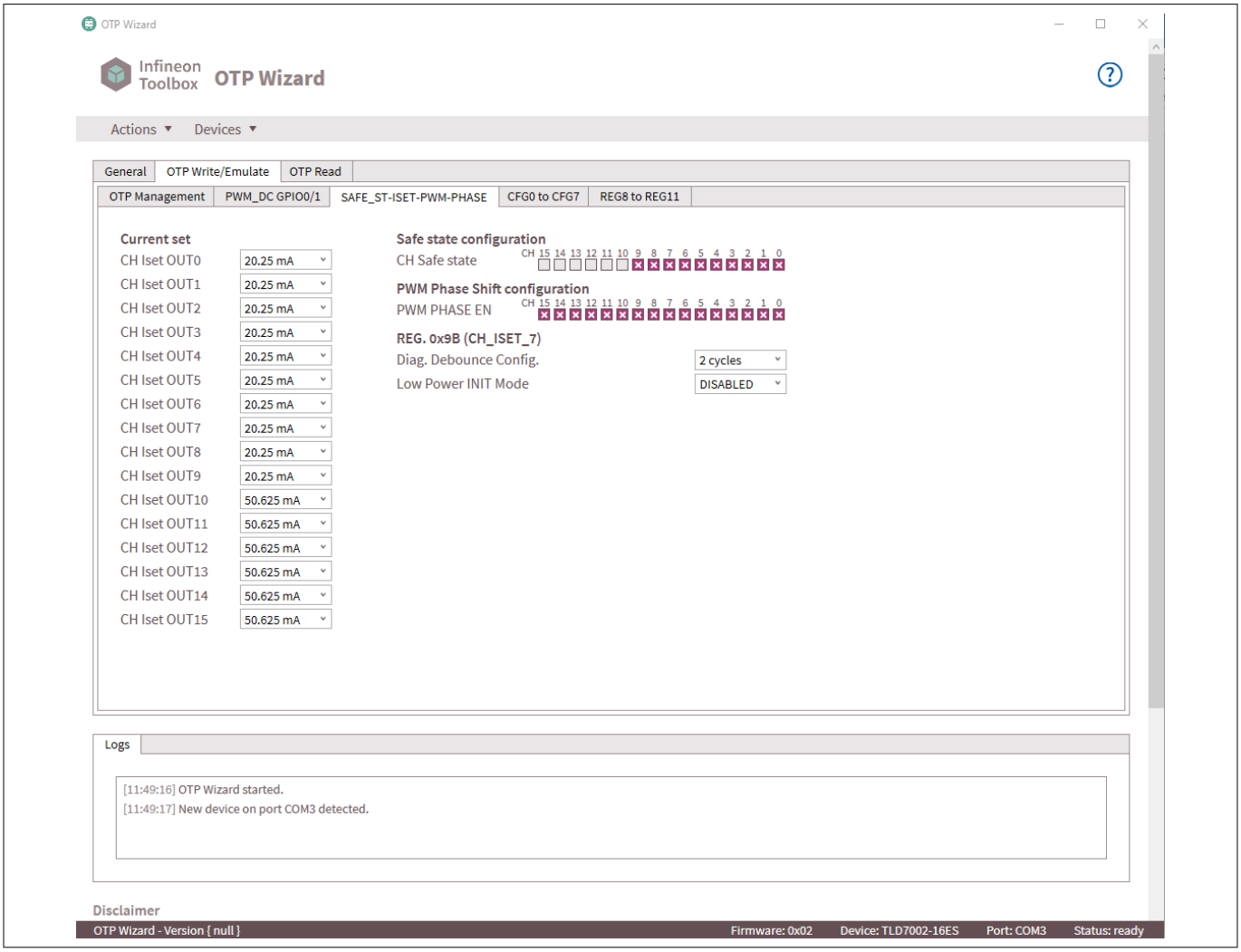

Figure 8 GUI – OTP device configuration example for a single device

5 OTP Wizard Tool

If the **Multiple devices** mode is selected it is possible to define the configuration of up to eight devices:

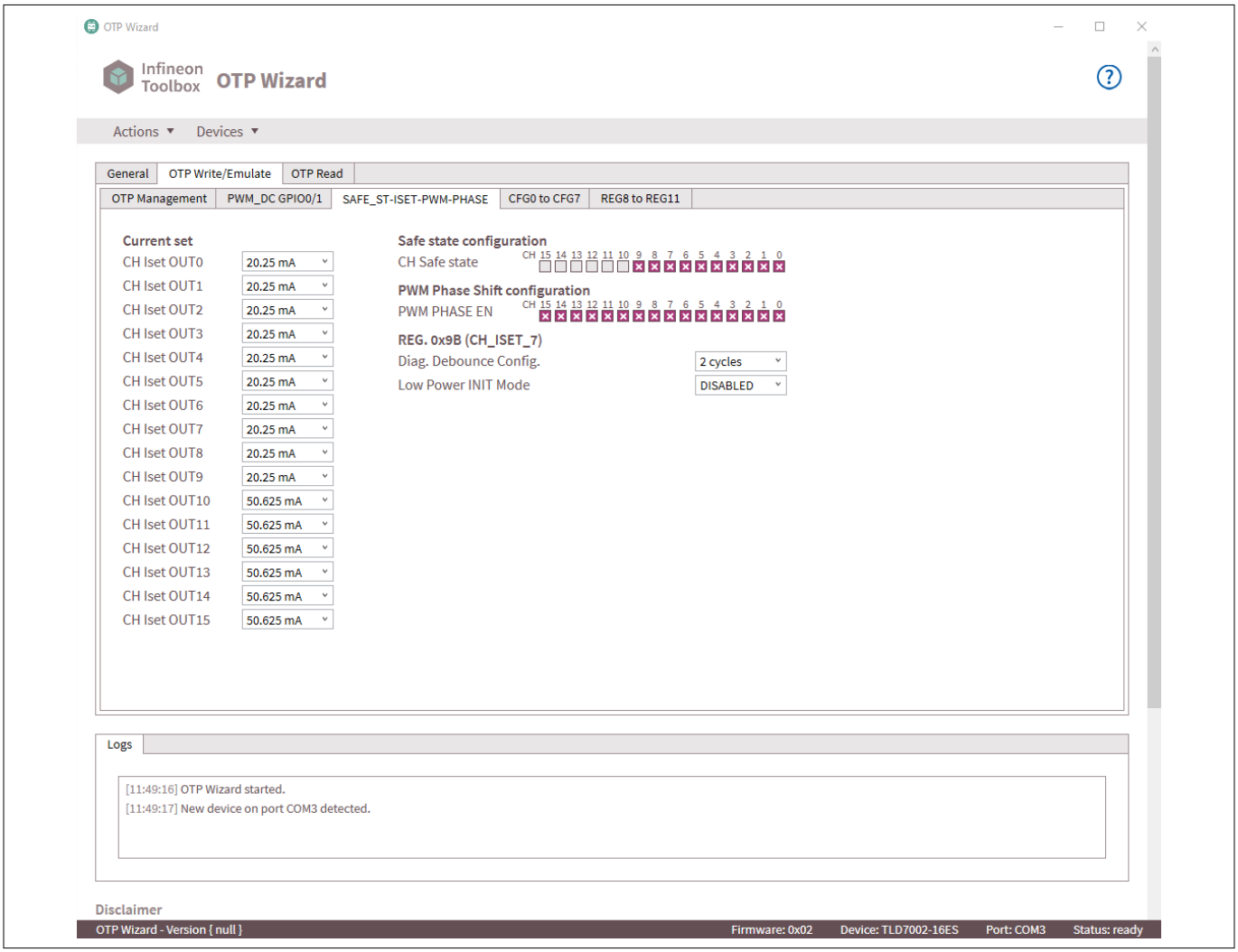

Figure 9 GUI – OTP device configuration example for multiple devices

5 OTP Wizard Tool

If the **Multiple devices** mode is selected, with the tab **Bulk OTP Configuration Management** it is possible to define which devices must be programmed in bulk and then proceed with the desired operation (write or emulate).

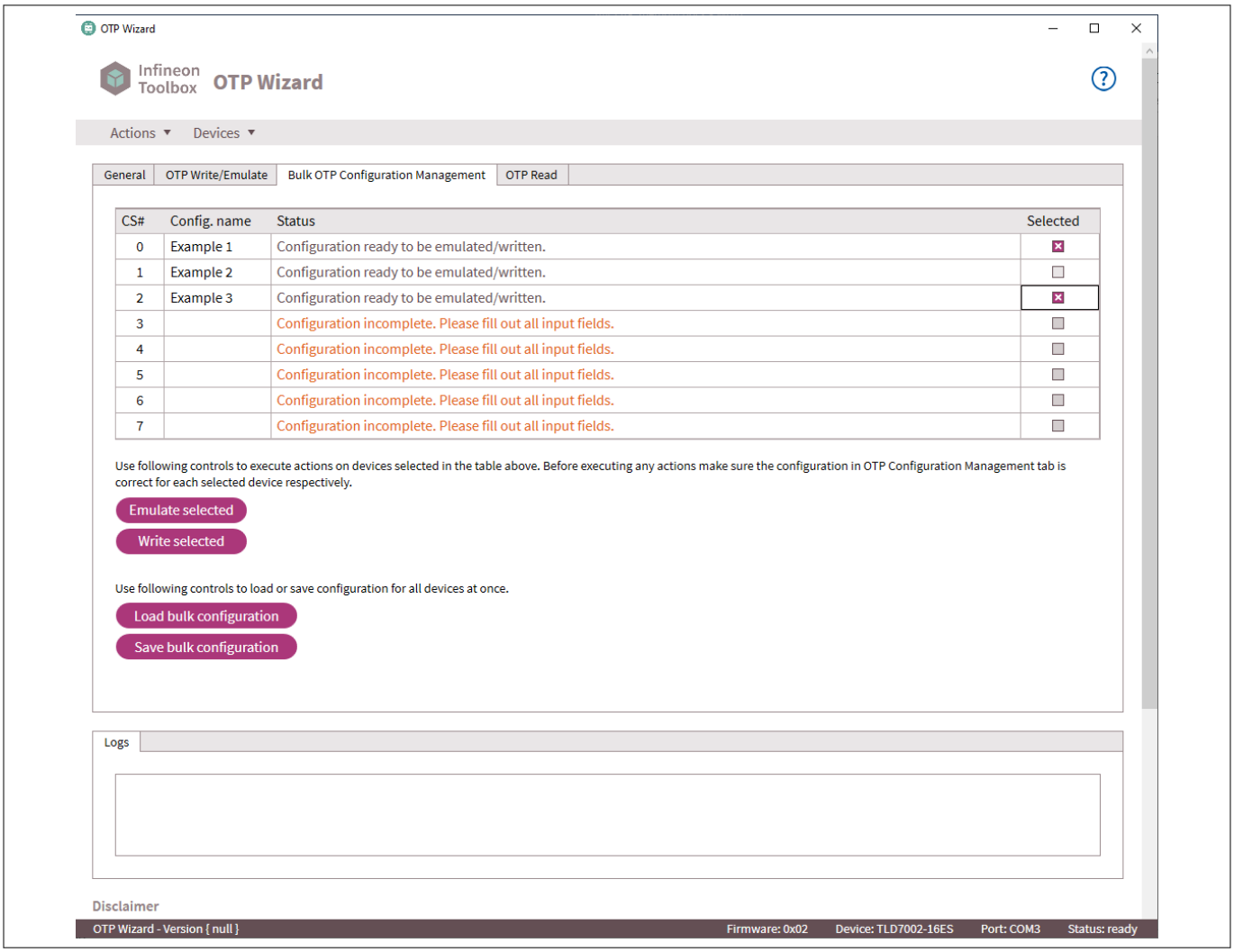

Figure 10 GUI – OTP bulk device configuration example for multiple devices

6 OTP definition and programming example

In this chapter, an example of the OTP configuration and a pseudocode example of the OTP programming sequence is presented.

6.1 System requirements used for the example

- 16 channels used, two red LED in series with maximum forward voltage of 2.6 V per LED and a minimum forward voltage of 1.8 V per LED, LED string current of 20 mA
- Load supply voltage of 6.5 V
- HSLI refresh rate of 5 ms, output PWM frequency of 300 Hz
- GPIN0 not used, GPIN1 used to sense an external NTC
- All channels "on", in safe state with 100% of duty cycle
- All the outputs mapped to VS for the diagnosis, diagnosis of “short between adjacent pin disabled”, “one fail all on” configuration requested, phase shift enabled in all the channels
- The device is controlled using the HSLI bus by a µC, the diagnosis is read by the µC and reported to the LCU

6.2 Definition of the OTP configuration using the OTP Wizard Tool

The address (**SLAVE ID**) 1 (one) has been selected for this example.

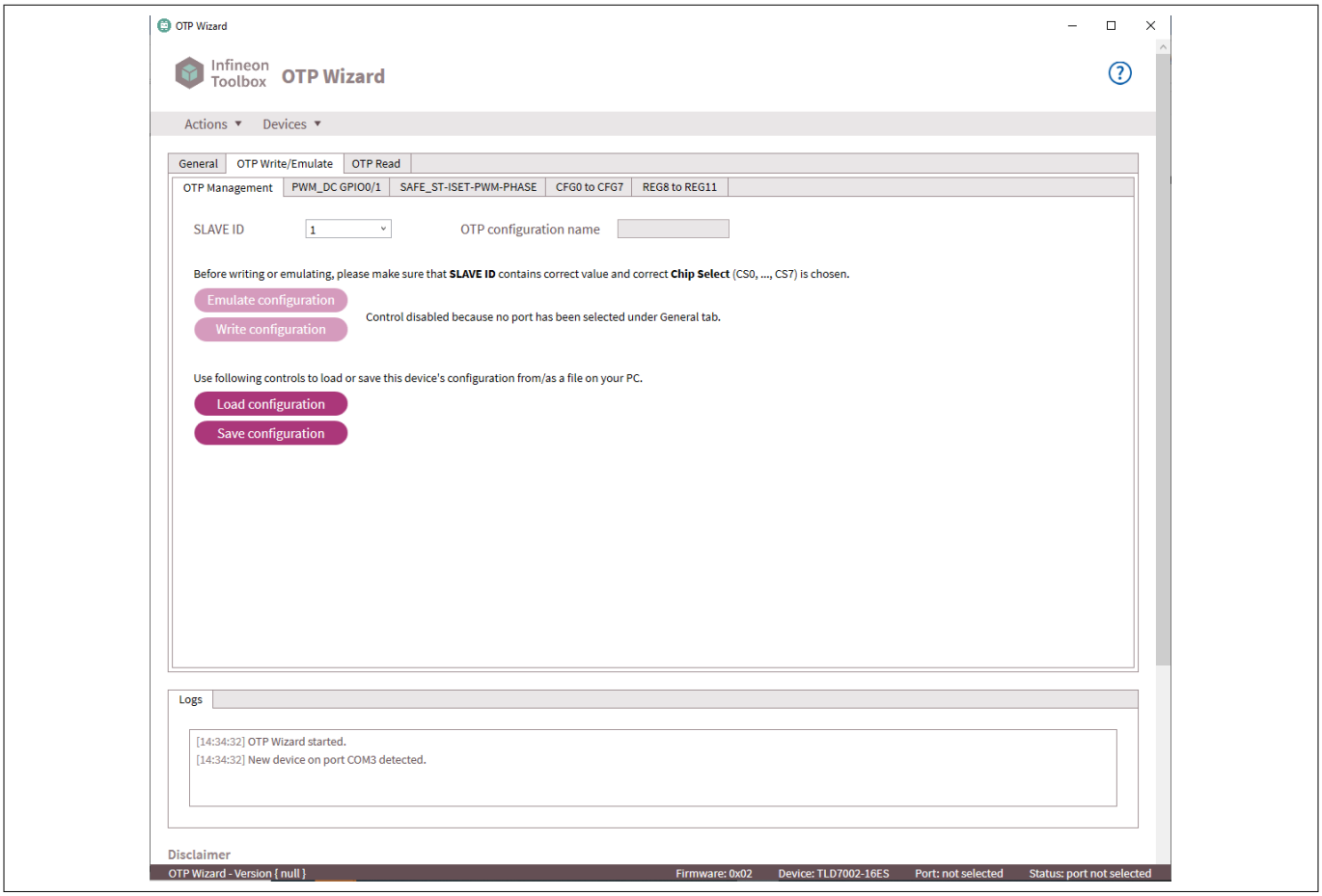

Figure 11 GUI – OTP Management tab

The tab **PWM_DC GPIN0/1** contains the duty cycle configurations of the output channels when controlled via the GPINn or in safe state.

- GPIN1 is not used to control the outputs and so the **DC OUTx for GPIN1** parameters have been set to 0%
- GPIN0 is not used as well to control the output channels but in the current example it is requested to have all the channels active in fail safe mode with a duty cycle of 100% and so the **DC OUTx for GPIN0** parameters have been set to 100%

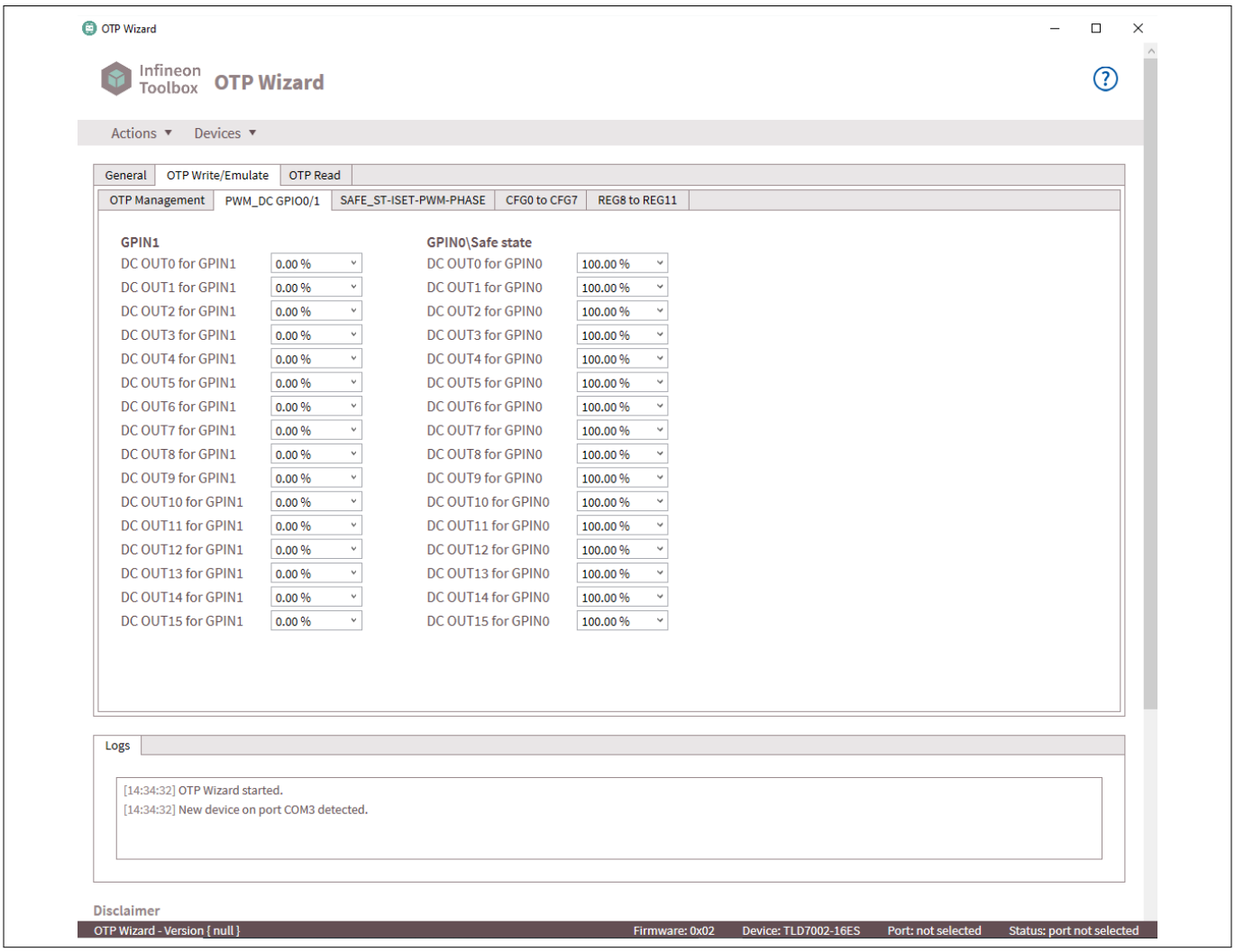

Figure 12 GUI – PWM_DC GPIN0/1 tab

The tab **SAFE_ST-ISET-PHASE** contains the configurations for the output currents, for the safe state, for the PWM phase shift and other configuration related to the diagnosis.

Output currents, phase shift and safe state configurations

- The output currents have been set to 20 mA (**CH Iset OUTx** properties)
- The phase shift has been activated on all the channels (**PWM PHASE EN** checked for all the channels)
- All channels have been set active in safe state (**CH Safe state** checked for all the channels)

Diagnostic configurations

- The number of debounce cycles (**Diag. Debounce Config.**) in case of fault has been set to 2
- The **Low Power INIT Mode** has been disabled since the diagnosis requested is **One Fail all ON** (see tab **CFG0 to CFG7**) and the faults are reported to the LCU via the bus and not via a low current consumption on the power line

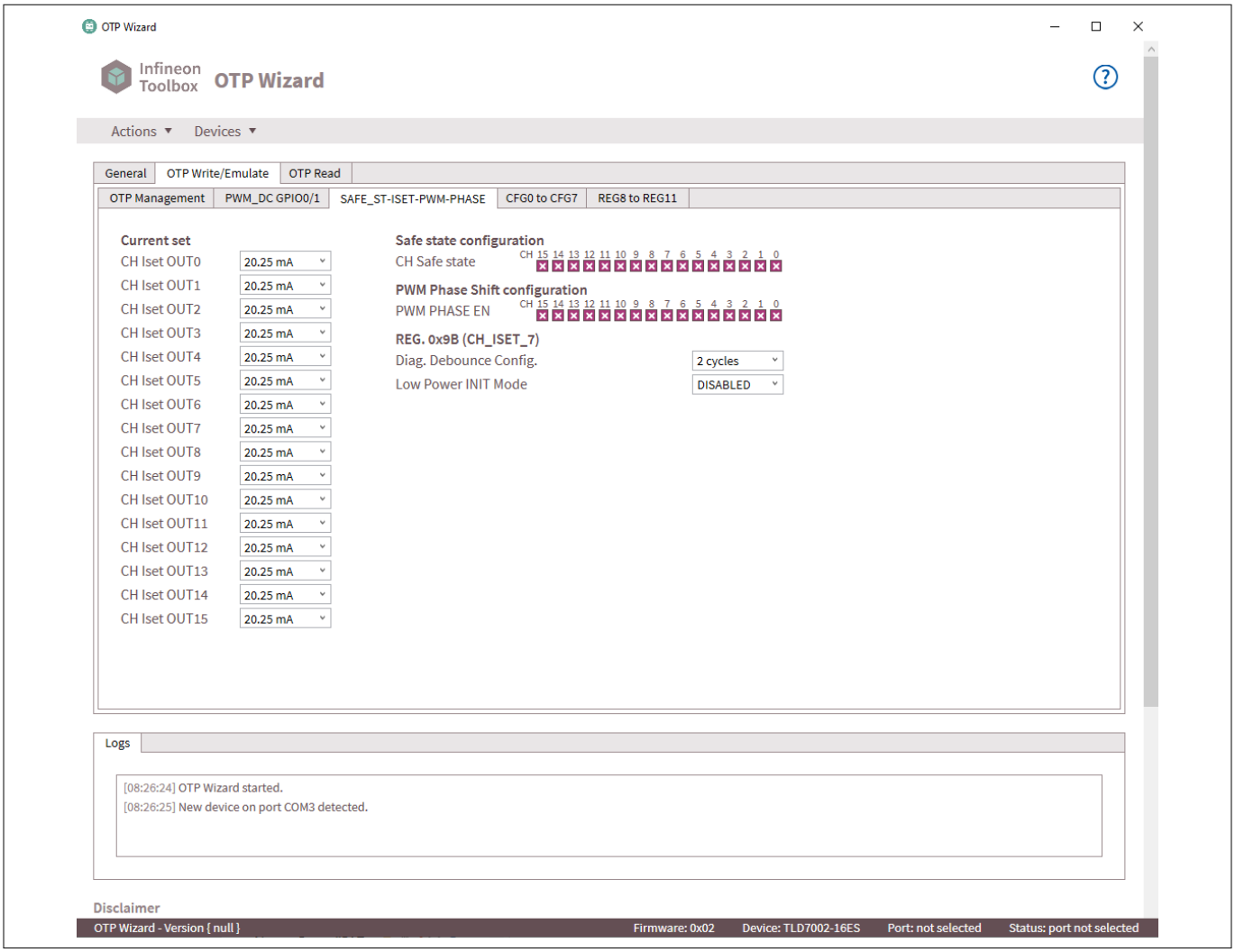

Figure 13 GUI – SAFE_ST-ISET-PWM-PHASE tab

Tab **CFG0 to CFG7** contains all the configurations related to the GPINn behavior, configurations related to the diagnosis and other configurations.

GPINn configurations:

- In this example GPIN1 is used as analog input (**GPIN1 Operating Mode**) to read an external NTC while GPIN0 is not used
- Both GPIN0 and GPIN1 do not control any output channel (**OUT mapped to GPIN0** and**OUT mapped to GPIN1** are not checked)

Diagnostic configurations

- All outputs are mapped to VS for the diagnosis (all the channels on the property **Diag. Output Group to VLED** are unchecked), the device will use the voltage value on the pin VS to perform the diagnostic checks
- Since the load supply voltage in this example is 6.5 V, the diagnosis activation threshold on VS (**VDEN Threshold for VS**) has been set to 6.27 V (at 6.27 V the outputs are fully regulated even with the maximum LED forward voltage)
- The property **VDEN Threshold for VLED** has been set to 0.00 V since there are not channels mapped to VLED for the diagnosis. If the VLED pin is connected in parallel to VS (even if not used), it is a good practice to set the same threshold for both the pins
- Diagnosis of the short between adjacent pins are disabled (**Short WRN enable** not checked) and so the **DIAG OUT SW OFF if DC=100%** is disabled
- The single LED short threshold (**SLS threshold VS**) has been set to 3.462 V. The voltage is higher than the maximum single LED forward voltage but lower than the minimum LED string forward voltage (refer to Figure 14)
- **SLS threshold VLED** has been set to 0 V since all output channels are mapped to VS and the property is not used
- The short to supply voltage thresholds of the output channels has been set to 1.25 V for both VS and VLED (**Threshold short to VLED** and **Threshold short to VS**) but only the **Threshold short to VS** will be used by the device since all the output channels are mapped to VS for the diagnosis. If the voltage difference between VS and an output channel drops below 1.25 V the device reports the short to power supply
- The **Fault Management Configuration** in this example has been set to **One Fail all ON**
- The **ERRN enable** has been set to **DISABLED** since the diagnosis information are reported via bus and the **Current WRN ERRN Report** has been set to **DISABLED** since the ERRN feature is not used

Other configurations

- The **PWM Frequency** has been set to 300.48 Hz and the **PWM Phase shift time** to 4.095% of the PWM period
- The **Watchdog timeout** is set to 1000 ms
- The **Lock of SLS TH** is set to **UNLOCKED** to have the possibility to update the single LED short threshold via the HSLI interface
- The power off load feature is disable (**Power OFF LOAD EN** set to **DISABLED**)

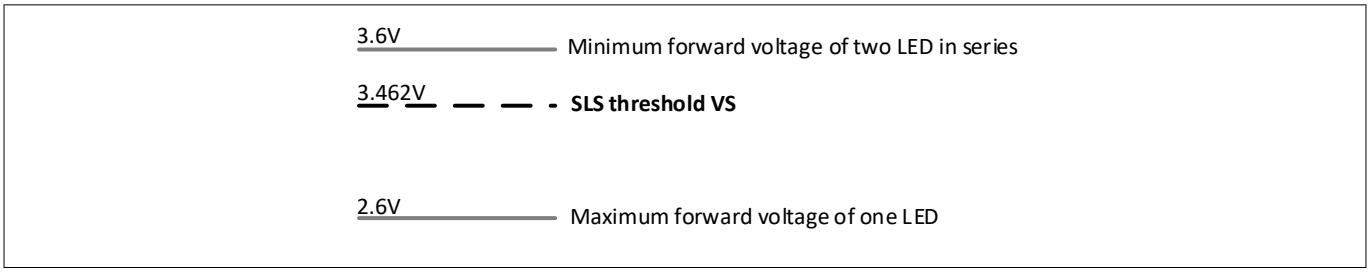

Figure 14 SLS threshold VS set

6 OTP definition and programming example

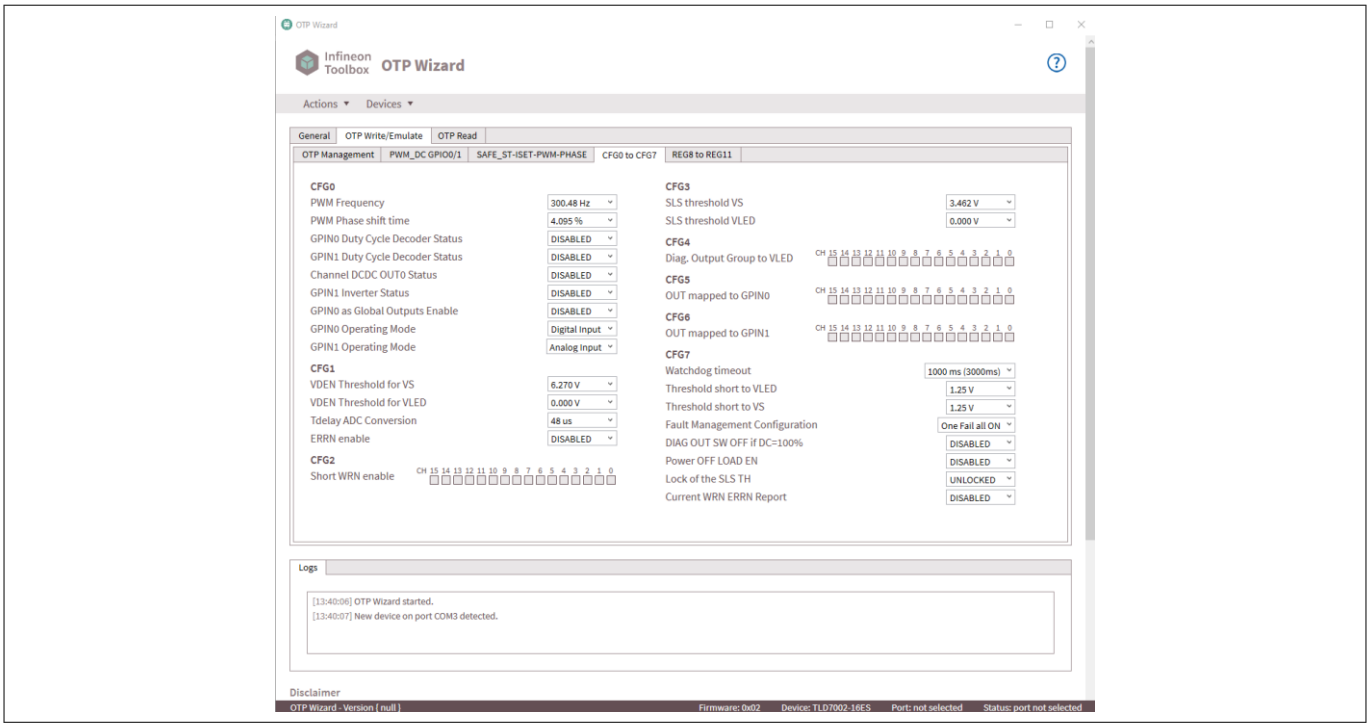

Figure 15 GUI – CFG0 to CFG7 tab

The tab **REG8 to REG11** contains the configuration for the power off load feature, the HSLI bus configurations, the turn-on/off time of the output channels and the **General Purpose Word** where it is possible to store a custom number with up to 4 hexadecimal digits.

Power off load configurations

- This feature is not used in this example and the properties **PWR OFF Load CH x-y Status** have been disabled therefore, whatever the values set in the properties **PWR OFF Load TH CHx-y**, they are not used by the device

HSLI bus configuration

- The **HSLI Sampling Time** has been set to 7,8,9, the HSLI sync break time (**HSLI SYNC BRK Time**) to 1 ms and the intra frame delay (**HSLI Intra-Frame Delay**) to 50 µs

Output channels slew rate

- To improve the EMC/EME the turn-on/off time (**LED Driver Ramp Status**) has been set to **ENABLED** which means the switching mode is set to normal

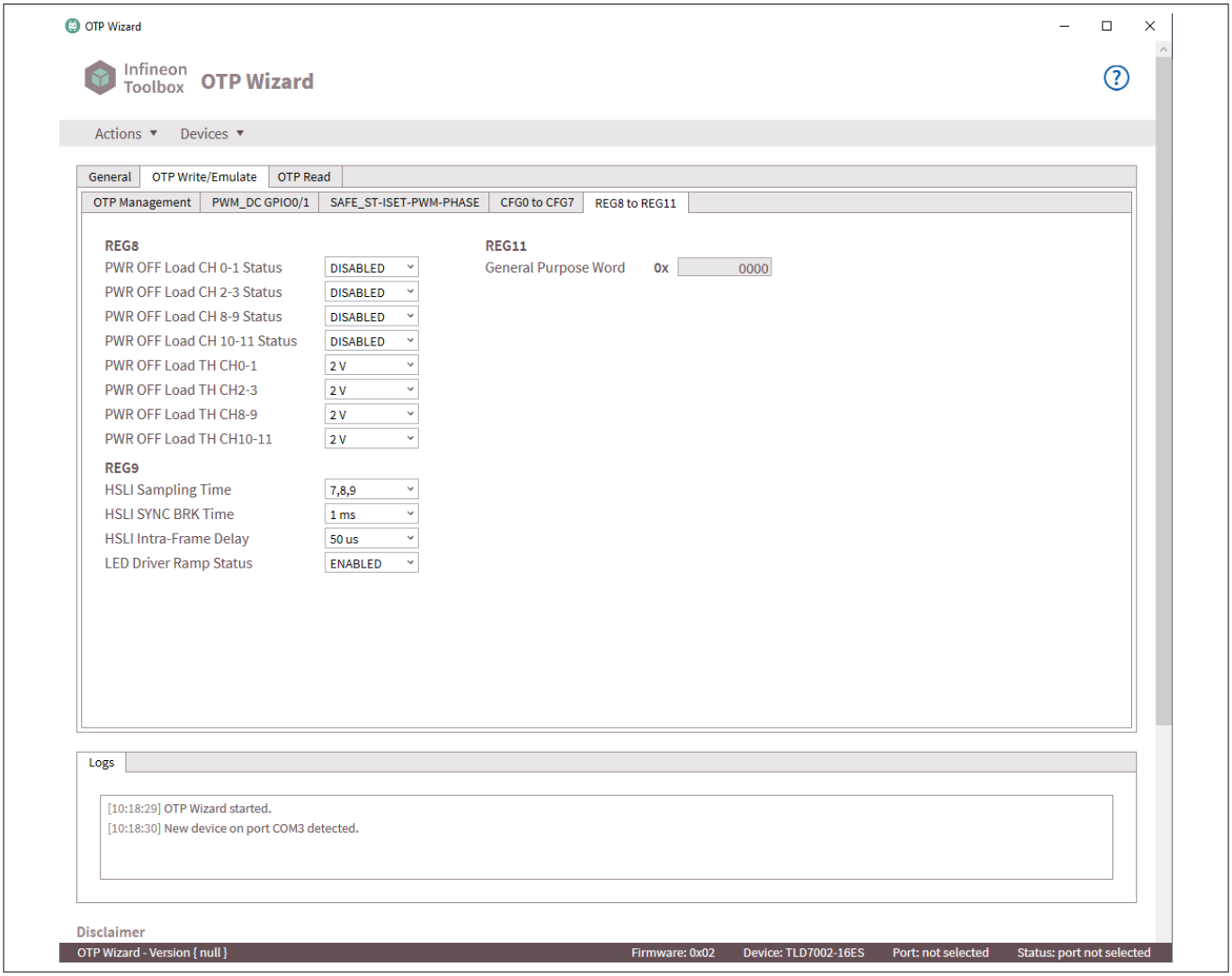

Figure 16 GUI – REG8 to REG11 tab

6.3 Example of the OTP programming procedure with pseudocode

The following chapters show an example of the OTP programming procedure base on the OTP content of the previous example used to show the use of the OTP Wizard tool.

The OTP procedure is presented with a pseudocode based on the Infineon Low Level Driver library of the TLD7002-16ES.

The OTP configuration array is prepared in advance. This can be easily achieved by clicking **save configuration** in the OTP wizard tool.

The configuration array is located at the end of the .ocfg saved file: in the **HEX_DATA_16BIT** field. The file is of txt format, and can be opened with a text editor.

6.3.1 OTP writing procedure with pseudocode

```
#include "TLD7002LowLevelAccess.h"

/********************************************************************************* Define section
*********************************************************************************/
#define DEVICE_ADDRESS     2 // device address after OTP write
#define DEVICE_ADD_DEFAULT 1 // if not programmed the OTP has default address 1
#define INTERFR_DELAY_50 50  //[ms] in this example interframe delay has been set to 55us (the
minimum possible )
#define PWM_PERIOD_uS    3300//[us] output PWM period, default is 300Hz=>3.3us

#define PIN_GPIN0        7  // GPIO connected to TLD7002-16 GPIN0 , used to enable OTP
programming

/********************************************************************************* Variable
section *********************************************************************************/
// buffFrame : buffer used to store HSLI transmission reception frames,
TLD7002_WRITE_REG_DLC7_FRAME_t is the Longest HSLI frame (aprox 171 bytes + 5 byte padding)
// this buffer is used instead of multiple specific frame structures (one per frame type) for
microcontrollers with small memory size
char    buffFrame [sizeof (TLD7002_READ_OST_FRAME_t)+5]; // 5 bytes of padding,

// OTP configuration array, coming from file: "EVB_ADD1_B1x_TAIL_Load_V2.0.ocfg"
const uint16 OTP_hex_cfg[]= {0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x990C, 0x990C, 0x990C, 0x990C, 0x990C, 0x990C, 0x990C, 0x990C, 0xFFFF, 0x0820, 0x0820, 0x0820,
0x0820, 0x0820, 0x0820, 0x0820, 0x1820, 0xFFFF, 0x80F3, 0x094A, 0x0000, 0x2020, 0x0000, 0x0000,
0x0000, 0x8007, 0x0001, 0xAAA3, 0x0000, 0x000C, 0x0000, 0x8111};

//********************************************************************************* CODE
*********************************************************************************
// setup function, called only once at startup, typically hardware and SW modules
initialization are performed here
void setup() {
  initMCLDSerial();
  TLD7002_Init(sendMCLDMessage, readReceivedMCLDAnswer, emptyingReceiveBuffer,
generateStartSyncBreak, generateStopSyncBreak);

  if ( OTP_Write(OTP_hex_cfg, DEVICE_ADDRESS, INTERFR_DELAY_50) == true)
    Serial.println( "OTP WRITE SUCCESS");
  else
    Serial.println( "OTP WRITE ERROR");
}

void loop() {
}

// OTP write PROCEDURE (NOT TESTED YET):
// warning, working only on not written devices
bool OTP_Write(uint16 *otp_cfg, uint16 address, uint16 newInterfrDelay  )
{
  uint8 res;
```

```
  bool  ret= true;
  bool  readBackMatch=true;

  digitalWrite (PIN_GPIN0,LOW); // ensure GPIN0 is LOW
  //  Step1: send command PM_CHANGE_FRAME to INIT

TLD7002_TX_PM_CHANGE_FRAME((TLD7002_PM_CHANGE_FRAME_t*)buffFrame,TLD7002_FRAME_SLAVE_ADDRESS_BRO
ADCAST, TLD7002_FRAME_PM_INIT_MODE);
  delayMicroseconds(1005); /*delay counted starting from the last bit transmitted*/

  // Step2: send command PM_CHANGE_FRAME twice to synchronize LLD library master rolling
counter and TLD7002 rolling counter

TLD7002_TX_PM_CHANGE_FRAME((TLD7002_PM_CHANGE_FRAME_t*)buffFrame,TLD7002_FRAME_SLAVE_ADDRESS_BRO
ADCAST, TLD7002_FRAME_PM_INIT_MODE);
  delayMicroseconds(1005);

  // Step3: set interframe delay to 50us
  TLD7002_TX_WRITE_REG_DLC1_FRAME((TLD7002_WRITE_REG_DLC1_FRAME_t*)buffFrame,
TLD7002_FRAME_SLAVE_ADDRESS_BROADCAST, 0x3B, 0x000C);
  delayMicroseconds(50); /*delay counted starting from the last bit transmitted*/

  // Step4: set the TLD7002-16 in OTP mode

TLD7002_TX_PM_CHANGE_FRAME((TLD7002_PM_CHANGE_FRAME_t*)buffFrame,TLD7002_FRAME_SLAVE_ADDRESS_BRO
ADCAST, TLD7002_FRAME_PM_OTP_MODE);
  delayMicroseconds(1005); /*delay counted starting from the last bit transmitted*/

  // Step5: set GPIN0 HIGH to enable programming
  digitalWrite (PIN_GPIN0,HIGH); //Set_GPIN0_to_high_value();
  delayMicroseconds(50);

  // Step6: write OTP Programming passw

TLD7002_TX_WRITE_REG_DLC1_FRAME((TLD7002_WRITE_REG_DLC1_FRAME_t*)buffFrame,TLD7002_FRAME_SLAVE_A
DDRESS_BROADCAST, 0x81, 0xA47B);// write OTP write password in 0x81
  delayMicroseconds(50); // delay counted starting from the last bit transmitted

  // Step7: write first 32 words

TLD7002_TX_WRITE_REG_DLC7_FRAME((TLD7002_WRITE_REG_DLC7_FRAME_t*)buffFrame,TLD7002_FRAME_SLAVE_A
DDRESS_BROADCAST, 0x83, OTP_hex_cfg);
  delay(18); // wait 17,5ms, OTP write is slow, delay counted starting from the last bit
transmitted*/

  // Step8: Write last 8 words

TLD7002_TX_WRITE_REG_DLC4_FRAME((TLD7002_WRITE_REG_DLC4_FRAME_t*)buffFrame,TLD7002_FRAME_SLAVE_A
DDRESS_BROADCAST, 0xA3, &OTP_hex_cfg[32]);
  delay(5); // wait 5ms, OTP write is slow

  // Step9:
  digitalWrite (PIN_GPIN0,LOW); //  Set GPIN0 to low_value();
```

```
  delayMicroseconds(INTERFR_DELAY_50);

  // Step 10:

TLD7002_TX_PM_CHANGE_FRAME((TLD7002_PM_CHANGE_FRAME_t*)buffFrame,TLD7002_FRAME_SLAVE_ADDRESS_BRO
ADCAST, TLD7002_FRAME_PM_INIT_MODE);
  delayMicroseconds(max(newInterfrDelay, 205)); //delay counted starting from the last bit
transmitted, delay shall be:delay > 200us or bigger than the just configured interfr.del.

  // Step 11:
  TLD7002_TRX_HWCR_ALL (buffFrame,  address); //First TRX command with device address!=
broadcast, result is probably bad becasue RC is not sync with Device drivers library
  delayMicroseconds(newInterfrDelay);       //wait > new programmed interframe delay

  // Step 12  // read OTP_STATUS register
  res = TLD7002_TRX_READ_REG_DLC1(buffFrame, address, TLD7002_OTP_STATUS);
  delayMicroseconds(newInterfrDelay);       // wait > new programmed interframe delay
  if ( res == TLD7002_FRAME_VAL_NO_ERROR )   // received frame is valied
  {
    if(  ((TLD7002_READ_REG_DLC1_FRAME_t*)buffFrame)->r_read_reg.Data[0] == 0x03) // OTP_STATUS
is as expected for correct OTP write
    {
        Serial.println("OTP wr:OTP_STATUS OK = 0x03");
    }
    else
    {
      Serial.print("OTP wr: err OTP_STATUS=0x");
      Serial.println(((TLD7002_READ_REG_DLC1_FRAME_t*)buffFrame)->r_read_reg.Data[0],HEX);
      ret= false;
    }
  }
  else
  {
    Serial.print("OTP wr: Err frame1 res=");
    Serial.println(res);
    ret= false;
  }

  // Step13: set the TLD7002-16 in OTP mode
  TLD7002_TRX_PM_CHANGE(buffFrame,address, TLD7002_FRAME_PM_OTP_MODE);
  delayMicroseconds(newInterfrDelay); // the interframe shall be counted starting from the last
bit transmitted from the TLD7002-16 (slave response)

  // STEP14 and Step 15: read back the entire OTP, unified read lenght with read 5 times 8 word
from the OTP and check
  for (int n =0;n<5;n++)
  {
    res = TLD7002_TRX_READ_REG_DLC4(buffFrame,address, 0x83+n*8);
    delayMicroseconds (newInterfrDelay);//wait interframe delay after TLD7002-16 answer

    if ( res == TLD7002_FRAME_VAL_NO_ERROR ){
      for (int i =0;i<8 ;i++ ) // lenght of DLC 6 is 16 words
(TLD7002_LEN_WRITE_REG_DLC6_WRITE-TLD7002_LEN_WRITE_REG_OVHD)/2
```

```
      {
        if ( ((TLD7002_READ_REG_DLC4_FRAME_t*)buffFrame)->r_read_reg.Data[i]!=
OTP_hex_cfg[i+n*8] ) // compare readed OTP with written OTP cfg
        {
          Serial.println("OTP wr: Err read Back Mismatch");
          ret= false;
          readBackMatch = false;
          break;
        }
      }
    }
    else
    {
      Serial.print("OTP wr: Err readBack frame err res=");
      Serial.println(res);
      ret= false;
      readBackMatch = false;
      break;
    }
  }
  if (readBackMatch == true)
    Serial.println("OTP wr: OTP readBack OK");

  // STEP 16 check output status FAULT BIT
  delayMicroseconds(2*PWM_PERIOD_uS); // wait for 2 PWM periods from the last HWCR command, CRC
is checked 1 per PWM period, this is the worst case
                                      // time needed from the TLD7002-16 to highlight eventual
CRC errors on the OUTPUT STATUS BYTE
  res = TLD7002_TRX_READ_REG_DLC1(buffFrame, address, TLD7002_OTP_STATUS);// dummy read reg, to
obtain the OUTPUT STATUS BYTE reply back
  if ( res == TLD7002_FRAME_VAL_NO_ERROR){
    if ( ((TLD7002_PM_CHANGE_FRAME_t*)buffFrame)->r_power_mode_change.std_answer.OST.OSB_FAULT
== 0){ // check if the OUTPUT STATUS byte FAULT bit is 0
      Serial.println("OTP wr: OK FAULT=0 ");
    }
    else{
        Serial.println("OTP wr: Err-OSB_FAULT=1");
        ret= false;
    }
  }
  else {
    Serial.println("OTP wr: Err-frame2");
    ret= false;
  }

  return ret;
}

// ****************************   Wrappers TX,RX combined and HWCR All
************************************************************

// combined TX and RX READ_REG_DLC1 function, with correct wait time in order to receive the
entire TLD7002-16 reply
```

```
uint8 TLD7002_TRX_READ_REG_DLC1(char* buffFrame, uint8 add, uint8 regStartAdd)
{
  TLD7002_TX_READ_REG_DLC1_FRAME((TLD7002_READ_REG_DLC1_FRAME_t*)buffFrame, add,
regStartAdd);// read OTP status register
  delayMicroseconds(TLD7002_LEN_READ_REG_DLC1_READ*BYTE_TIME_uS ); // wait > data from
TLD7002-16 to be available
  return TLD7002_RX_READ_REG_DLC1_FRAME((TLD7002_READ_REG_DLC1_FRAME_t*)buffFrame);// send and
receive twice to syncronize the LLD receiver rolling counter
}

uint8 TLD7002_TRX_READ_REG_DLC4(char* buffFrame, uint8 add, uint8 regStartAdd)
{
  TLD7002_TX_READ_REG_DLC4_FRAME((TLD7002_READ_REG_DLC4_FRAME_t*)buffFrame, add,
regStartAdd);// read OTP status register
  delayMicroseconds( TLD7002_LEN_READ_REG_DLC4_READ*BYTE_TIME_uS ); // wait > data from
TLD7002-16 to be available
  return TLD7002_RX_READ_REG_DLC4_FRAME((TLD7002_READ_REG_DLC4_FRAME_t*)buffFrame);// send and
receive twice to syncronize the LLD receiver rolling counter
}


uint8 TLD7002_TRX_DC_UPDATE_14BIT(char* buffFrame, uint8 add, uint16* dc_val)
{
  TLD7002_TX_DC_UPDATE_14BIT_FRAME((TLD7002_DC_UPDATE_14BIT_FRAME_t*)buffFrame, add,
dc_val); // send a DC_UPDATE to move to active mode,  providing also new duty cycle array to
shadow registers
  delayMicroseconds(TLD7002_LEN_DC_UPDATE_14BIT_READ*BYTE_TIME_uS ); // wait > data from
TLD7002-16 to be available
  return TLD7002_RX_DC_UPDATE_14BIT_FRAME((TLD7002_DC_UPDATE_14BIT_FRAME_t*)buffFrame);
}

uint8 TLD7002_TRX_DC_UPDATE_DLC0(char* buffFrame, uint8 add)
{
  TLD7002_TX_DC_UPDATE_8BIT_DLC0_FRAME((TLD7002_DC_UPDATE_8BIT_FRAME_t*)buffFrame, add); //
send a DC_UPDATE to move to active mode,  providing also new duty cycle array to shadow
registers
  delayMicroseconds( TLD7002_LEN_DC_UPDATE_8BIT_DLC0_READ*BYTE_TIME_uS ); // wait > data from
TLD7002-16 to be available
  return TLD7002_RX_DC_UPDATE_8BIT_DLC0_FRAME((TLD7002_DC_UPDATE_8BIT_FRAME_t*)buffFrame);
}

uint8 TLD7002_TRX_PM_CHANGE(char* buffFrame, uint8 add, TLD7002_FRAME_POWER_MODE_CHANGE_t mode)
{
  TLD7002_TX_PM_CHANGE_FRAME((TLD7002_PM_CHANGE_FRAME_t*)buffFrame, add, mode);
  delayMicroseconds( TLD7002_LEN_PM_CHANGE_READ*BYTE_TIME_uS ); // wait > data from TLD7002-16
to be available
  return TLD7002_RX_PM_CHANGE_FRAME((TLD7002_PM_CHANGE_FRAME_t*) buffFrame);
}

uint8 TLD7002_TRX_WRITE_REG_DLC7(char* buffFrame, uint8 add, uint8 regStartAdd, uint16* data)
{
   TLD7002_TX_WRITE_REG_DLC7_FRAME((TLD7002_WRITE_REG_DLC7_FRAME_t*)buffFrame, add,
regStartAdd, data);
```

```
    delayMicroseconds( TLD7002_LEN_WRITE_REG_READ*BYTE_TIME_uS ); // wait > data from TLD7002-16
to be available
    return TLD7002_RX_WRITE_REG_DLC7_FRAME((TLD7002_WRITE_REG_DLC7_FRAME_t*)buffFrame);
}

uint8 TLD7002_TRX_WRITE_REG_DLC4(char* buffFrame, uint8 add, uint8 regStartAdd, uint16* data)
{
    TLD7002_TX_WRITE_REG_DLC4_FRAME((TLD7002_WRITE_REG_DLC4_FRAME_t*)buffFrame, add,
regStartAdd, data);
    delayMicroseconds( TLD7002_LEN_WRITE_REG_READ*BYTE_TIME_uS ); // wait > data from TLD7002-16
to be available
    return TLD7002_RX_WRITE_REG_DLC4_FRAME((TLD7002_WRITE_REG_DLC4_FRAME_t*)buffFrame);
}

uint8 TLD7002_TRX_WRITE_REG_DLC1(char* buffFrame, uint8 add, uint8 regStartAdd, uint16 data)
{
    TLD7002_TX_WRITE_REG_DLC1_FRAME((TLD7002_WRITE_REG_DLC1_FRAME_t*)buffFrame, add,
regStartAdd, data);// write OTP write password in 0x81
    delayMicroseconds( TLD7002_LEN_WRITE_REG_READ*BYTE_TIME_uS ); // wait > data from TLD7002-16
to be available
    return TLD7002_RX_WRITE_REG_DLC1_FRAME((TLD7002_WRITE_REG_DLC1_FRAME_t*)buffFrame);
}

uint8 TLD7002_TRX_HWCR_ALL (char* buffFrame, uint8 add)
{
  TLD7002_TX_HWCR_ALL((TLD7002_HWCR_FRAME_t*)buffFrame,add);
  delayMicroseconds( TLD7002_LEN_HWCR_READ*BYTE_TIME_uS ); // wait > data from TLD7002-16 to be
available
  return TLD7002_RX_HWCR_FRAME((TLD7002_HWCR_FRAME_t*)buffFrame);
}

// TLD7002_TX_HWCR_ALL clear all fault flags for all the outputs
boolean TLD7002_TX_HWCR_ALL (TLD7002_HWCR_FRAME_t* frm, uint8 add) {
  TLD7002_FRAME_HWCR_RESET_OUT_t reset_overload;
  TLD7002_FRAME_HWCR_RESET_OUT_t reset_openload;
  TLD7002_FRAME_HWCR_RESET_OUT_t reset_sls;
  TLD7002_FRAME_HWCR_RESET_STATUS_t reset_status;

  reset_overload.RES_OUT0 = true;
  reset_overload.RES_OUT1 = true;
  reset_overload.RES_OUT2 = true;
  reset_overload.RES_OUT3 = true;
  reset_overload.RES_OUT4 = true;
  reset_overload.RES_OUT5 = true;
  reset_overload.RES_OUT6 = true;
  reset_overload.RES_OUT7 = true;
  reset_overload.RES_OUT8 = true;
  reset_overload.RES_OUT9 = true;
  reset_overload.RES_OUT10 = true;
  reset_overload.RES_OUT11 = true;
  reset_overload.RES_OUT12 = true;
  reset_overload.RES_OUT13 = true;
  reset_overload.RES_OUT14 = true;
```

```
  reset_overload.RES_OUT15 = true;
  reset_status.CUR_WRN     = true;
  reset_status.DC_WRN      = true;
  reset_status.OSB_FAULT   = true;
  reset_status.GPINn_WRN   = true;
  reset_status.OUT_SHRT_WRN= true;
  reset_status.OVLD        = true;
  reset_status.VFWD_WRN    = true;
  reset_status.VLED_VS_UV  = true;

  reset_openload = (TLD7002_FRAME_HWCR_RESET_OUT_t) reset_overload;
  reset_sls      = (TLD7002_FRAME_HWCR_RESET_OUT_t) reset_overload;

  return TLD7002_TX_HWCR_FRAME(frm, add, reset_overload, reset_openload, reset_sls,
reset_status);
}
```

7 List of abbreviations

Table 1 List of abbreviations

Acronym	Description
CRC	Cyclic redundancy check
CRC16	Cyclic redundancy check 16-bit
EMC	Electromagnetic compatibility
EME	Electromagnetic emission
GUI	Graphical user interface
HSLI	High-speed lighting interface
LUT	Lookup table
OTP	One time programmable

References

[1] *TLD7002-16ES Datasheet*, Z8F64248201

Revision history

Document version	Date of release	Description of changes
Rev.1.10	2022-05-03	• Product family name is assigned as LITIX™ Pixel Rear
Rev.1.00	2022-03-31	• Initial release

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2022-05-03

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-vkm1625643329460

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.