Infineon Arduino Library Documentation

Author: Infineon Technologies AG
Date: August 28, 2018

Contents

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

2 Class Index

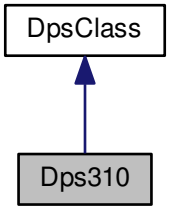
2.1 Class List

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

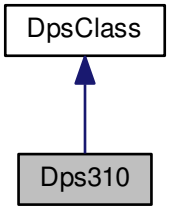
3 Class Documentation

3.1 Dps310 Class Reference

Inheritance diagram for Dps310:



Collaboration diagram for Dps310:



Public Member Functions

- int16_t **getContResults** (float ∗tempBuffer, uint8_t &tempCount, float ∗prsBuffer, uint8_t &prsCount)
- int16_t setInterruptSources (uint8_t intr_source, uint8_t polarity=1)

  Set the source of interrupt (FIFO full, measurement values ready)

Protected Member Functions

- void init (void)
- int16_t configTemp (uint8_t temp_mr, uint8_t temp_osr)
- int16_t configPressure (uint8_t prs_mr, uint8_t prs_osr)
- int16_t readcoeffs (void)
- int16_t **flushFIFO** ()
- float **calcTemp** (int32_t raw)
- float **calcPressure** (int32_t raw)

Protected Attributes

- uint8_t **m_tempSensor**
- int32_t **m_c0Half**
- int32_t **m_c1**

Additional Inherited Members

3.1.1 Member Function Documentation

3.1.1.1 int16_t Dps310::configPressure (uint8_t *prs_mr,* uint8_t *prs_osr*) [protected], [virtual]

Configures pressure measurement

Parameters

prs_mr	DPS__MEASUREMENT_RATE_1, DPS__MEASUREMENT_RATE_2,DPS__MEASUREMENT_RATE_4 ... DPS__MEASUREMENT_RATE_128
prs_osr	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128

Returns

0 normally or -1 on fail

Reimplemented from DpsClass.

3.1.1.2 int16_t Dps310::configTemp (uint8_t *temp_mr,* uint8_t *temp_osr*) [protected], [virtual]

Configures temperature measurement

Parameters

temp_mr	DPS__MEASUREMENT_RATE_1, DPS__MEASUREMENT_RATE_2,DPS__MEASUREMENT_RATE_4 ... DPS__MEASUREMENT_RATE_128
temp_osr	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128

Returns

0 normally or -1 on fail

Reimplemented from DpsClass.

3.1.1.3 void Dps310::init (void) [protected], [virtual]

Initializes the sensor. This function has to be called from begin() and requires a valid bus initialization.

Implements DpsClass.

3.1.1.4 int16_t Dps310::readcoeffs (void) [protected], [virtual]

reads the compensation coefficients from the sensor this is called once from init(), which is called from begin()

Returns

0 on success, -1 on fail

Implements DpsClass.

3.1.1.5 int16_t Dps310::setInterruptSources (uint8_t *intr_source*, uint8_t *polarity* = 1)

Set the source of interrupt (FIFO full, measurement values ready)

Parameters

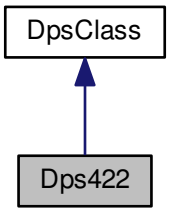
intr_source	Interrupt source as defined by Interrupt_source_310_e
polarity	

Returns

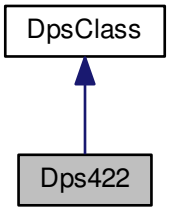
status code

3.2 Dps422 Class Reference

Inheritance diagram for Dps422:



Collaboration diagram for Dps422:



Public Member Functions

- int16_t **getContResults** (float ∗tempBuffer, uint8_t &tempCount, float ∗prsBuffer, uint8_t &prsCount)
- int16_t setInterruptSources (uint8_t intr_source, uint8_t polarity=1)

  Set the source of interrupt (FIFO full, measurement values ready)
- int16_t measureBothOnce (float &prs, float &temp)

  measures both temperature and pressure values, when op mode is set to CMD_BOTH
- int16_t **measureBothOnce** (float &prs, float &temp, uint8_t prs_osr, uint8_t temp_osr)

Protected Member Functions

- void init (void)
- int16_t readcoeffs (void)
- int16_t **flushFIFO** ()

- float **calcTemp** (int32_t raw)
- float **calcPressure** (int32_t raw)

Protected Attributes

- float **a_prime**
- float **b_prime**
- int32_t **m_c02**
- int32_t **m_c12**

Additional Inherited Members

3.2.1 Member Function Documentation

3.2.1.1 void Dps422::init (void) [protected], [virtual]

Initializes the sensor. This function has to be called from begin() and requires a valid bus initialization.

Implements DpsClass.

3.2.1.2 int16_t Dps422::measureBothOnce (float & *prs*, float & *temp*)

measures both temperature and pressure values, when op mode is set to CMD_BOTH

Parameters

prs	reference to the pressure value
temp	prs reference to the temperature value

Returns

status code

3.2.1.3 int16_t Dps422::readcoeffs (void) [protected], [virtual]

reads the compensation coefficients from the sensor this is called once from init(), which is called from begin()

Returns

0 on success, -1 on fail

Implements DpsClass.

3.2.1.4 int16_t Dps422::setInterruptSources (uint8_t *intr_source*, uint8_t *polarity* = 1)

Set the source of interrupt (FIFO full, measurement values ready)

Parameters

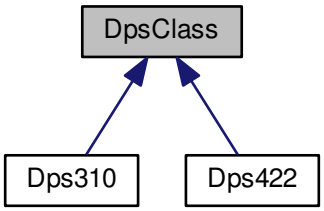
intr_source	Source of interrupt as defined by Interrupt_source_420_e
polarity	

Returns

int16_t

3.3 DpsClass Class Reference

Inheritance diagram for DpsClass:



Public Member Functions

- void begin (TwoWire &bus)
- void begin (TwoWire &bus, uint8_t slaveAddress)
- void begin (SPIClass &bus, int32_t chipSelect)
- void begin (SPIClass &bus, int32_t chipSelect, uint8_t threeWire)
- void end (void)
- uint8_t getProductId (void)
- uint8_t getRevisionId (void)
- int16_t standby (void)
- int16_t measureTempOnce (float &result)
- int16_t measureTempOnce (float &result, uint8_t oversamplingRate)
- int16_t startMeasureTempOnce (void)
- int16_t startMeasureTempOnce (uint8_t oversamplingRate)
- int16_t measurePressureOnce (float &result)
- int16_t measurePressureOnce (float &result, uint8_t oversamplingRate)
- int16_t startMeasurePressureOnce (void)
- int16_t startMeasurePressureOnce (uint8_t oversamplingRate)

- int16_t getSingleResult (float &result)
- int16_t startMeasureTempCont (uint8_t measureRate, uint8_t oversamplingRate)
- int16_t startMeasurePressureCont (uint8_t measureRate, uint8_t oversamplingRate)
- int16_t startMeasureBothCont (uint8_t tempMr, uint8_t tempOsr, uint8_t prsMr, uint8_t prsOsr)
- int16_t getIntStatusFifoFull (void)
- int16_t getIntStatusTempReady (void)
- int16_t getIntStatusPrsReady (void)
- int16_t correctTemp (void)

Protected Member Functions

- virtual void init (void)=0
- virtual int16_t readcoeffs (void)=0
- int16_t setOpMode (uint8_t opMode)
- virtual int16_t configTemp (uint8_t temp_mr, uint8_t temp_osr)
- virtual int16_t configPressure (uint8_t prs_mr, uint8_t prs_osr)
- virtual int16_t **flushFIFO** ()=0
- virtual float **calcTemp** (int32_t raw)=0
- virtual float **calcPressure** (int32_t raw)=0
- int16_t **enableFIFO** ()
- int16_t **disableFIFO** ()
- uint16_t calcBusyTime (uint16_t temp_rate, uint16_t temp_osr)
- int16_t getFIFOvalue (int32_t *value)
- int16_t getContResults (float *tempBuffer, uint8_t &tempCount, float *prsBuffer, uint8_t &prsCount, RegMask_t reg)
- int16_t readByte (uint8_t regAddress)
- int16_t readByteSPI (uint8_t regAddress)
- int16_t readBlock (RegBlock_t regBlock, uint8_t *buffer)
- int16_t readBlockSPI (RegBlock_t regBlock, uint8_t *readbuffer)
- int16_t writeByte (uint8_t regAddress, uint8_t data)
- int16_t writeByte (uint8_t regAddress, uint8_t data, uint8_t check)
- int16_t writeByteSpi (uint8_t regAddress, uint8_t data, uint8_t check)
- int16_t writeByteBitfield (uint8_t data, RegMask_t regMask)
- int16_t writeByteBitfield (uint8_t data, uint8_t regAddress, uint8_t mask, uint8_t shift, uint8_t check)
- int16_t readByteBitfield (RegMask_t regMask)
- void getTwosComplement (int32_t *raw, uint8_t length)

  converts non-32-bit negative numbers to 32-bit negative numbers with 2's complement
- int16_t getRawResult (int32_t *raw, RegBlock_t reg)

  Get a raw result from a given register block.

Protected Attributes

- dps::Mode **m_opMode**
- uint8_t **m_initFail**
- uint8_t **m_productID**
- uint8_t **m_revisionID**
- uint8_t **m_tempMr**
- uint8_t **m_tempOsr**
- uint8_t **m_prsMr**
- uint8_t **m_prsOsr**
- int32_t **m_c00**
- int32_t **m_c10**
- int32_t **m_c01**
- int32_t **m_c11**
- int32_t **m_c20**
- int32_t **m_c21**
- int32_t **m_c30**
- float **m_lastTempScal**
- uint8_t **m_SpiI2c**
- TwoWire * **m_i2cbus**
- uint8_t **m_slaveAddress**
- SPIClass * **m_spibus**
- int32_t **m_chipSelect**
- uint8_t **m_threeWire**

Static Protected Attributes

- static const int32_t **scaling_facts** [DPS__NUM_OF_SCAL_FACTS] = {524288, 1572864, 3670016, 7864320, 253952, 516096, 1040384, 2088960}

3.3.1 Detailed Description

Arduino library to control Dps310

"Dps310" represents Infineon's high-sensetive pressure and temperature sensor. It measures in ranges of 300 - 1200 hPa and -40 and 85 °C. The sensor can be connected via SPI or I2C. It is able to perform single measurements or to perform continuous measurements of temperature and pressure at the same time, and stores the results in a FIFO to reduce bus communication.

Have a look at the datasheet for more information.

3.3.2 Member Function Documentation

3.3.2.1 void DpsClass::begin (TwoWire & *bus*)

I2C begin function with standard address

3.3.2.2 void DpsClass::begin (TwoWire & *bus,* uint8_t *slaveAddress*)

Standard I2C begin function

Parameters

&bus	I2CBus which connects MC to the sensor
slaveAddress	I2C address of the sensor (0x77 or 0x76)

3.3.2.3 void DpsClass::begin (SPIClass & *bus,* int32_t *chipSelect*)

SPI begin function for Dps310 with 4-wire SPI

3.3.2.4 void DpsClass::begin (SPIClass & *bus,* int32_t *chipSelect,* uint8_t *threeWire*)

Standard SPI begin function

Parameters

&bus	SPI bus which connects MC to Dps310
chipSelect	Number of the CS line for the Dps310
threeWire	1 if Dps310 is connected with 3-wire SPI 0 if Dps310 is connected with 4-wire SPI (standard)

3.3.2.5 uint16_t DpsClass::calcBusyTime (uint16_t *temp_rate,* uint16_t *temp_osr*) [protected]

calculates the time that the sensor needs for 2^mr measurements with an oversampling rate of 2^osr (see table "pressure measurement time (ms) versus oversampling rate") Note that the total measurement time for temperature and pressure must not be more than 1 second. Timing behavior of pressure and temperature sensors can be considered the same.

Parameters

mr	DPS__MEASUREMENT_RATE_1, DPS__MEASUREMENT_RATE_2,DPS__MEASUREMENT_RATE_4 ... DPS__MEASUREMENT_RATE_128
osr	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128

Returns

time that the sensor needs for this measurement

3.3.2.6 int16_t DpsClass::configPressure (uint8_t *prs_mr,* uint8_t *prs_osr*) [protected], [virtual]

Configures pressure measurement

Parameters

prs_mr	DPS__MEASUREMENT_RATE_1, DPS__MEASUREMENT_RATE_2,DPS__MEASUREMENT_RATE_4 ... DPS__MEASUREMENT_RATE_128
prs_osr	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128

Returns

0 normally or -1 on fail

Reimplemented in Dps310.

3.3.2.7 int16_t DpsClass::configTemp (uint8_t *temp_mr*, uint8_t *temp_osr*) [protected], [virtual]

Configures temperature measurement

Parameters

temp_mr	DPS__MEASUREMENT_RATE_1, DPS__MEASUREMENT_RATE_2,DPS__MEASUREMENT_RATE_4 ... DPS__MEASUREMENT_RATE_128
temp_osr	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128

Returns

0 normally or -1 on fail

Reimplemented in Dps310.

3.3.2.8 int16_t DpsClass::correctTemp (void)

Function to fix a hardware problem on some devices You have this problem if you measure a temperature which is too high (e.g. 60°C when temperature is around 20°C) Call correctTemp() directly after begin() to fix this issue

3.3.2.9 void DpsClass::end (void)

End function for Dps310 Sets the sensor to idle mode

3.3.2.10 int16_t DpsClass::getContResults (float * *tempBuffer*, uint8_t & *tempCount*, float * *prsBuffer*, uint8_t & *prsCount*, RegMask_t *reg*) [protected]

Gets the results from continuous measurements and writes them to given arrays

Parameters

**tempBuffer*	The start address of the buffer where the temperature results are written If this is NULL, no temperature results will be written out

Parameters

&tempCount	The size of the buffer for temperature results. When the function ends, it will contain the number of bytes written to the buffer.
∗prsBuffer	The start address of the buffer where the pressure results are written If this is NULL, no pressure results will be written out
&prsCount	The size of the buffer for pressure results. When the function ends, it will contain the number of bytes written to the buffer.
reg	The FIFO empty register field; needed since this field is different for each sensor

Returns

status code

3.3.2.11 int16_t DpsClass::getFIFOvalue (int32_t ∗ *value*) [protected]

reads the next raw value from the FIFO

Parameters

value	the raw pressure or temperature value read from the pressure register blocks, where the LSB of PRS_B0 marks wheather the value is a temperatur or a pressure.

Returns

-1 on fail 0 if result is a temperature raw value 1 if result is a pressure raw value

3.3.2.12 int16_t DpsClass::getIntStatusFifoFull (void)

Gets the interrupt status flag of the FIFO

Returns

1 if the FIFO is full and caused an interrupt 0 if the FIFO is not full or FIFO interrupt is disabled -1 on fail

3.3.2.13 int16_t DpsClass::getIntStatusPrsReady (void)

Gets the interrupt status flag that indicates a finished pressure measurement

Returns

1 if a finished pressure measurement caused an interrupt; 0 if there is no finished pressure measurement or interrupts are disabled; -1 on fail.

3.3.2.14 int16_t DpsClass::getIntStatusTempReady (void)

Gets the interrupt status flag that indicates a finished temperature measurement

Returns

1 if a finished temperature measurement caused an interrupt; 0 if there is no finished temperature measurement or interrupts are disabled; -1 on fail.

3.3.2.15 uint8_t DpsClass::getProductId (void)

returns the Product ID of the connected Dps310 sensor

3.3.2.16 int16_t DpsClass::getRawResult (int32_t * *raw*, RegBlock_t *reg*) [protected]

Get a raw result from a given register block.

Parameters

raw	The address where the raw value is to be written
reg	The register block to be read from

Returns

status code

3.3.2.17 uint8_t DpsClass::getRevisionId (void)

returns the Revision ID of the connected Dps310 sensor

3.3.2.18 int16_t DpsClass::getSingleResult (float & *result*)

gets the result a single temperature or pressure measurement in °C or Pa

Parameters

&result	reference to a float value where the result will be written

Returns

status code

3.3.2.19 void DpsClass::getTwosComplement (int32_t * *raw*, uint8_t *length*) [protected]

converts non-32-bit negative numbers to 32-bit negative numbers with 2's complement

Parameters

raw	The raw number of less than 32 bits
length	The bit length

3.3.2.20 virtual void DpsClass::init (void) [protected], [pure virtual]

Initializes the sensor. This function has to be called from begin() and requires a valid bus initialization.

Implemented in Dps422, and Dps310.

3.3.2.21 int16_t DpsClass::measurePressureOnce (float & *result*)

performs one pressure measurement

Parameters

&result	reference to a float value where the result will be written

Returns

status code

3.3.2.22 int16_t DpsClass::measurePressureOnce (float & *result*, uint8_t *oversamplingRate*)

performs one pressure measurement with specified oversamplingRate

Parameters

&result	reference to a float where the result will be written
oversamplingRate	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128

Returns

status code

3.3.2.23 int16_t DpsClass::measureTempOnce (float & *result*)

performs one temperature measurement

Parameters

&result	reference to a float value where the result will be written

Returns

status code

3.3.2.24 int16_t DpsClass::measureTempOnce (float & *result*, uint8_t *oversamplingRate*)

performs one temperature measurement with specified oversamplingRate

Parameters

&result	reference to a float where the result will be written
oversamplingRate	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128, which are defined as integers 0 - 7 The number of measurements equals to 2^n, if the value written to the register field is n. 2^n internal measurements are combined to return a more exact measurement

Returns

status code

3.3.2.25 int16_t DpsClass::readBlock (RegBlock_t *regBlock*, uint8_t * *buffer*) [protected]

reads a block from the sensor

Parameters

regAdress	Address that has to be read
length	Length of data block
buffer	Buffer where data will be stored

Returns

number of bytes that have been read successfully, which might not always equal to length due to rx-Buffer overflow etc.

3.3.2.26 int16_t DpsClass::readBlockSPI (RegBlock_t *regBlock*, uint8_t * *readbuffer*) [protected]

reads a block from the sensor via SPI

Parameters

regAdress	Address that has to be read
length	Length of data block
readbuffer	Buffer where data will be stored

Returns

number of bytes that have been read successfully, which might not always equal to length due to rx-Buffer overflow etc.

3.3.2.27 int16_t DpsClass::readByte (uint8_t *regAddress*) [protected]

reads a byte from the sensor

Parameters

regAdress	Address that has to be read

Returns

register content or -1 on fail

3.3.2.28 int16_t DpsClass::readByteBitfield (RegMask_t *regMask*) [protected]

reads a bit field from the sensor regMask: Mask of the register that has to be updated data: BitValues that will be written to the register

Returns

read and processed bits or -1 on fail

3.3.2.29 int16_t DpsClass::readByteSPI (uint8_t *regAddress*) [protected]

reads a byte from the sensor via SPI this function is automatically called by readByte if the sensor is connected via SPI

Parameters

regAdress	Address that has to be read

Returns

register content or -1 on fail

3.3.2.30 virtual int16_t DpsClass::readcoeffs (void) [protected], [pure virtual]

reads the compensation coefficients from the sensor this is called once from init(), which is called from begin()

Returns

0 on success, -1 on fail

Implemented in Dps422, and Dps310.

3.3.2.31 int16_t DpsClass::setOpMode (uint8_t *opMode*) [protected]

Sets the Operation Mode of the sensor

Parameters

opMode	the new OpMode as defined by dps::Mode; CMD_BOTH should not be used for DPS310

Returns

0 on success, -1 on fail

3.3.2.32 int16_t DpsClass::standby (void)

Sets the Dps310 to standby mode

Returns

status code

3.3.2.33 int16_t DpsClass::startMeasureBothCont (uint8_t *tempMr,* uint8_t *tempOsr,* uint8_t *prsMr,* uint8_t *prsOsr*)

starts a continuous temperature and pressure measurement with specified measurement rate and oversampling rate for temperature and pressure measurement respectvely.

Parameters

tempMr	measure rate for temperature
tempOsr	oversampling rate for temperature
prsMr	measure rate for pressure
prsOsr	oversampling rate for pressure

Returns

status code

3.3.2.34 int16_t DpsClass::startMeasurePressureCont (uint8_t *measureRate,* uint8_t *oversamplingRate*)

starts a continuous temperature measurement with specified measurement rate and oversampling rate

Parameters

measureRate	DPS__MEASUREMENT_RATE_1, DPS__MEASUREMENT_RATE_2,DPS__MEASUREMENT_RATE_4 ... DPS__MEASUREMENT_RATE_128
oversamplingRate	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128

Returns

status code

3.3.2.35 int16_t DpsClass::startMeasurePressureOnce (void)

starts a single pressure measurement

Returns

status code

3.3.2.36 int16_t DpsClass::startMeasurePressureOnce (uint8_t *oversamplingRate*)

starts a single pressure measurement with specified oversamplingRate

Parameters

oversamplingRate	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128

Returns

status code

3.3.2.37 int16_t DpsClass::startMeasureTempCont (uint8_t *measureRate,* uint8_t *oversamplingRate*)

starts a continuous temperature measurement with specified measurement rate and oversampling rate If measure rate is n and oversampling rate is m, the DPS310 performs 2^(n+m) internal measurements per second. The DPS310 cannot operate with high precision and high speed at the same time. Consult the datasheet for more information.

Parameters

measureRate	DPS__MEASUREMENT_RATE_1, DPS__MEASUREMENT_RATE_2,DPS__MEASUREMENT_RATE_4 ... DPS__MEASUREMENT_RATE_128
oversamplingRate	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128

Returns

status code

3.3.2.38 int16_t DpsClass::startMeasureTempOnce (void)

starts a single temperature measurement

Returns

status code

3.3.2.39 int16_t DpsClass::startMeasureTempOnce (uint8_t *oversamplingRate*)

starts a single temperature measurement with specified oversamplingRate

Parameters

oversamplingRate	DPS__OVERSAMPLING_RATE_1, DPS__OVERSAMPLING_RATE_2, DPS__OVERSAMPLING_RATE_4 ... DPS__OVERSAMPLING_RATE_128, which are defined as integers 0 - 7

Returns

status code

3.3.2.40 int16_t DpsClass::writeByte (uint8_t *regAddress*, uint8_t *data*) [protected]

writes a byte to a given register of the sensor without checking

Parameters

regAdress	Address of the register that has to be updated
data	Byte that will be written to the register

Returns

0 if byte was written successfully or -1 on fail

3.3.2.41 int16_t DpsClass::writeByte (uint8_t *regAddress*, uint8_t *data*, uint8_t *check*) [protected]

writes a byte to a register of the sensor

Parameters

regAdress	Address of the register that has to be updated
data	Byte that will be written to the register
check	If this is true, register content will be read after writing to check if update was successful

Returns

0 if byte was written successfully or -1 on fail

3.3.2.42 int16_t DpsClass::writeByteBitfield (uint8_t *data*, RegMask_t *regMask*) [protected]

updates a bit field of the sensor without checking

Parameters

regMask	Mask of the register that has to be updated
data	BitValues that will be written to the register

Returns

0 if byte was written successfully or -1 on fail

3.3.2.43 int16_t DpsClass::writeByteBitfield (uint8_t *data*, uint8_t *regAddress*, uint8_t *mask*, uint8_t *shift*, uint8_t *check*) [protected]

updates a bit field of the sensor

regMask: Mask of the register that has to be updated data: BitValues that will be written to the register check↩
: enables/disables check after writing; 0 disables check. if check fails, -1 will be returned

Returns

0 if byte was written successfully or -1 on fail

3.3.2.44 int16_t DpsClass::writeByteSpi (uint8_t *regAddress,* uint8_t *data,* uint8_t *check*) [protected]

writes a byte to a register of the sensor via SPI

Parameters

regAdress	Address of the register that has to be updated
data	Byte that will be written to the register
check	If this is true, register content will be read after writing to check if update was successful

Returns

0 if byte was written successfully or -1 on fail

Index