TLx493D 3D Hall Sensor Generic Library

1.3

Generated by Doxygen 1.8.14

Contents

Chapter 1

3D Hall Sensor Generic Library

1.1 General Description

The TLx493D Generic Library is a microcontroller-agnostic implementation of a software stack abstraction for sensors of the TLx493D 3D Hall family.

The supported hardware versions are:
∗TLV493D-A1B6
∗TLE493D-A2B6
∗TLE493D-W2B6
∗TLI493D-W2BW

The library presents the following three levels of abstraction:
∗TLx493D abstraction level
∗TLV493D/TLE493D + TLI493D-W2BW abstraction levels
∗Drivers abstraction levels

1.2 Quick start guide

Before calling any library function it is important to note that the functions require the ability to communicate with the 3D Hall sensor on the I2C bus. This implementation of such functionality depends on the microcontroller that the library will be compiled for. As such, the user is required to provide certain functions that when called by the library, will establish communication with the sensor. For details about the aforementioned functions regarding implementation and interfacing with the library, please see the file:
interface.h

The Core distribution of this library does not provide implementations for such functions.
The XMC distribution of the library provides XMC drivers already interfaced with the library.

Follow the TODO comments in the code for additional interfacing instructions.

When importing the library as a project in Dave IDE, only use the Debug Build Configuration, otherwise the project might not build !

1.2.1 TLx493D abstraction level

The library is able to automatically detect the sensor type and version, making it very easy to get started with such a sensor. The downside of using this level is that regardless of the sensor used, only basic functionality is available to the user, and only one sensor can be used at a time (bus mode not supported).

Example code for reading a data frame in **Master Control Mode**:

```
// please note that the value of status should be checked and properly handler
int32_t status;
TLx493D_data_frame_t frame;
int16_t Bx_LSB, By_LSB, Bz_LSB, sensor_temperature_LSB;

// Initialize the sensor
status = TLx493D_init();

// set operation mode to Master Control Mode
status =  TLx493D_set_operation_mode(TLx493D_OP_MODE_MCM);

// read a data frame
status = TLx493D_read_frame(&frame);

// Copy Magnetic Field Intensity and temperature values in LSB format
Bx_LSB = frame.x;
By_LSB = frame.y;
Bz_LSB = frame.z;
sensor_temperature_LSB = frame.temp;
```

For this abstraction level please see the following files:
TLx493D.h
TLx493D.c

1.2.2 TLV493D/TLE493D/TLI493D-W2BW abstraction levels

By using the functions defined at this abstraction level, the user may change most of the sensor parameters while also not having to manually change register values or ensure that parity values are correct. This level presents a balance between ease of use and access to sensor settings and is the **recommended mode for testing sensor features**. The sensor type and version must be known by the user (it is written on the PCB of newer kit versions).

Example code for reading a data frame in **Master Control Mode** on **TLE493D-A1B6**:

```
// please note that the value of status should be checked and properly handler
int32_t status;
TLx493D_data_frame_t frame;
int16_t Bx_LSB, By_LSB, Bz_LSB, sensor_temperature_LSB;
TLV493D_data_t sensor_state;

// power-cycle the sensor
// On the 2Go Kit, the sensor may not be powered by default
// (some 2Go kits do not support sensor power control)
_POWER_DISABLE();
_POWER_ENABLE();

// Initialize the sensor
// sensor_state - data structure used to store the sensor configuration.
//                "NULL" can be passed to use the internal data structure of the
//                library to store the sensor state but when working with several sensors,
//                "NULL" can be passed for only one of them.
// false        - ADDR (i.e. SDA) line will be HIGH at sensor power up. Use false otherwise.
// TLV493D_A1B6_ADDR_1E_9C - the address to be configured to the sensor. Since true was passed
//                           for the previous parameter, the greater address (9C) will be used for
//                           further I2C communications.
```

```
status = TLV493D_A1B6_init(&sensor_state, true, TLV493D_A1B6_ADDR_1E_9C);

// set operation mode to Master Control Mode
status =  TLV493D_A1B6_set_operation_mode(&sensor_state, TLx493D_OP_MODE_MCM
      );

// read a data frame
status = TLV493D_A1B6_read_frame(&sensor_state, &frame);

// Copy Magnetic Field Intensity and temperature values in LSB format
Bx_LSB = frame.x;
By_LSB = frame.y;
Bz_LSB = frame.z;
sensor_temperature_LSB = frame.temp;
```

Example code for reading a data frame in **Master Control Mode** on **TLE493D-A2B6/W2B6/TLI493D-W2BW**:

```
// please note that the value of status should be checked and properly handler
int32_t status;
TLx493D_data_frame_t frame;
int16_t Bx_LSB, By_LSB, Bz_LSB, sensor_temperature_LSB;
TLE493D_data_t sensor_state;


// power-cycle the sensor
// On the 2Go Kit, the sensor may not be powered by default
_POWER_DISABLE();
_POWER_ENABLE();

// Initialize the sensor
// sensor_state - data structure used to store the sensor configuration.
//                "NULL" can be passed to use the internal data structure of the
//                library to store the sensor state but when working with several sensors,
//                "NULL" can be passed for only one of them.
//                Please note that the structure type is different then in the TLV example!
// TLE493D_AW2B6_I2C_A0_ADDR - the fused address of the sensor. It can be A0, A1, A2 or A3.
status = TLE493D_AW2B6_init(&sensor_state, TLE493D_AW2B6_I2C_A0_ADDR);

// set operation mode to Master Control Mode
status = TLE493D_AW2B6_set_operation_mode(&sensor_state,
      TLx493D_OP_MODE_MCM);

// read a data frame
status = TLE493D_AW2B6_read_frame(&sensor_state, &frame);

// Copy Magnetic Field Intensity and temperature values in LSB format
Bx_LSB = frame.x;
By_LSB = frame.y;
Bz_LSB = frame.z;
sensor_temperature_LSB = frame.temp;
```

For TLV493D-A1B6 please see the following files:
TLV_A1B6.h
TLV_A1B6.c

For TLE493D-A2B6/-W2B6 please see the following files:
TLE_AW2B6.h
TLE_AW2B6.c

1.2.3 Drivers abstraction levels

The lowest abstraction level presented by the library is the driver level allowing basic read and write operations with reserved data correction for the TLV493D-A1B6. The implementation is stateless and allows reading and writing sensor registers.

Example code for reading a data frame in **Master Control Mode** on **TLE493D-A1B6**:

```
// please note that the value of status should be checked and properly handler
int32_t status;
TLV493D_data_t sensor_state;


// power-cycle the sensor
// On the 2Go Kit, the sensor may not be powered by default
_POWER_DISABLE();
_POWER_ENABLE();

// { Initialize the sensor }
// consider the line ADDR(i.e. SDA) as HIGH at sensor startup
// and user default I2C address for that particular case
sensor_state.IIC_addr = TLV493D_A1B6_I2C_DEFAULT_ADDR_HIGH;

// copy the state of ALL the read registers, used for auto-correction
// on write operations
status = TLV493D_A1B6_read_regs(sensor_state.IIC_addr,
                                &(sensor_state.regmap_read),
                                TLV493D_A1B6_READ_REGS_COUNT - 1
);

// { Configure Master Control Mode }
// clear IICAddr, INT, FAST, LOW flags
sensor_state.regmap_write.MOD1 &= ~(TLV493D_A1B6_MOD1_IICAddr_MSK
                                  | TLV493D_A1B6_MOD1_INT_MSK
                                  | TLV493D_A1B6_MOD1_FAST_MSK
                                  | TLV493D_A1B6_MOD1_LOW_MSK);

// set Master Control Mode configuration and 9C as I2C address
// (it ADDR were LOW at sensor power-up, the address would have been 1E)
sensor_state.regmap_write.MOD1 |= TLV493D_A1B6_MOD1_IICAddr_1E_9C
                                | TLV493D_A1B6_MOD1_INT_DISABLE
                                | TLV493D_A1B6_MOD1_FAST_ENABLE
                                | TLV493D_A1B6_MOD1_LOW_ENABLE;

// write the register to the sensor; use read registers for correction of
// reserved bits (so they do not need to be copied manually)
status = TLV493D_A1B6_write_regs(sensor_state.IIC_addr,
                                 &(sensor_state.regmap_write),
                                 &(sensor_state.regmap_read));

// { after changing the address, manually update it in the data structure }
sensor_state.IIC_addr = 0x9Cu;

// { read a data frame }
// note that here the information is located in sensor_state.regmap_read
// and it is NOT parsed. In order to extract the Bx, By, Bz and temperature
// values, please see the TLV493D_A1B6_read_frame function (found in TLV_A1B6.c).
status = TLV493D_A1B6_read_regs(sensor_state.IIC_addr,
                                &(sensor_state.regmap_read),
                                TLV493D_A1B6_Temp2_REG);
```

Example code for reading a data frame in **Master Control Mode** on **TLE493D-A1B6**:

```
// please note that the value of status should be checked and properly handler
int32_t status;
TLV493D_data_t sensor_state;


// power-cycle the sensor
// On the 2Go Kit, the sensor may not be powered by default
_POWER_DISABLE();
_POWER_ENABLE();

// { Initialize the sensor }
// set I2C address
sensor_state.IIC_addr = TLE493D_AW2B6_I2C_A0_ADDR;

// I2C address A0 (not hexadecimal value A0, but rated the address with the label A0)
// 1-byte read protocol, Collision Avoidance Enabled, Interrupt Disabled
// Mode Master Control Mode
sensor_state.regmap.MOD1 = TLE493D_AW2B6_MOD1_IICadr_A0
                         | TLE493D_AW2B6_MOD1_PR_1BYTE
                         | TLE493D_AW2B6_MOD1_CA_ENABLE
                         | TLE493D_AW2B6_MOD1_INT_DISABLE
                         | TLE493D_AW2B6_MOD1_MODE_MCM;

// compute the parity bit
sensor_state.regmap.MOD1 |= (_get_FP_bit(&sensor_state) << TLE493D_AW2B6_MOD1_FP_POS);

// write registers
```

```
status = TLE493D_AW2B6_write_reg(sensor_state.IIC_addr,
                                 TLE493D_AW2B6_MOD1_REG,
                                 sensor_state.regmap.MOD1
);

// write config reg, set trigger
sensor_state.regmap.Config = TLE493D_AW2B6_Config_TRIG_R6;
status = TLE493D_AW2B6_write_reg(sensor_state.IIC_addr,
                                TLE493D_AW2B6_Config_REG,
                                sensor_state.regmap.Config
);

// { read entire register map }
// note that here the information is located in sensor_state.regmap
// and it is NOT parsed. In order to extract the Bx, By, Bz and temperature
// values, please see the TLE493D_AW2B6_read_frame function (found in TLE_AW2B6.c).
status = TLE493D_AW2B6_read_regs(sensor_state.IIC_addr,
                                 &(sensor_state.regmap),
                                 (TLE493D_AW2B6_REGS_COUNT - 1)
);
```

For this level please see the following files:

For TLV493D-A1B6:
TLV_A1B6_driver.h
TLV_A1B6_defines.h
TLV_A1B6_driver.c

For TLE493D-A2B6/-W2B6 TLI493D-W2BW:
TLE_AW2B6_driver.h
TLE_AW2B6_driver.c
TLE_AW2B6_defines.h

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 TLE493D_data_t Struct Reference

Data structure containing information about the internal state of a sensor. Also used to identify a sensor on a bus.

`#include <TLE_AW2B6.h>`

Public Attributes

- TLE493D_address_t IIC_addr

  I2C address to be written on the bus.
- TLE493D_regmap_t regmap

  Last known state of the internal sensor registers.

4.1.1 Detailed Description

Data structure containing information about the internal state of a sensor. Also used to identify a sensor on a bus.

The documentation for this struct was generated from the following file:

- TLE_AW2B6.h

4.2 TLE493D_regmap_t Struct Reference

Internal registers of the TLE493D sensor family.

`#include <TLE_AW2B6_defines.h>`

Public Attributes

- uint8_t **Bx**
- uint8_t **By**
- uint8_t **Bz**
- uint8_t **Temp**
- uint8_t **Bx2**
- uint8_t **Temp2**
- uint8_t **Diag**
- uint8_t **XL**
- uint8_t **XH**
- uint8_t **YL**
- uint8_t **YH**
- uint8_t **ZL**
- uint8_t **ZH**
- uint8_t **WU**
- uint8_t **TMode**
- uint8_t **TPhase**
- uint8_t **Config**
- uint8_t **MOD1**
- uint8_t **Reserved**
- uint8_t **MOD2**
- uint8_t **Reserved2**
- uint8_t **Reserved3**
- uint8_t **Ver**

4.2.1 Detailed Description

Internal registers of the TLE493D sensor family.

The documentation for this struct was generated from the following file:

- TLE_AW2B6_defines.h

4.3 TLV493D_data_t Struct Reference

Data structure containing information about the internal state of a sensor. Also used to identify a sensor on a bus.

`#include <TLV_A1B6.h>`

Public Attributes

- uint8_t IIC_addr

  I2C address to be written on the bus for sensor addressing.
- uint8_t frame_count

  Last frame value from the sensor ADC used to detect a stuck ADC.
- TLV493D_address_t addr_type

  Type of I2C address (addr bit unspecified)
- bool ADDR_high

  Address bit, representing state of ADDR line at power up.
- TLV493D_regmap_read_t regmap_read

  Last known state of the Read registers.
- TLV493D_regmap_write_t regmap_write

  Last known state of the Write registers.

4.3.1 Detailed Description

Data structure containing information about the internal state of a sensor. Also used to identify a sensor on a bus.

The documentation for this struct was generated from the following file:

- TLV_A1B6.h

4.4 TLV493D_regmap_read_t Struct Reference

Data structure describing the TLV493D read registers.

```
#include <TLV_A1B6_driver.h>
```

Public Attributes

- uint8_t **Bx**
- uint8_t **By**
- uint8_t **Bz**
- uint8_t **Temp**
- uint8_t **Bx2**
- uint8_t **Bz2**
- uint8_t **Temp2**
- uint8_t **FactSet1**
- uint8_t **FactSet2**
- uint8_t **FactSet3**

4.4.1 Detailed Description

Data structure describing the TLV493D read registers.

The documentation for this struct was generated from the following file:

- TLV_A1B6_driver.h

4.5 TLV493D_regmap_write_t Struct Reference

Data structure describing the TLV493D write registers.

```
#include <TLV_A1B6_driver.h>
```

Public Attributes

- uint8_t **Res**
- uint8_t **MOD1**
- uint8_t **Res2**
- uint8_t **MOD2**

4.5.1 Detailed Description

Data structure describing the TLV493D write registers.

The documentation for this struct was generated from the following file:

- TLV_A1B6_driver.h

4.6 TLx493D_data_frame_t Struct Reference

Generic data frame, common to all supported hardware version.

`#include <TLx493D.h>`

Public Attributes

- int16_t x

  Magnetic field intensity raw value on the X axis.
- int16_t y

  Magnetic field intensity raw value on the Y axis.
- int16_t z

  Magnetic field intensity raw value on the Z axis.
- int16_t temp

  Raw Temperature value.

4.6.1 Detailed Description

Generic data frame, common to all supported hardware version.

The documentation for this struct was generated from the following file:

- TLx493D.h

Chapter 5

File Documentation

5.1 atomic.h File Reference

Atomic blocks.

```
#include <stdint.h>
#include <cmsis_gcc.h>
```

Variables

- uint32_t **__atomic_int_flag**

5.1.1 Detailed Description

Atomic blocks.

5.2 conf_i2c.h File Reference

Configuration file for the I2C driver.

```
#include <xmc_i2c.h>
#include <xmc_gpio.h>
```

Macros

- #define CONF_I2C_SDA_PIN P2_10

  Pin used as SDA.
- #define CONF_I2C_SCL_PIN P2_11

  Pin used as SCL.
- #define CONF_I2C_CH XMC_I2C0_CH1

  USIC Channel used for I2C.

5.2.1 Detailed Description

Configuration file for the I2C driver.

Configure the SDA and SCL pins and communication speed.

5.3 conf_interrupts.h File Reference

ERU Module Configuration.

```
#include <xmc_eru.h>
```

5.3.1 Detailed Description

ERU Module Configuration.

5.4 conf_uart.h File Reference

Configure UART pins and communication parameters.

```
#include <xmc_usic.h>
#include <xmc_uart.h>
#include <xmc_gpio.h>
```

Macros

- #define CONF_UART_CH XMC_UART0_CH0

  define USIC module and channel used for UART
- #define CONF_UART_RX_PIN P2_2

  Pin used for UART RX.
- #define CONF_UART_TX_PIN P2_1

  Pin used for UART TX.

5.4.1 Detailed Description

Configure UART pins and communication parameters.

5.5 debug.h File Reference

Debug and logging.

```
#include <stdint.h>
#include "../TLx493D/TLx493D.h"
```

Macros

- #define _ENABLE_LOGGING_ 0
  Enable/Disable (1/0) logging on UART.
- #define **dbg_log**(x) ;

5.5.1 Detailed Description

Debug and logging.

The functions in this file are used for debugging and logging over UART.

5.6 flash_storage.h File Reference

Storage to flash.

```
#include <stdint.h>
#include <stddef.h>
```

Macros

- #define **BASE_ADDR_SECTOR**(x) ((uint32_t∗)(0x10000000ull + 0x1000ull ∗ (x)))

Functions

- void FLASH_erase_stored_data (void)
  Erase one flash page at storage address.
- void FLASH_store (uint32_t ∗to_store, uint32_t size)
  Store data to flash.
- void FLASH_load (uint32_t ∗to_load, uint32_t size)
  Load data from flash.

5.6.1 Detailed Description

Storage to flash.

5.7 gpio.h File Reference

GPIO based functions for controlling the power supply and ADDR pin state of the sensor.

```
#include <stdbool.h>
```

Functions

- void GPIO_sensor_supply (bool on)

  Set the sensor supply voltage and the discrete I2C bus pull-up resistors voltage HIGH or LOW.
- void GPIO_set_addr_wait (bool high)

  Configure the ADDR pin logical level at startup.

5.7.1 Detailed Description

GPIO based functions for controlling the power supply and ADDR pin state of the sensor.

5.7.2 Function Documentation

5.7.2.1 GPIO_set_addr_wait()

```
void GPIO_set_addr_wait (
            bool high )
```

Configure the ADDR pin logical level at startup.

(Necessary for TLV493D-A1B6 sensor only!) Set the ADDR(SDA) I2C pin HIGH or LOW and wait for the sensor startup. Finally, set ADDR to HIGH. Must be called right after the sensor power-up.

5.8 i2c.h File Reference

I2C Driver for XMC1100 USIC Module.

```
#include <stdint.h>
```

Enumerations

- enum I2C_status_t { **I2C_STATUS_OK** = 0, **I2C_STATUS_TIMEOUT** = 1, **I2C_STATUS_BUSY** = 2, **I2C_STATUS_NACK** = 3 }

  Status values returned by the I2C read/write commands.

Functions

- void I2C_init (void)

  Initialize the I2C peripheral.
- void I2C_enable (void)

  Enable the I2C channel.
- void I2C_disable (void)

  Disable the I2C channel.
- void I2C_mode_normal (void)

  Connect SDA and SCL pins to the I2C peripheral.
- void I2C_mode_gpio_input (void)

  Switch I2C pins to GPIO mode Disconnect SDA and SCL pins from the I2C peripheral. Configure them as GPIO inputs (in order to avoid unintended input signals to the I2C peripheral that will cause it to freeze)
- void I2C_peripheral_recover (void) __attribute__((deprecated))

  Recovery procedure according to XMC1100 Errata Sheet.
- int32_t I2C_read (uint8_t addr, uint8_t *data, uint8_t count)

  *Read a number of bytes from an I2C device to a **uint8_t** array.*
- int32_t I2C_write (uint8_t addr, const uint8_t *data, uint8_t count)

  *Write a number of **uint8_t** bytes to an I2C device.*
- void I2C_wait_transmit (void)

  Wait for an I2C transmission to end.
- void I2C_write_recover (void)

  Write FF (the recover value) to the I2C Bus.
- void I2C_write_reset (void)

  Write 00 (the reset value) to the I2C Bus.

5.8.1 Detailed Description

I2C Driver for XMC1100 USIC Module.

5.8.2 Function Documentation

5.8.2.1 I2C_read()

```
int32_t I2C_read (
            uint8_t addr,
            uint8_t * data,
            uint8_t count )
```

Read a number of bytes from an I2C device to a **uint8_t** array.

Read **count** bytes from the device with the given I2C address to the data array

Parameters

addr	I2C device address.
data	Address where the data should be stored.
count	Number of bytes to be read from the device to the **data** array

5.8.2.2 I2C_wait_transmit()

```
void I2C_wait_transmit (
            void  )
```

Wait for an I2C transmission to end.

Blocks the program execution until the last I2C operation is completed.

5.8.2.3 I2C_write()

```
int32_t I2C_write (
            uint8_t addr,
            const uint8_t * data,
            uint8_t count )
```

Write a number of **uint8_t** bytes to an I2C device.

Read **count** bytes from the device with the given I2C address to the data array

Parameters

addr	I2C device address.
data	Address where the data should be stored.
count	Number of bytes to be read from the device to the **data** array

5.9 i2c_int.h File Reference

Interrupt-based I2C Driver for XMC1100 USIC Module.

```
#include <stdint.h>
#include <stdbool.h>
```

Enumerations

- enum I2C_INT_state_t { **I2C_INT_SUCCESS**, **I2C_INT_IN_PROGRESS**, **I2C_INT_ERR_PROTOCOL**, **I2C_INT_NACK** }

  Status values returned by the I2C read/write commands.

Functions

- void I2C_INT_init (void)

  Initialize the I2C interrupt-based peripheral.
- bool I2C_INT_write (uint8_t addr, const uint8_t *data, uint8_t count, I2C_INT_state_t *handle)

start asynchronous I2C write

- bool I2C_INT_read (uint8_t addr, uint8_t *data, uint8_t count, I2C_INT_state_t *handle)

  start asynchronous I2C read
- void I2C_INT_write_reset (void)

  Write Reset address (for TLx493D sensors) on I2C bus (blocking)
- void I2C_INT_write_recover (void)

  Write Recover address (for TLx493D sensors) on I2C bus (blocking)

5.9.1 Detailed Description

Interrupt-based I2C Driver for XMC1100 USIC Module.

5.10 interface.h File Reference

Generic Library interface to the peripheral drivers.

```
#include "src/xmc1100/interrupt/interrupts.h"
#include "src/xmc1100/gpio/gpio.h"
#include "src/xmc1100/uart/uart.h"
#include "src/xmc1100/i2c_int/i2c_int.h"
```

Macros

- #define _I2C_read I2C_INT_read_block

  *Function Header: (uint8_t addr, uint8_t *data, uint8_t count)*
- #define _I2C_write I2C_INT_write_block

  Function Header: (uint8_t addr, const uint8_t data, uint8_t count)*
- #define _I2C_recover() I2C_INT_write_recover()

  Function Header: (void)
- #define _I2C_reset() I2C_INT_write_reset()

  Function Header: (void)
- #define _SET_ADDR_AND_WAIT(high) GPIO_set_addr_wait(high)

  Function Header: (bool high)
- #define _POWER_ENABLE() GPIO_sensor_supply(true)

  Function Header: (void)
- #define _POWER_DISABLE() GPIO_sensor_supply(false)

  Function Header: (void)
- #define _LOG_STR UART_write

  *Function Header: (void *data, uint32_t count)*

5.10.1 Detailed Description

Generic Library interface to the peripheral drivers.

The purpose of this file is to connect microcontroller dependent functions to the generic TLx493D library which is microcontroller agnostic. The functions specified below are needed for the normal functioning of the sensor.

Note: ALL I2C functions should return a positive value of type int32_t (defined in stdint.h) indicating some communication error or Zero(0) indicating a successful communication. The positive return values themselves, aside from indicating an error, are meaningless to the library and thus can be arbitrarily chosen by the user as seen fit. Negative return value are reserved and used internally by the generic library, thus no function referred in this library should return a negative error as it may collide with the library reserved return value!

5.10.2 Macro Definition Documentation

5.10.2.1 _I2C_read

```
#define _I2C_read I2C_INT_read_block
```

Function Header: (uint8_t addr, uint8_t *data, uint8_t count)

I2C read command must have a header precisely of type: (uint8_t addr, uint8_t *data, uint8_t count):

Parameters

addr	The I2C address of the sensor
data	The array that the function will read to
count	The number of bytes the function will read

```
// ================== EXAMPLE ==================
// Read 10 bytes from the I2C device with address 0x63
// to the array data_ptr
// The error code will be written to error. On success
// it will be 0 (Zero).
uint8_t data_ptr[10];
error = _I2C_read(0x23, data_ptr, 10);
```

5.10.2.2 _I2C_recover

```
#define _I2C_recover( ) I2C_INT_write_recover()
```

Function Header: (void)

_I2C_recover should take no parameter. It will write the recover address (FF) on the I2C bus

```
// ============ EXAMPLE ==============
// Perform am I2C recovery command
_I2C_recover();
```

5.10.2.3 _I2C_reset

```
#define _I2C_reset( ) I2C_INT_write_reset()
```

Function Header: (void)

_I2C_reset should take no parameter. It will write the reset address (00) on the I2C bus

```
// ============ EXAMPLE ==============
// Perform am I2C reset command
_I2C_reset();
```

5.10.2.4 _I2C_write

```
#define _I2C_write I2C_INT_write_block
```

Function Header: (uint8_t addr, const uint8_t* data, uint8_t count)

I2C write command must have a header precisely of type: (uint8_t addr, const uint8_t* data, uint8_t count), where:

Parameters

addr	Is the I2C address to read from data is the;
data	is the array that the function will read to
count	is the number of bytes the function will read

```
// =============== EXAMPLE ================
uint8_t payload[] = {5, 1, 2, 3};
// write 4 bytes from payload to the I2C device with address 0x23
error = _I2C_write(0x23, payload, 4);
```

5.10.2.5 _LOG_STR

```
#define _LOG_STR UART_write
```

Function Header: (void *data, uint32_t count)

Offers a method to log a string. The header of the method should be of type (void *data, uint32_t count)

Parameters

data	An array of unit8_t to be written
count	The number of bytes to be written `// ============ EXAMPLE ============== // log the string 'Example' _LOG_STR("Example", sizeof("Example") - 1);`

5.10.2.6 _POWER_DISABLE

```
#define _POWER_DISABLE( ) GPIO_sensor_supply(false)
```

Function Header: (void)

Set the pin responsible with supplying the sensor voltage to LOW. The function will be called with no arguments.

```
// ============ EXAMPLE ==============
// Power Down the sensor
_POWER_DISABLE();
```

5.10.2.7 _POWER_ENABLE

```
#define _POWER_ENABLE( ) GPIO_sensor_supply(true)
```

Function Header: (void)

Set the pin responsible with supplying the sensor voltage to HIGH. The function will be called with no arguments.

```
// ============ EXAMPLE ==============
// Power Up the sensor
_POWER_ENABLE();
```

5.10.2.8 _SET_ADDR_AND_WAIT

```
#define _SET_ADDR_AND_WAIT(
              high ) GPIO_set_addr_wait(high)
```

Function Header: (bool high)

Parameters

high	A value of **true** will set the ADDR pin HIGH at sensor power up, and a value of **false** will set the ADDR pin to LOW at sensor power up.

Set the desired level on ADDR(SDA) pin and wait at least 200us. Header should be of type (bool high) where: _SET_ADDR_AND_WAIT(true) will set the ADDR(SDA) line to HIGH and then wait for at least 200us. _SET_ADDR_AND_WAIT(false) will set the ADDR(SDA) line to LOW and then wait for at least 200us. Finally, the SDA line should be set back to HIGH.

```
// ============ EXAMPLE ============
// set voltage on ADDR pin to low and wait for sensor startup
_SET_ADDR_AND_WAIT(false);
```

5.11 interrupts.h File Reference

ERU Module used for Interrupt Handling.

```
#include <stdbool.h>
#include <XMC1100.h>
```

Functions

- void INT_init_ext_interrupts (void)

  Enable falling edge interrupt on P2.11 and P0.0 (TLI support) Using ERU and CCU4 slice CC40 (0) And pull-up input on P0.0 Disable mentioned interrupts.
- bool INT_get_TLI_detected (void)

  Return value of flag showing TLI detected. Requires sensor be kept in Low Power/Fast Mode until a pulse is detected Requires CCU40_0 interrupt enabled.

5.11.1 Detailed Description

ERU Module used for Interrupt Handling.

5.12 main.c File Reference

Generic Library usage example entry point.

```
#include <stdbool.h>
#include "src/xmc1100/uart/uart.h"
#include "src/xmc1100/i2c_int/i2c_int.h"
#include "src/xmc1100/interrupt/interrupts.h"
#include "src/TLx493D/TLx493D.h"
```

Functions

- int main (void)
  main() - Application entry point

5.12.1 Detailed Description

Generic Library usage example entry point.

5.12.2 Function Documentation

5.12.2.1 main()

```
int main (
            void  )
```

main() - Application entry point

Details of function
This routine is the application entry point. It is invoked by the device startup code.

5.13 misc.h File Reference

Miscellaneous functions.

```
#include <stdint.h>
#include <stddef.h>
```

Functions

- void MISC_memcpy (uint8_t *dest, const uint8_t *src, size_t n)

  Copy a number of byted from source to destination.

5.13.1 Detailed Description

Miscellaneous functions.

5.13.2 Function Documentation

5.13.2.1 MISC_memcpy()

```
void MISC_memcpy (
            uint8_t * dest,
            const uint8_t * src,
            size_t n )
```

Copy a number of byted from source to destination.

Parameters

dest	Destination of the copy.
src	Source of the copy.
n	Number of bytes to be copied.

5.14 time.h File Reference

Timing functions.

```
#include <stdint.h>
```

Functions

- void wait (uint32_t time_us)

  Delay execution for time_us microseconds.

5.14.1 Detailed Description

Timing functions.

5.15 TLE_AW2B6.c File Reference

```
#include "../TLx493D.h"
#include "TLE_AW2B6.h"
#include "driver/TLE_AW2B6_defines.h"
#include "driver/TLE_AW2B6_driver.h"
#include "src/misc/misc.h"
#include "src/TLx493D/interface.h"
#include "src/debug/debug.h"
```

Functions

- uint8_t TLE493D_AW2B6_get_FP_bit (TLE493D_data_t *data)

  Compute the value of the FP bit using the internal register state of the sensor.
- uint8_t TLE493D_AW2B6_get_CP_bit (TLE493D_data_t *data)

  Compute the value of the CP bit using the internal register state of the sensor.
- int32_t **TLE493D_AW2B6_init** (TLE493D_data_t *data, TLE493D_address_t i2c_addr)
- void TLE493D_AW2B6_reset_all (void)

  Write the reset sequence on the I2C bus.
- int32_t TLE493D_AW2B6_set_operation_mode (TLE493D_data_t *data, TLV493D_op_mode_t mode)

  Set the operation mode of the sensor.
- int32_t TLE493D_AW2B6_read_frame (TLE493D_data_t *data, TLx493D_data_frame_t *frame)

  Read a data frame from the sensor. An ADC sampling must be completed before calling this method.
- int32_t TLE493D_AW2B6_WU_enable (TLE493D_data_t *data, uint16_t wu_xl, uint16_t wu_xh, uint16_t wu_yl, uint16_t wu_yh, uint16_t wu_zl, uint16_t wu_zh)

  Enable the Wake Up mode (available only on the -W2B6 hardware version) with the provided upper and lower limits.
- int32_t TLE493D_AW2B6_WU_disable (TLE493D_data_t *data)

  Disable the Wake Up mode.
- int32_t TLE493D_AW2B6_set_IIC_address (TLE493D_data_t *data, TLE493D_address_t i2c_addr)

  Set a new I2C address for the sensor.
- int32_t TLE493D_AW2B6_magnetic_tmp_comp (TLE493D_data_t *data, TLE493D_magnetic_comp_t sens)

  Set the magnetic temperature compensation mode.
- int32_t TLE493D_AW2B6_set_high_sensitivity (TLE493D_data_t *data, bool on)

  Double the measurement sensitivity(when on=true). This will decrease the ADC integration speed.
- int32_t TLE493D_AW2B6_set_angle_mode (TLE493D_data_t *data, bool on)

  Enable/Disable angle mode. In order to enable angle mode, the temperature measurement must be disabled.
- int32_t TLE493D_AW2B6_set_temp_measure (TLE493D_data_t *data, bool on)

  Enable/Disable temperature measurement.
- int32_t TLV493D_A1B6_set_lowpower_update_frequency (TLE493D_data_t *data, TLE493D_lp_update_freq_t freq)

  Set the update frequency while in LOW POWER Mode.
- int32_t TLV493D_A1B6_set_trigger_mode (TLE493D_data_t *data, TLE493D_Config_trigger_mode_t mode)

  Set trigger mode. Note that the TLE493D_AW2B6_Config_TRIG_R0 mode is momentarily not safe to use in this software implementation.
- TLV493D_sensor_type_t TLE493D_get_hw_version (TLE493D_data_t *data)

  Return hardware version of the TLE493D.
- void TLE493D_AW2B6_get_data (TLE493D_data_t *dest)

  Copy the data stored in the library to the dest structure.
- int32_t TLE493D_AW2B6_set_data (TLE493D_data_t *src)

  Copy the data from src to the library and the sensor.

5.16 TLE_AW2B6.h File Reference

TLE493D-A2B6/-W2B6 abstraction.

```
#include <stdint.h>
#include <stdbool.h>
#include "../TLx493D.h"
#include "driver/TLE_AW2B6_defines.h"
```

Classes

- struct TLE493D_data_t

  Data structure containing information about the internal state of a sensor. Also used to identify a sensor on a bus.

Functions

- int32_t TLE493D_AW2B6_init (TLE493D_data_t *data, uint8_t i2c_addr)

  Initialize the sensor having the specified I2C address by reading the internal registers and disabling periodic interrupt pulses.
- uint8_t TLE493D_AW2B6_get_FP_bit (TLE493D_data_t *data)

  Compute the value of the FP bit using the internal register state of the sensor.
- uint8_t TLE493D_AW2B6_get_CP_bit (TLE493D_data_t *data)

  Compute the value of the CP bit using the internal register state of the sensor.
- void TLE493D_AW2B6_reset_all (void)

  Write the reset sequence on the I2C bus.
- int32_t TLE493D_AW2B6_set_operation_mode (TLE493D_data_t *data, TLV493D_op_mode_t mode)

  Set the operation mode of the sensor.
- int32_t TLE493D_AW2B6_read_frame (TLE493D_data_t *data, TLx493D_data_frame_t *frame)

  Read a data frame from the sensor. An ADC sampling must be completed before calling this method.
- int32_t TLE493D_AW2B6_WU_enable (TLE493D_data_t *data, uint16_t wu_xl, uint16_t wu_xh, uint16_t wu_yl, uint16_t wu_yh, uint16_t wu_zl, uint16_t wu_zh)

  Enable the Wake Up mode (available only on the -W2B6 hardware version) with the provided upper and lower limits.
- int32_t TLE493D_AW2B6_WU_disable (TLE493D_data_t *data)

  Disable the Wake Up mode.
- int32_t TLE493D_AW2B6_set_IIC_address (TLE493D_data_t *data, TLE493D_address_t i2c_addr)

  Set a new I2C address for the sensor.
- int32_t TLE493D_AW2B6_magnetic_tmp_comp (TLE493D_data_t *data, TLE493D_magnetic_comp_t sens)

  Set the magnetic temperature compensation mode.
- int32_t TLE493D_AW2B6_set_high_sensitivity (TLE493D_data_t *data, bool on)

  Double the measurement sensitivity(when on=true). This will decrease the ADC integration speed.
- int32_t TLE493D_AW2B6_set_angle_mode (TLE493D_data_t *data, bool on)

  Enable/Disable angle mode. In order to enable angle mode, the temperature measurement must be disabled.
- int32_t TLE493D_AW2B6_set_temp_measure (TLE493D_data_t *data, bool on)

  Enable/Disable temperature measurement.
- int32_t TLV493D_A1B6_set_lowpower_update_frequency (TLE493D_data_t *data, TLE493D_lp_update_freq_t freq)

  Set the update frequency while in LOW POWER Mode.

- int32_t TLV493D_A1B6_set_trigger_mode (TLE493D_data_t *data, TLE493D_Config_trigger_mode_t mode)

  Set trigger mode. Note that the TLE493D_AW2B6_Config_TRIG_R0 mode is momentarily not safe to use in this software implementation.
- TLV493D_sensor_type_t TLE493D_get_hw_version (TLE493D_data_t *data)

  Return hardware version of the TLE493D.
- void TLE493D_AW2B6_get_data (TLE493D_data_t *dest)

  Copy the data stored in the library to the dest structure.
- int32_t TLE493D_AW2B6_set_data (TLE493D_data_t *src)

  Copy the data from src to the library and the sensor.

5.16.1 Detailed Description

TLE493D-A2B6/-W2B6 abstraction.

Abstracts the basic functions of the TLE493D-A1B6/-W2B6 and offers a way to store the internal state of the sensor registers.

5.16.2 Function Documentation

5.16.2.1 TLE493D_AW2B6_init()

```
int32_t TLE493D_AW2B6_init (
            TLE493D_data_t * data,
            uint8_t i2c_addr )
```

Initialize the sensor having the specified I2C address by reading the internal registers and disabling periodic interrupt pulses.

Parameters

data	Structure to copy the values of the internal registers to. If data is NULL, the internal library data structure will be used instead. This approach support only one sensor, and for a bus of several sensors, a different data structure should be used for each one of them.
i2c_addr	The initial address of the sensor. Sensors may have different fused default addresses.

5.17 TLE_AW2B6_defines.h File Reference

Define the registers addresses and the positions and masks of the variables from the registers.

Classes

- struct TLE493D_regmap_t

  Internal registers of the TLE493D sensor family.

Macros

- #define **TLE493D_AW2B6_REGS_COUNT** (0x16U + 1U)
- #define **TLE493D_AW2B6_Bx_REG** (0x00U)
- #define **TLE493D_AW2B6_By_REG** (0x01U)
- #define **TLE493D_AW2B6_Bz_REG** (0x02U)
- #define **TLE493D_AW2B6_Temp_REG** (0x03U)
- #define **TLE493D_AW2B6_Bx2_REG** (0x04U)
- #define **TLE493D_AW2B6_Temp2_REG** (0x05U)
- #define **TLE493D_AW2B6_Diag_REG** (0x06U)
- #define **TLE493D_AW2B6_Diag_P_MSK** (1U << 7)
- #define **TLE493D_AW2B6_Diag_P_POS** (7U)
- #define **TLE493D_AW2B6_Diag_FF_MSK** (1U << 6)
- #define **TLE493D_AW2B6_Diag_FF_POS** (6U)
- #define **TLE493D_AW2B6_Diag_CF_MSK** (1U << 5)
- #define **TLE493D_AW2B6_Diag_CF_POS** (5U)
- #define **TLE493D_AW2B6_Diag_T_MSK** (1U << 4)
- #define **TLE493D_AW2B6_Diag_T_POS** (4U)
- #define **TLE493D_AW2B6_Diag_PD3_MSK** (1U << 3)
- #define **TLE493D_AW2B6_Diag_PD3_POS** (3U)
- #define **TLE493D_AW2B6_Diag_PD0_MSK** (1U << 2)
- #define **TLE493D_AW2B6_Diag_PD0_POS** (2U)
- #define **TLE493D_AW2B6_Diag_FRM_MSK** (3U << 0)
- #define **TLE493D_AW2B6_Diag_FRM_POS** (0U)
- #define **TLE493D_AW2B6_XL_REG** (0x07U)
- #define **TLE493D_AW2B6_XH_REG** (0x08U)
- #define **TLE493D_AW2B6_YL_REG** (0x09U)
- #define **TLE493D_AW2B6_YH_REG** (0x0AU)
- #define **TLE493D_AW2B6_ZL_REG** (0x0BU)
- #define **TLE493D_AW2B6_ZH_REG** (0x0CU)
- #define **TLE493D_AW2B6_WU_REG** (0x0DU)
- #define **TLE493D_AW2B6_WU_WA_POS** (0x7U)
- #define **TLE493D_AW2B6_WU_WA_MSK** (0x1U << 7)
- #define **TLE493D_AW2B6_WU_WU_POS** (0x6U)
- #define **TLE493D_AW2B6_WU_WU_ENABLE** (0x1U << 6)
- #define **TLE493D_AW2B6_WU_WU_DISABLE** (0x0U << 6)
- #define **TLE493D_AW2B6_WU_WU_MSK** (0x1U << 6)
- #define **TLE493D_AW2B6_WU_XH_POS** (0x3U)
- #define **TLE493D_AW2B6_WU_XH_MSK** (0x7U << 3)
- #define **TLE493D_AW2B6_WU_XL_POS** (0x3U)
- #define **TLE493D_AW2B6_WU_XL_MSK** (0x7U << 0)
- #define **TLE493D_AW2B6_TMode_REG** (0x0EU)
- #define **TLE493D_AW2B6_TMode_TST_POS** (6U)
- #define **TLE493D_AW2B6_TMode_TST_MSK** (3U << 6)
- #define **TLE493D_AW2B6_TMode_TST_NORMAL** (0U << 6)
- #define **TLE493D_AW2B6_TMode_TST_Vhall** (1U << 6)
- #define **TLE493D_AW2B6_TMode_TST_Spintest** (2U << 6)
- #define **TLE493D_AW2B6_TMode_TST_SAT** (3U << 6)
- #define **TLE493D_AW2B6_TMode_YH_POS** (3U)
- #define **TLE493D_AW2B6_TMode_YH_MSK** (7U << 3)
- #define **TLE493D_AW2B6_TMode_YL_POS** (0U)
- #define **TLE493D_AW2B6_TMode_YL_MSK** (7U << 0)
- #define **TLE493D_AW2B6_TPhase_REG** (0x0FU)
- #define **TLE493D_AW2B6_TPhase_PH_POS** (6U)
- #define **TLE493D_AW2B6_TPhase_PH_MSK** (3U << 6)

- #define **TLE493D_AW2B6_TPhase_ZH_POS** (3U)
- #define **TLE493D_AW2B6_TPhase_ZH_MSK** (7U << 3)
- #define **TLE493D_AW2B6_TPhase_ZL_POS** (0U)
- #define **TLE493D_AW2B6_TPhase_ZL_MSK** (7U << 0)
- #define **TLE493D_AW2B6_Config_REG** (0x10U)
- #define **TLE493D_AW2B6_Config_DT_POS** (0x7U)
- #define **TLE493D_AW2B6_Config_DT_MSK** (0x1U << 7)
- #define **TLE493D_AW2B6_Config_DT_ENABLE** (0x0U << 7)
- #define **TLE493D_AW2B6_Config_DT_DISABLE** (0x1U << 7)
- #define **TLE493D_AW2B6_Config_AM_POS** (0x6U)
- #define **TLE493D_AW2B6_Config_AM_MSK** (0x1U << 6)
- #define **TLE493D_AW2B6_Config_AM_ENABLE_BZ_MEASURE** (0x0U << 6)
- #define **TLE493D_AW2B6_Config_AM_DISABLE_BZ_MEASURE** (0x1U << 6)
- #define **TLE493D_AW2B6_Config_TRIG_POS** (0x4U)
- #define **TLE493D_AW2B6_Config_TRIG_MSK** (0x30U)
- #define **TLE493D_AW2B6_Config_X2_POS** (0x3U)
- #define **TLE493D_AW2B6_Config_X2_MSK** (1U << 3)
- #define **TLE493D_AW2B6_Config_X2_DOUBLE** (1U << 3)
- #define **TLE493D_AW2B6_Config_X2_SIMPLE** (0U << 3)
- #define **TLE493D_AW2B6_Config_TL_mag_POS** (0x1U)
- #define **TLE493D_AW2B6_Config_TL_mag_MSK** (3U << 1)
- #define **TLE493D_AW2B6_Config_CP_POS** (0x0U)
- #define **TLE493D_AW2B6_Config_CP_MSK** (0x1U)
- #define **TLE493D_AW2B6_MOD1_REG** (0x11U)
- #define **TLE493D_AW2B6_MOD1_FP_POS** (0x7U)
- #define **TLE493D_AW2B6_MOD1_FP_MSK** (1 << 0x7U)
- #define **TLE493D_AW2B6_MOD1_IICadr_POS** (0x5U)
- #define **TLE493D_AW2B6_MOD1_IICadr_MSK** (0x3U << 0x5U)
- #define **TLE493D_AW2B6_MOD1_IICadr_A0** (0x0U << 0x5U)
- #define **TLE493D_AW2B6_MOD1_IICadr_A1** (0x1U << 0x5U)
- #define **TLE493D_AW2B6_MOD1_IICadr_A2** (0x2U << 0x5U)
- #define **TLE493D_AW2B6_MOD1_IICadr_A3** (0x3U << 0x5U)
- #define **TLE493D_AW2B6_MOD1_PR_POS** (0x4U)
- #define **TLE493D_AW2B6_MOD1_PR_MSK** (0x1U << 0x4U)
- #define **TLE493D_AW2B6_MOD1_PR_2BYTE** (0x0U << 0x4U)
- #define **TLE493D_AW2B6_MOD1_PR_1BYTE** (0x1U << 0x4U)
- #define **TLE493D_AW2B6_MOD1_CA_POS** (0x3U)
- #define **TLE493D_AW2B6_MOD1_CA_MSK** (1 << 0x3U)
- #define **TLE493D_AW2B6_MOD1_CA_ENABLE** (0U << 0x3U)
- #define **TLE493D_AW2B6_MOD1_CA_DISABLE** (1U << 0x3U)
- #define **TLE493D_AW2B6_MOD1_INT_POS** (0x2U)
- #define **TLE493D_AW2B6_MOD1_INT_MSK** (0x1U << 0x2U)
- #define **TLE493D_AW2B6_MOD1_INT_ENABLE** (0x0U << 0x2U)
- #define **TLE493D_AW2B6_MOD1_INT_DISABLE** (0x1U << 0x2U)
- #define **TLE493D_AW2B6_MOD1_MODE_POS** (0x0U)
- #define **TLE493D_AW2B6_MOD1_MODE_MSK** (0x3U)
- #define **TLE493D_AW2B6_MOD1_MODE_LOW_POWER** (0U)
- #define **TLE493D_AW2B6_MOD1_MODE_MCM** (0x1U)
- #define **TLE493D_AW2B6_MOD1_MODE_FAST_MODE** (0x3U)
- #define **TLE493D_AW2B6_Reserved_REG** (0x12U)
- #define **TLE493D_AW2B6_MOD2_REG** (0x13U)
- #define **TLE493D_AW2B6_MOD2_PRD_POS** (0x5U)
- #define **TLE493D_AW2B6_MOD2_PRD_MSK** (0x7U << 5)
- #define **TLE493D_AW2B6_Reserved2_REG** (0x14U)
- #define **TLE493D_AW2B6_Reserved3_REG** (0x15U)

- #define **TLE493D_AW2B6_Ver_REG** (0x16U)
- #define **TLE493D_AW2B6_Ver_HWV_POS** (0x00U)
- #define **TLE493D_AW2B6_Ver_HWV_MSK** (0x0FU)
- #define **TLE493D_AW2B6_Ver_HWV_B21** (0x09U)
- #define **TLE493D_AW2B6_Ver_TYPE_POS** (0x4U)
- #define **TLE493D_AW2B6_Ver_TYPE_MSK** (0x3U << 4)

Enumerations

- enum TLE493D_Config_trigger_mode_t { **TLE493D_AW2B6_Config_TRIG_NONE** = (0x0U << 4), **TLE493D_AW2B6_Config_TRIG_R0** = (0x1U << 4), **TLE493D_AW2B6_Config_TRIG_R6** = (0x2U << 4) }

  Register-configurable trigger modes.
- enum TLE493D_magnetic_comp_t { **TLE493D_AW2B6_Config_TL_mag_TC0** = (0U << 1), **TLE493D_AW2B6_Config_TL_mag_TC1** = (1U << 1), **TLE493D_AW2B6_Config_TL_mag_TC2** = (2U << 1), **TLE493D_AW2B6_Config_TL_mag_TC3** = (3U << 1) }

  Sensitivity for magnetic compensation.
- enum TLE493D_i2c_trigger_mode_t { **TLE493D_AW2B6_I2C_NOTRIG** = (0x00U << 5), **TLE493D_AW2B6_I2C_TRIG_AFTER_WRITE** = (0x01U << 5), **TLE493D_AW2B6_I2C_TRIG_BEFORE_READ** = (0x02U << 5), **TLE493D_AW2B6_I2C_TRIG_AFTER_READ_R06** = (0x04U << 5) }

  Trigger bits for I2C Write commands.
- enum TLE493D_lp_update_freq_t { **TLE493D_AW2B6_MOD2_PRD_770** = (0x0U << 5), **TLE493D_AW2B6_MOD2_PRD_97** = (0x1U << 5), **TLE493D_AW2B6_MOD2_PRD_24** = (0x2U << 5), **TLE493D_AW2B6_MOD2_PRD_12** = (0x3U << 5), **TLE493D_AW2B6_MOD2_PRD_6** = (0x4U << 5), **TLE493D_AW2B6_MOD2_PRD_3** = (0x5U << 5), **TLE493D_AW2B6_MOD2_PRD_04** = (0x6U << 5), **TLE493D_AW2B6_MOD2_PRD_005** = (0x7U << 5) }

  Low power mode update frequencies.
- enum TLE493D_address_t { **TLE493D_AW2B6_I2C_A0_ADDR** = 0x6AU, **TLE493D_AW2B6_I2C_A1_ADDR** = 0x44U, **TLE493D_AW2B6_I2C_A2_ADDR** = 0xF0U, **TLE493D_AW2B6_I2C_A3_ADDR** = 0x88U }

  Sensor bus addresses.

5.17.1 Detailed Description

Define the registers addresses and the positions and masks of the variables from the registers.

Defines:
∗_REG register positions
∗_POS Position of value in register (starting from MSB)
∗_MSK Mask for a value in register

5.18 TLE_AW2B6_driver.c File Reference

```
#include "src/TLx493D/interface.h"
#include "TLE_AW2B6_defines.h"
#include "TLE_AW2B6_driver.h"
#include "../../TLx493D.h"
#include "src/misc/misc.h"
```

Macros

- #define **NULL** ((void∗)0)

Functions

- int32_t TLE493D_AW2B6_read_regs (uint8_t i2c_addr, TLE493D_regmap_t ∗regmap, uint8_t upto)

  *Read register values from the sensor, starting with the register at address 0 up to register **upto***
- int32_t TLE493D_AW2B6_write_reg (uint8_t i2c_addr, uint8_t reg_addr, uint8_t data)

  *Write the **data** value to the **reg_addr** register on the sensor with the I2C address **i2c_addr**.*
- int32_t TLE493D_AW2B6_write_reg_multi (uint8_t i2c_addr, uint8_t reg_addr_start, uint8_t ∗data, uint8_t count)

  *Write **count** bytes from the **data** array to the sensor with the I2C address **addr**, starting with the register **addr_reg_start**.*

5.18.1 Function Documentation

5.18.1.1 TLE493D_AW2B6_read_regs()

```
int32_t TLE493D_AW2B6_read_regs (
            uint8_t addr,
            TLE493D_regmap_t * regmap,
            uint8_t upto )
```

Read register values from the sensor, starting with the register at address 0 up to register **upto**

Parameters

addr	the I2C address of the sensor;
regmap	Register map structure used to store the read registers of the sensor.
upto	The reading process will start with register 0 and will continue incrementally up to the register **upto**.

5.18.1.2 TLE493D_AW2B6_write_reg()

```
int32_t TLE493D_AW2B6_write_reg (
            uint8_t i2c_addr,
            uint8_t reg_addr,
            uint8_t data )
```

Write the **data** value to the **reg_addr** register on the sensor with the I2C address **i2c_addr**.

Parameters

i2c_addr	I2C address of the sensor.
reg_addr	Address of the register that is to be written.
data	Data to be written to the register.

5.18.1.3 TLE493D_AW2B6_write_reg_multi()

```
int32_t TLE493D_AW2B6_write_reg_multi (
            uint8_t addr,
            uint8_t addr_reg_start,
            uint8_t * data,
            uint8_t count )
```

Write **count** bytes from the **data** array to the sensor with the I2C address **addr**, starting with the register **addr_↩ reg_start**.

Parameters

addr	I2C sensor address
addr_reg_start	Address of the first register to be written
data	Data to be written to the registers
count	Number of bytes to be written

5.19 TLE_AW2B6_driver.h File Reference

```
#include "TLE_AW2B6_defines.h"
```

Functions

- int32_t TLE493D_AW2B6_read_regs (uint8_t addr, TLE493D_regmap_t *regmap, uint8_t upto)

  *Read register values from the sensor, starting with the register at address 0 up to register **upto***
- int32_t TLE493D_AW2B6_write_reg (uint8_t i2c_addr, uint8_t reg_addr, uint8_t data)

  *Write the **data** value to the **reg_addr** register on the sensor with the I2C address **i2c_addr**.*
- int32_t TLE493D_AW2B6_write_reg_multi (uint8_t addr, uint8_t addr_reg_start, uint8_t *data, uint8_t count)

  *Write **count** bytes from the **data** array to the sensor with the I2C address **addr**, starting with the register **addr_reg↩ _start**.*

5.19.1 Detailed Description

Warning

IMPORTANT: The TLE493D driver assumes that the 1-Byte read mode is always activated before any read operation. The 2-Byte read mode is NOT supported!

5.19.2 Function Documentation

5.19.2.1 TLE493D_AW2B6_read_regs()

```
int32_t TLE493D_AW2B6_read_regs (
            uint8_t addr,
            TLE493D_regmap_t * regmap,
            uint8_t upto )
```

Read register values from the sensor, starting with the register at address 0 up to register **upto**

Parameters

addr	the I2C address of the sensor;
regmap	Register map structure used to store the read registers of the sensor.
upto	The reading process will start with register 0 and will continue incrementally up to the register **upto**.

5.19.2.2 TLE493D_AW2B6_write_reg()

```
int32_t TLE493D_AW2B6_write_reg (
            uint8_t i2c_addr,
            uint8_t reg_addr,
            uint8_t data )
```

Write the **data** value to the **reg_addr** register on the sensor with the I2C address **i2c_addr**.

Parameters

i2c_addr	I2C address of the sensor.
reg_addr	Address of the register that is to be written.
data	Data to be written to the register.

5.19.2.3 TLE493D_AW2B6_write_reg_multi()

```
int32_t TLE493D_AW2B6_write_reg_multi (
            uint8_t addr,
            uint8_t addr_reg_start,
            uint8_t * data,
            uint8_t count )
```

Write **count** bytes from the **data** array to the sensor with the I2C address **addr**, starting with the register **addr_reg_start**.

Parameters

addr	I2C sensor address
addr_reg_start	Address of the first register to be written
data	Data to be written to the registers
count	Number of bytes to be written

5.20 TLV_A1B6.c File Reference

```
#include <stdint.h>
#include "driver/TLV_A1B6_defines.h"
#include "driver/TLV_A1B6_driver.h"
#include "TLV_A1B6.h"
#include "../TLx493D.h"
```

Macros

- #define **NULL** ((void∗) 0)

Functions

- int32_t TLV493D_A1B6_init (TLV493D_data_t ∗data, bool addr_high, TLV493D_address_t addr_type)

  Initialize the sensor.
- int32_t TLV493D_A1B6_set_operation_mode (TLV493D_data_t ∗data, TLV493D_op_mode_t mode)

  Change the operation mode of the sensor.
- void TLV493D_A1B6_hard_reset_reconfigure (TLV493D_data_t ∗data)

  Hard reset the sensor by executing a power cycle and reinitialize using the settings from the data structure. Will only set the address.
- int32_t TLV493D_A1B6_read_frame (TLV493D_data_t ∗data, TLx493D_data_frame_t ∗frame)

  Read the registers of the TLx493D sensor and create a data frame.
- int32_t TLV493D_A1B6_set_temp_measure (TLV493D_data_t ∗data, bool enabled)

  Enable or disable the temperature measurement.
- int32_t TLV493D_A1B6_set_parity_test (TLV493D_data_t ∗data, bool enabled)

  Enable or disable the parity test.
- int32_t TLV493D_A1B6_set_IIC_address (TLV493D_data_t ∗data, TLV493D_address_t new_addr_type)

  Set a new I2C address for the sensor, considering the ADDR pin level at startup.
- void TLV493D_A1B6_get_data (TLV493D_data_t ∗dest)

  Copy the data stored in the library to the dest structure.
- int32_t TLV493D_A1B6_set_data (TLV493D_data_t ∗src)

  Copy the data from src to the library.

5.20.1 Function Documentation

5.20.1.1 TLV493D_A1B6_hard_reset_reconfigure()

```
void TLV493D_A1B6_hard_reset_reconfigure (
            TLV493D_data_t * data )
```

Hard reset the sensor by executing a power cycle and reinitialize using the settings from the data structure. Will only set the address.

Parameters

data	Sensor data structure. By passing NULL, local data will be used.

5.20.1.2 TLV493D_A1B6_init()

```
int32_t TLV493D_A1B6_init (
              TLV493D_data_t * data,
              bool ADDR_high,
              TLV493D_address_t addr_type )
```

Initialize the sensor.

Parameters

data	parameter is optional (can be replaced with NULL) and specifies a data structure that should store the state of the sensor. If no data structure is sepcifiec, an internal data structure will be used. This parameter should be used in a bus configuration to easily identify sensors and also to manually inspect the internal state of the sensor.
ADDR_high	indicates the level of ADDR at the time the sensor was powered up. ADDR_high=true indicates that the sensor was powered up with ADDR=HIGH ADDR_high=false indicates that the sensor was powered up with ADDR=LOW
addr_type	indicates the desired address after initialization while keeping in mind the value of ADDR_high and the ADDR pin logic value at startup.

```
// ======================== Example ============================
// NULL -> store sensor information internally inside the library
//  This mode supports only one sensor at a time
// true -> At startup the ADDR pin was HIGH so the internal sensor
// address pin is set to 1
// TLV493D_A1B6_ADDR_1E_9C -> the desired sensor address is either
// 1E or 9C. Since the ADDR pin was HIGH at startup, the address will
// always be 9C
TLV493D_A1B6_init(NULL, true, TLV493D_A1B6_ADDR_1E_9C);
```

5.21 TLV_A1B6.h File Reference

TLV493D-A1B6 abstraction.

```
#include "../TLx493D.h"
#include "driver/TLV_A1B6_driver.h"
```

Classes

- struct TLV493D_data_t

  Data structure containing information about the internal state of a sensor. Also used to identify a sensor on a bus.

Enumerations

- enum TLV493D_address_t { **TLV493D_A1B6_ADDR_3E_BC**, **TLV493D_A1B6_ADDR_36_B4**, **TLV493D_A1B6_ADDR_1E_9C**, **TLV493D_A1B6_ADDR_16_94** }

  I2C addresses supported by the TLV493D-A1B6 sensor.

Functions

- int32_t TLV493D_A1B6_init (TLV493D_data_t *data, bool ADDR_high, TLV493D_address_t addr_type)

  Initialize the sensor.
- void TLV493D_A1B6_hard_reset_reconfigure (TLV493D_data_t *data)

  Hard reset the sensor by executing a power cycle and reinitialize using the settings from the data structure. Will only set the address.
- int32_t TLV493D_A1B6_read_frame (TLV493D_data_t *data, TLx493D_data_frame_t *frame)

  Read the registers of the TLx493D sensor and create a data frame.
- int32_t TLV493D_A1B6_set_operation_mode (TLV493D_data_t *data, TLV493D_op_mode_t mode)

  Change the operation mode of the sensor.
- int32_t TLV493D_A1B6_set_temp_measure (TLV493D_data_t *data, bool enabled)

  Enable or disable the temperature measurement.
- int32_t TLV493D_A1B6_set_parity_test (TLV493D_data_t *data, bool enabled)

  Enable or disable the parity test.
- int32_t TLV493D_A1B6_set_IIC_address (TLV493D_data_t *data, TLV493D_address_t new_addr_type)

  Set a new I2C address for the sensor, considering the ADDR pin level at startup.
- void TLV493D_A1B6_get_data (TLV493D_data_t *dest)

  Copy the data stored in the library to the dest structure.
- int32_t TLV493D_A1B6_set_data (TLV493D_data_t *src)

  Copy the data from src to the library.

5.21.1 Detailed Description

TLV493D-A1B6 abstraction.

Abstracts the basic functions of the TLV493D-A1B6 and offers a way to store the internal state of the sensor registers.

5.21.2 Enumeration Type Documentation

5.21.2.1 TLV493D_address_t

```
enum TLV493D_address_t
```

I2C addresses supported by the TLV493D-A1B6 sensor.

The left side addresses from the define names (3E, 36, 1E, 16) are relevant when the sensor is powered up with the ADDR pin LOW. The right side addresses can be used when the sensor is powered up with the ADDR pin HIGH. All values are in hexadecimal representation.

5.21.3 Function Documentation

5.21.3.1 TLV493D_A1B6_hard_reset_reconfigure()

```
void TLV493D_A1B6_hard_reset_reconfigure (
            TLV493D_data_t * data )
```

Hard reset the sensor by executing a power cycle and reinitialize using the settings from the data structure. Will only set the address.

Parameters

data	Sensor data structure. By passing NULL, local data will be used.

5.21.3.2 TLV493D_A1B6_init()

```
int32_t TLV493D_A1B6_init (
            TLV493D_data_t * data,
            bool ADDR_high,
            TLV493D_address_t addr_type )
```

Initialize the sensor.

Parameters

data	parameter is optional (can be replaced with NULL) and specifies a data structure that should store the state of the sensor. If no data structure is sepcifiec, an internal data structure will be used. This parameter should be used in a bus configuration to easily identify sensors and also to manually inspect the internal state of the sensor.
ADDR_high	indicates the level of ADDR at the time the sensor was powered up. ADDR_high=true indicates that the sensor was powered up with ADDR=HIGH ADDR_high=false indicates that the sensor was powered up with ADDR=LOW
addr_type	indicates the desired address after initialization while keeping in mind the value of ADDR_high and the ADDR pin logic value at startup.

```
// ======================== Example ============================
// NULL -> store sensor information internally inside the library
//  This mode supports only one sensor at a time
// true -> At startup the ADDR pin was HIGH so the internal sensor
// address pin is set to 1
// TLV493D_A1B6_ADDR_1E_9C -> the desired sensor address is either
// 1E or 9C. Since the ADDR pin was HIGH at startup, the address will
// always be 9C
TLV493D_A1B6_init(NULL, true, TLV493D_A1B6_ADDR_1E_9C);
```

5.22 TLV_A1B6_defines.h File Reference

Define the registers addresses and the positions and masks of the variables from the registers.

```
#include <stdint.h>
```

Macros

- #define **TLV493D_A1B6_I2C_RESET_ADDR** (0x00U)
- #define **TLV493D_A1B6_I2C_RECOV_ADDR** (0xFFU)
- #define **TLV493D_A1B6_I2C_DEFAULT_ADDR_HIGH** (0xBCU)
- #define **TLV493D_A1B6_I2C_DEFAULT_ADDR_LOW** (0x3EU)
- #define **TLV493D_A1B6_READ_REGS_COUNT** (0x0AU)
- #define **TLV493D_A1B6_Bx_REG** (0x0U)
- #define **TLV493D_A1B6_By_REG** (0x1U)
- #define **TLV493D_A1B6_Bz_REG** (0x2U)
- #define **TLV493D_A1B6_Temp_REG** (0x3U)
- #define **TLV493D_A1B6_Temp_Temp_POS** (0x4U)
- #define **TLV493D_A1B6_Temp_Temp_MSK** (0xFU << 4)
- #define **TLV493D_A1B6_Temp_FRM_POS** (0x2U)
- #define **TLV493D_A1B6_Temp_FRM_MSK** (0x3U << 2)
- #define **TLV493D_A1B6_Temp_CH_POS** (0x0U)
- #define **TLV493D_A1B6_Temp_CH_MSK** (0x3U)
- #define **TLV493D_A1B6_Bx2_REG** (0x4U)
- #define **TLV493D_A1B6_Bx2_Bx_POS** (0x4U)
- #define **TLV493D_A1B6_Bx2_Bx_MSK** (0xFU << 4)
- #define **TLV493D_A1B6_Bx2_By_POS** (0x0U)
- #define **TLV493D_A1B6_Bx2_By_MSK** (0xFU)
- #define **TLV493D_A1B6_Bz2_REG** (0x5U)
- #define **TLV493D_A1B6_Bz2_Reserved_POS** (0x7U)
- #define **TLV493D_A1B6_Bz2_Reserved_MSK** (0x1U << 7)
- #define **TLV493D_A1B6_Bz2_T_POS** (0x6U)
- #define **TLV493D_A1B6_Bz2_T_MSK** (0x1U << 6)
- #define **TLV493D_A1B6_Bz2_F_POS** (0x5U)
- #define **TLV493D_A1B6_Bz2_F_MSK** (0x1U << 5)
- #define **TLV493D_A1B6_Bz2_PD_POS** (0x4U)
- #define **TLV493D_A1B6_Bz2_PD_MSK** (0x1U << 4)
- #define **TLV493D_A1B6_Bz2_Bz_POS** (0x0U)
- #define **TLV493D_A1B6_Bz2_Bz_MSK** (0xFU)
- #define **TLV493D_A1B6_Temp2_REG** (0x6U)
- #define **TLV493D_A1B6_Temp2_Temp_POS** (0x0U)
- #define **TLV493D_A1B6_Temp2_Temp_MSK** (0xFF)
- #define **TLV493D_A1B6_FactSet1_REG** (0x7U)
- #define **TLV493D_A1B6_FaceSet1_Reserved_POS** (0x0U)
- #define **TLV493D_A1B6_FaceSet1_Reserved_MSK** (0xFF)
- #define **TLV493D_A1B6_FactSet2_REG** (0x8U)
- #define **TLV493D_A1B6_FaceSet2_Reserved_POS** (0x0U)
- #define **TLV493D_A1B6_FaceSet2_Reserved_MSK** (0xFF)
- #define **TLV493D_A1B6_FactSet3_REG** (0x9U)
- #define **TLV493D_A1B6_FaceSet3_Reserved_POS** (0x0U)
- #define **TLV493D_A1B6_FaceSet3_Reserved_MSK** (0xFF)
- #define **TLV493D_A1B6_WRITE_REGS_COUNT** (0x04U)
- #define **TLV493D_A1B6_Res_REG** (0x0U)
- #define **TLV493D_A1B6_Res_Reserved_POS** (0x0U)
- #define **TLV493D_A1B6_Res_Reserved_MSK** (0xFF)
- #define **TLV493D_A1B6_MOD1_REG** (0x1U)
- #define **TLV493D_A1B6_MOD1_P_POS** (0x7U)

- #define **TLV493D_A1B6_MOD1_P_MSK** (0x1U << 7)
- #define **TLV493D_A1B6_MOD1_IICAddr_POS** (0x5U)
- #define **TLV493D_A1B6_MOD1_IICAddr_MSK** (0x3U << 5)
- #define **TLV493D_A1B6_MOD1_IICAddr_16_94** (0x3U << 5)
- #define **TLV493D_A1B6_MOD1_IICAddr_1E_9C** (0x2U << 5)
- #define **TLV493D_A1B6_MOD1_IICAddr_36_B4** (0x1U << 5)
- #define **TLV493D_A1B6_MOD1_IICAddr_3E_BC** (0x0U << 5)
- #define **TLV493D_A1B6_MOD1_Reserved_POS** (0x3U)
- #define **TLV493D_A1B6_MOD1_Reserved_MSK** (0x3U << 3)
- #define **TLV493D_A1B6_MOD1_INT_POS** (0x2U)
- #define **TLV493D_A1B6_MOD1_INT_MSK** (0x1U << 2)
- #define **TLV493D_A1B6_MOD1_INT_ENABLE** (0x1U << 2)
- #define **TLV493D_A1B6_MOD1_INT_DISABLE** (0x0U << 2)
- #define **TLV493D_A1B6_MOD1_FAST_POS** (0x1U)
- #define **TLV493D_A1B6_MOD1_FAST_MSK** (0x1U << 1)
- #define **TLV493D_A1B6_MOD1_FAST_ENABLE** (0x1U << 1)
- #define **TLV493D_A1B6_MOD1_FAST_DISABLE** (0x0U << 1)
- #define **TLV493D_A1B6_MOD1_LOW_POS** (0x0U)
- #define **TLV493D_A1B6_MOD1_LOW_MSK** (0x1U)
- #define **TLV493D_A1B6_MOD1_LOW_ENABLE** (0x1U)
- #define **TLV493D_A1B6_MOD1_LOW_DISABLE** (0x0U)
- #define **TLV493D_A1B6_Res2_REG** (0x2U)
- #define **TLV493D_A1B6_Res2_Reserved_POS** (0x0U)
- #define **TLV493D_A1B6_Res2_Reserved_MSK** (0xFF)
- #define **TLV493D_A1B6_MOD2_REG** (0x3U)
- #define **TLV493D_A1B6_MOD2_T_POS** (0x7U)
- #define **TLV493D_A1B6_MOD2_T_MSK** (0x1U << 7)
- #define **TLV493D_A1B6_MOD2_T_DISABLE** (0x1U << 7)
- #define **TLV493D_A1B6_MOD2_T_ENABLE** (0x0U << 7)
- #define **TLV493D_A1B6_MOD2_LP_POS** (0x6U)
- #define **TLV493D_A1B6_MOD2_LP_MSK** (0x1U << 6)
- #define **TLV493D_A1B6_MOD2_LP_ULTRA_LOW_POWER** (0x0U << 6)
- #define **TLV493D_A1B6_MOD2_LP_LOW_POWER** (0x1U << 6)
- #define **TLV493D_A1B6_MOD2_PT_POS** (0x5U)
- #define **TLV493D_A1B6_MOD2_PT_MSK** (0x1U << 5)
- #define **TLV493D_A1B6_MOD2_PT_DISABLE** (0x0U << 5)
- #define **TLV493D_A1B6_MOD2_PT_ENABLE** (0x1U << 5)
- #define **TLV493D_A1B6_MOD2_Reserved_POS** (0x0U)
- #define **TLV493D_A1B6_MOD2_Reserved_MSK** (0x1FU)

5.22.1 Detailed Description

Define the registers addresses and the positions and masks of the variables from the registers.

Defines:
∗_REG register positions
∗_POS Position of value in register (starting from MSB)
∗_MSK Mask for a value in register

5.23 TLV_A1B6_driver.c File Reference

```
#include "TLV_A1B6_driver.h"
#include "../../TLx493D.h"
```

Macros

- #define **NULL** ((void∗) 0)

Functions

- int32_t TLV493D_A1B6_read_regs (uint8_t addr, TLV493D_regmap_read_t ∗regmap, uint8_t upto)

  *Read register values from the sensor, starting with the register at address 0 up to register **upto**.*
- int32_t TLV493D_A1B6_write_regs (uint8_t addr, TLV493D_regmap_write_t ∗regmap, const TLV493D_regmap_read_t ∗regmap_check)

  Write the register data from regmap to the sensor registers.

5.23.1 Function Documentation

5.23.1.1 TLV493D_A1B6_read_regs()

```
int32_t TLV493D_A1B6_read_regs (
            uint8_t addr,
            TLV493D_regmap_read_t * regmap,
            uint8_t upto )
```

Read register values from the sensor, starting with the register at address 0 up to register **upto**.

Parameters

addr	the I2C address of the sensor;
regmap	register map read structure used to store the read registers of the sensor.
upto	The reading process will start with register 0 and will continue incrementally up to the register **upto**

Returns

Error code.

5.23.1.2 TLV493D_A1B6_write_regs()

```
int32_t TLV493D_A1B6_write_regs (
            uint8_t addr,
            TLV493D_regmap_write_t * regmap,
            const TLV493D_regmap_read_t * regmap_check )
```

Write the register data from regmap to the sensor registers.

If the **regmap_check** pointer points to a valid structure (is not NULL), the reserved registers data from the **regmap** will be overwritten with the reserved data from **regmap_check**. This ensures that the reserved data read from the sensor is properly written back to the sensor. This overwrite will need to happend only once, as the corrections will be stored inside **regmap**.

Returns

Error code.

5.24 TLV_A1B6_driver.h File Reference

Low level driver for the TLV493D-A1B6.

```
#include <stdint.h>
#include "TLV_A1B6_defines.h"
#include "src/TLx493D/interface.h"
```

Classes

- struct TLV493D_regmap_read_t

  Data structure describing the TLV493D read registers.
- struct TLV493D_regmap_write_t

  Data structure describing the TLV493D write registers.

Functions

- int32_t TLV493D_A1B6_read_regs (uint8_t addr, TLV493D_regmap_read_t ∗regmap, uint8_t upto)

  *Read register values from the sensor, starting with the register at address 0 up to register **upto**.*
- int32_t TLV493D_A1B6_write_regs (uint8_t addr, TLV493D_regmap_write_t ∗regmap, const TLV493D_regmap_read_t ∗regmap_check)

  Write the register data from regmap to the sensor registers.

5.24.1 Detailed Description

Low level driver for the TLV493D-A1B6.

It simplifies the read and write operations when working with the internal registers of the sensor and also help prevent changes to the reserved data registers.

5.24.2 Function Documentation

5.24.2.1 TLV493D_A1B6_read_regs()

```
int32_t TLV493D_A1B6_read_regs (
            uint8_t addr,
            TLV493D_regmap_read_t * regmap,
            uint8_t upto )
```

Read register values from the sensor, starting with the register at address 0 up to register **upto**.

Parameters

addr	the I2C address of the sensor;
regmap	register map read structure used to store the read registers of the sensor.
upto	The reading process will start with register 0 and will continue incrementally up to the register **upto**

Returns

Error code.

5.24.2.2 TLV493D_A1B6_write_regs()

```
int32_t TLV493D_A1B6_write_regs (
            uint8_t addr,
            TLV493D_regmap_write_t * regmap,
            const TLV493D_regmap_read_t * regmap_check )
```

Write the register data from regmap to the sensor registers.

If the **regmap_check** pointer points to a valid structure (is not NULL), the reserved registers data from the **regmap** will be overwritten with the reserved data from **regmap_check**. This ensures that the reserved data read from the sensor is properly written back to the sensor. This overwrite will need to happend only once, as the corrections will be stored inside **regmap**.

Returns

Error code.

5.25 TLx493D.c File Reference

```
#include "TLx493D.h"
#include <stdbool.h>
#include "src/xmc1100/interrupt/interrupts.h"
#include "src/misc/misc.h"
#include "src/xmc1100/uart/uart.h"
#include "TLV_A1B6/TLV_A1B6.h"
#include "TLE_AW2B6/TLE_AW2B6.h"
#include "TLE_AW2B6/driver/TLE_AW2B6_defines.h"
#include "src/debug/debug.h"
#include "src/xmc1100/time/time.h"
```

Functions

- int32_t TLx493D_init (void)

  Detect and initialize the connected sensor.
- TLV493D_sensor_type_t TLx493D_get_sensor_type (void)

  Return the type of sensor present on the board.
- int32_t TLx493D_set_operation_mode (TLV493D_op_mode_t mode)

  Set the operation mode of the sensors, if supported.
- TLV493D_op_mode_t TLx493D_get_operation_mode ()

  Get the operation mode of the sensors.
- int32_t TLx493D_read_frame (TLx493D_data_frame_t *frame)

  Read a data frame from the sensor.
- uint8_t MISC_get_parity (uint8_t data)

  Compute the EVEN parity of a byte of data.

5.25.1 Function Documentation

5.25.1.1 MISC_get_parity()

```
uint8_t MISC_get_parity (
            uint8_t data )
```

Compute the EVEN parity of a byte of data.

Returns

Even parity of the data, either the value 0 or 1.

5.25.1.2 TLx493D_init()

```
int32_t TLx493D_init (
            void  )
```

Detect and initialize the connected sensor.

Automatically detect the sensor hardware version and call the appropriate initialization sequences. Must be called prior to any other call to a TLx493D_* method.

Returns

Error code.

5.25.1.3 TLx493D_read_frame()

```
int32_t TLx493D_read_frame (
            TLx493D_data_frame_t * frame )
```

Read a data frame from the sensor.

Returns

Error code.

5.25.1.4 TLx493D_set_operation_mode()

```
int32_t TLx493D_set_operation_mode (
            TLV493D_op_mode_t mode )
```

Set the operation mode of the sensors, if supported.

Returns

Error code.

5.26 TLx493D.h File Reference

TLx 3D Hall Sensor Family Abstraction.

```
#include <stdbool.h>
#include <stdint.h>
```

Classes

- struct TLx493D_data_frame_t

  Generic data frame, common to all supported hardware version.

Enumerations

- enum {
  TLx493D_OK = 0, TLx493D_INVALID_ARGUMENT = -1, TLx493D_INVALID_FRAME = -2, TLx493D_NOT_IMPLEMENETED = -3,
  TLx493D_INVALID_SENSOR_STATE = -4, TLx493D_WU_ENABLE_FAIL = -5 }

  Error codes returned by the TLx493D library.
- enum TLV493D_sensor_type_t {
  TLx493D_TYPE_UNKNOWN, **TLx493D_TYPE_TLV_A1B6**, **TLx493D_TYPE_TLE_A2B6**, **TLx493D_TYPE_TLE_W2B6**,
  TLx493D_TYPE_TLI_W2BW }

  Type of sensor on board.
- enum TLV493D_op_mode_t {
  TLx493D_OP_MODE_NOT_INITIALIZED, **TLx493D_OP_MODE_POWER_DOWN**, **TLx493D_OP_MODE_MCM**, **TLx493D_OP_MODE_FAST**,
  TLx493D_OP_MODE_LOW_POWER, **TLx493D_OP_MODE_ULTRA_LOW_POWER** }

  Operating Mode.

Functions

- int32_t TLx493D_init (void)

  Detect and initialize the connected sensor.
- TLV493D_sensor_type_t TLx493D_get_sensor_type (void)

  Return the type of sensor present on the board.
- int32_t TLx493D_set_operation_mode (TLV493D_op_mode_t mode)

  Set the operation mode of the sensors, if supported.
- TLV493D_op_mode_t TLx493D_get_operation_mode ()

  Get the operation mode of the sensors.
- int32_t TLx493D_read_frame (TLx493D_data_frame_t *frame)

  Read a data frame from the sensor.
- uint8_t MISC_get_parity (uint8_t data)

  Compute the EVEN parity of a byte of data.

5.26.1 Detailed Description

TLx 3D Hall Sensor Family Abstraction.

This file presents and abstraction for the sensors of the TLx493D family, offering basic functionality like changing the operation mode of the sensor, or reading a data frame.

5.26.2 Enumeration Type Documentation

5.26.2.1 anonymous enum

`anonymous enum`

Error codes returned by the TLx493D library.

Enumerator

TLx493D_OK	No error encountered.
TLx493D_INVALID_ARGUMENT	Function called with invalid argument.
TLx493D_INVALID_FRAME	The returned frame is invalid and should be discarded.
TLx493D_NOT_IMPLEMENETED	The called method has not been implemented yet.
TLx493D_INVALID_SENSOR_STATE	One or mode sensor registers are set incorrectly.
TLx493D_WU_ENABLE_FAIL	The WU feature failed to activate; unknown error.

5.26.3 Function Documentation

5.26.3.1 MISC_get_parity()

```
uint8_t MISC_get_parity (
            uint8_t data )
```

Compute the EVEN parity of a byte of data.

Returns

Even parity of the data, either the value 0 or 1.

5.26.3.2 TLx493D_init()

```
int32_t TLx493D_init (
            void  )
```

Detect and initialize the connected sensor.

Automatically detect the sensor hardware version and call the appropriate initialization sequences. Must be called prior to any other call to a TLx493D_∗ method.

Returns

Error code.

5.26.3.3 TLx493D_read_frame()

```
int32_t TLx493D_read_frame (
            TLx493D_data_frame_t * frame )
```

Read a data frame from the sensor.

Returns

Error code.

5.26.3.4 TLx493D_set_operation_mode()

```
int32_t TLx493D_set_operation_mode (
            TLV493D_op_mode_t mode )
```

Set the operation mode of the sensors, if supported.

Returns

Error code.

5.27 uart.h File Reference

UART Driver for XMC1100 USIC Module.

```
#include <stdint.h>
#include <stdbool.h>
```

Functions

- void UART_init (void)

  Initialize the UART peripheral.
- uint8_t UART_available (void)

  Return number of bytes available for reading.
- void UART_read (uint8_t *data, uint32_t count)

  Read count bytes to the data array. If not enough bytes available, will block until available.
- uint8_t UART_read_byte (void)

  Read last received data byte. Not blocking.
- void UART_clear_rx_fifo (void)

  Discard all data from the receive FIFO buffer.
- void UART_write (void *data, uint32_t count)

  Write count bytes from the data array to UART.

5.27.1 Detailed Description

UART Driver for XMC1100 USIC Module.

Index